CITS3002 Project Report Semester 1 2021
Martin Nastevski – 21964614

How would I scale up my solution to handle many more clients?
Due to my implementation running at least 4 threads per game, one for the main thread, which accepts new connections, one for the client managing thread, and one thread for each connection. If there were more than 2 players, there would be one extra thread for each other connection whether it would be a player or spectator.

- Firstly, I could see how many threads this server could handle, and to do this I would see if any large number of clients connected to the one server all got game time of some kind instead of just spectating time, and if there was any delay between communication between clients and server.
- Secondly, I could instead make a server per set number of clients, so that every time there is at least two players, a server could run a game between, however this implementation would need to dynamically change based on if there are also enough clients for a single four player game instead of two servers running two different two payer games, however this would need a server manager that has access to all servers and all clients. This could be done by either automatically merging clients to be handled by a different server, which would use that managing server, or by receiving input from clients to determine if they want to be merged into a game with more clients. The second implementation would require more threads, however more games would be played meaning more happy clients.

How could you deal with identical messages arriving simultaneously on the same socket?
Currently my implementation deals with a thread per client, and only accepts input from that client if it is their turn, however receiving two identical messages at the same time on the same socket would cause an error on my server. A way to deal with this would be to use a lock, so one client performs a task, for example pushing an item onto the queue for the manager to deal with, and then the other client would do the same. Since the managing thread would have already dealt with the first message, the second message would not make sense and would not fall into any other conditions which the manager thread checks. For example, if two identical tile placements were sent, the first one would be done, but then the second one would be caught due to either it not being a valid move (with that tile spot now being full), and it not being that current players turn anymore, since the first pass through would have handled a player turn change.

With reference to your project, what are some key differences between designing network programs, and other programs you have developed?
I have developed mobile and web apps, where the programs are hosted on external servers, so the networking management was not done by me. Comparing this project to my other non-network projects, the main difference would be having to deal with multiple instances of the same thing, in a non-object orientated sense. In network programs, it is common to have infinite loops, which endlessly do the same thing. Trying to logically find the correct conditions for each different client working in parallel, along with debugging is difficult, as you are forced to think in a more general way, instead of applying logic for each object instance. Since each client runs in parallel, to debug I must know what they are all doing at any point in time and having the base server code and functions given to us made it easier to implement the multi-client version. In comparison to my non-network programs, the most difficult thing was learning which functions were the right ones to make.

What are the limitations of your current implementation (e.g., scale, performance, complexity)?
My implementation can run a two-player match with any number of spectators, and when that match is finished, it will pick a random order of those clients to play the next match, if there are at least 2 clients. If a client quits during someone else's turn, the game will give that player an extra turn, and the game will run normally carrying on. Game runs very fast, with very little delay due to it being run on multiple threads in parallel. My functions within my managing thread run at different complexities, for example 'get_random_players()' runs at O(n), where n is the number of clients. My 'start_game()' function runs at O(n*(p+t)), where n is the number of clients, p is the number of players, and t is the number of tiles a player can hold. These functions run once per game, however the other functions within the manager thread which run the actual game itself each run between O(n^2) and O(n^3) where n is the number of clients connected, depending on each function. The server and each client run indefinitely.

Are there any other implementations outside the scope of this project you would like to mention?
For this project, I considered implementing a waiting lobby so that if there were many clients connected, that each were guaranteed a game after some determined amount of time. The implementation we were asked for, was to randomly select clients from the connection pool after every game, with the chance of some clients not ever playing a game. Another thing that could be implemented would be a menu, where clients could choose if they wanted to wait, watch, or play.

Notable things to discuss:
My project contains a handful of duplicate code, however not all of it is the same, and some parts cannot be changed. Other parts of this code can most definitely be condensed down for neatness and readability.
Upon disconnects, or sometimes player eliminations, the client will not return the correct current player turn, this is due to the manipulation of my global queue which holds the current player and next players, to fix this I would have to do multiple checks during the eliminations or disconnects to ensure that the player who is disconnecting or being eliminated is not the current player, and to adjust the new global player list accordingly.
Code was run with Python 3.9.4 on a Windows 10 machine. Server with "python3 server.py", and clients with "python3 client.py"