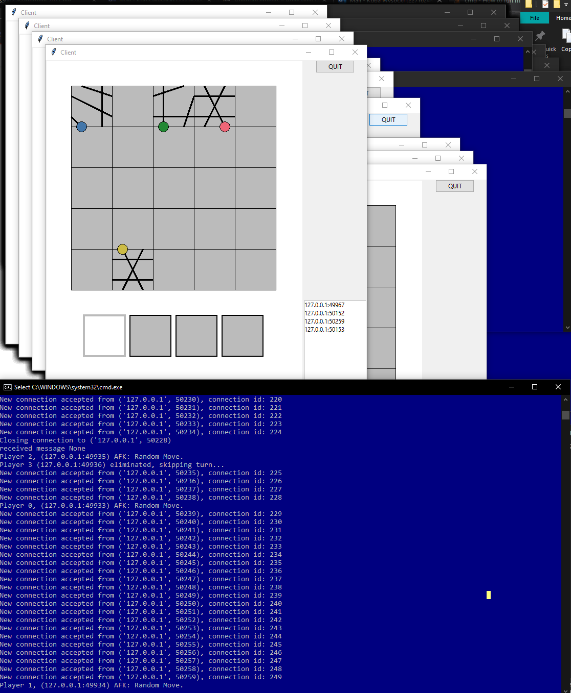
**CITS3002 Computer Networks – Sockets Programming Report**

Jakub Wysocki - 22716248

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

After performing some tests, the server was able to handle over 200 clients connected to it simultaneously, with minor performance drawback. Despite this being many people to play one tiles game and redesigning the flags and game-state checks to happen earlier in the program. This will shave some processing time where the loop goes over each client. To implement more active players, I would redesign the way the server stores data relating to each client. There are multiple lists, dictionaries and flags holding various game-state defining values, which could possibly be reduced to a slimmer data structure. Another method of scaling would have multiple servers running at once, with multiple games in progress; this would allow fewer spectators aimlessly waiting for a new game to start. One server could also be used to have the ability of running multiple games simultaneously. There could also be one central server that redirects new clients to empty sub-servers that are waiting for more players. This would be a more efficient and user-friendly way of handling a much larger number of clients.

<- 250 connected clients

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

Using the abstraction given by the TCP protocol, the server receives complete, non-collided messages in its buffer. So, from the perspective of my program, two identical messages would arrive at the socket, one after another. Using my implementation of the socket’s library, the first message would be received, then processed, then all the other available sockets are handled, finally going back to the duplicated message. There are checks within the server to test whether that new, repeated message is to be interpreted as a valid turn or message. As for the listening port, each new incoming connection is handled straight away, and the other simultaneous message would be ignored or handled as an additional client. Additionally, the packets have headers so our application layer program can discern if this is the packet for its current use. This server application could be doing various other tasks, rather than just the tiles game, on the host machine and might use the same socket for communication. The same message could also be for two different “tasks” on the server application.

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

I have run into many unique design problems when working on this project. One significant difference was how the program had to be updated while reading different data from multiple different sources; this required the use of sockets, and in my case using selectors to manage this data. Another challenge was the ambiguity of the current states of the program and how many different states there could be. For instance, the data received from the clients may cause them to be eliminated at any moment, and the server must react appropriately and run different possible events. Similarly, each different client may have their own states that are important to maintaining the correct game-state, and every client must be accounted for. This program also must run continuously, instead of starting, executing, closing, like many other programs I have written. The server must be set to listen and constantly monitor its connections for some unset amount of time. Additionally, there is no real end goal or final desired state for this program, it must just react to messages sent by clients and respond accordingly until it has been closed.

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

The first issue with performance would be in the *make\_valid\_move ()* function, where if a random move is needed, the algorithm is simply a randomized brute force strategy. This ensures a random move is performed for the player and that the move is valid. Given the small number of permutations of placement (given by the size of the board), this inefficiency is not noticeable in practice. Despite meeting the project specifications, this algorithm has an inefficient **worse-case** complexity of O(N4). The time to find a valid location is an inverse relationship and increases as the number of valid locations on the board are decreased. This could be improved by using known positions and the state of the board and the clients’ past moves. In practice however, finding a move is almost instant. Given the server runs locally on the machine which the clients connect to, it is limited by the number of ports per the local address, at 65536 maximum clients. Similarly, it is limited to the host machine it is on and does not connect via the internet. The way sockets are handles could have been done using event-driver programming or threading instead of selectors. Selectors works well for this application, as shown above. However, there are times where the program loops over each socket to check its status and how its current game state may change, which wastes CPU time. This could be solved using another multi-connection socket strategy. Finally, the number of global variables could be reduced to a slimmer data structure, which would take up less room and reduce complexity.

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

There seems to be an issue where if a player is disconnected before they are given a turn to play, the clients crash. This seems to be because they do not use Player Joined messages to disconnect rather than the Turns. This design issue could best be fixed with a simple update to the client, but this lies outside the server-side scope of the project.

1. *Any other notable things to discuss.*

Some parts of the code were left as-is for future scalability of the code. These take up negligible space, nor add much more complexity, and are rather inbuilt to existing structures. The features these would add fall outside the scope of this project, and such were not used. All four tiers of the project were successfully implemented in my final solution, which is bundled with this report. Spectators receive the *game start message*, to catch them up on **each** event that has happened. This had some issues with the given tester, but no specific definition was given in specifications, and was left in the final implementation.