CITS3002 Project Report 2021

Kane Alexander 22710428

# Player Scalability, Multiple Messages and Limitations

The server was designed with scalability in mind, as all the given game variables are held within a single class object and only the socket addresses have a global scope. This ultimately allows multiple independent game objects to be populated in the same run time, all accessing the array of global connections. As a proof of concept, the game variable can be replaced with an array of games each having the same desired functionality as the initial server. This concept can be generalized to n games, where each game can communicate with the server and access its own independent game object. Scaling the server by adding more game objects would work for a substantial number of connections, however, would ultimately fail if the number of connections increases to disproportionate amounts.

To handle an overwhelming number of clients, multiple servers each with own independent socket would need to be used. This is because the actual games themselves, although contained within independent objects, communicate through the same socket. The controlling while loop of the server uses the selector module to monitor all connections, and act accordingly to the data received. This is equivalent to a single threaded server, with the same downfall that if multiple connections are all sending lots of data, a backlog could form. This could result in a higher latency for the clients as the server has to work through the packets one by one. In the worst-case scenario, the kernel could start dropping connections if the server cannot handle the incoming data fast enough. Employing multithreading would allow communication of more then one client at a time and could increase the performance of a single server dramatically.

The project requirements desire a maximum of 4 player concurrent players per game and only 1 game running, and thus, a single threaded server seemed more then adequate. Implementing threading or adding multiple servers seemed well out of the scope of the project. A single threaded server makes dealing with concurrent messages relatively straight forward as only one can be acted on at a time. If two of the same messages are received, the server will execute the message first selected by the selector module and update the game accordingly. The repeat message will then likely be ignored and dumped as the server is updated and acknowledges that the connection has already made its game move (given the first message was a valid move).