Satisfying Requirements

Stateful

My implementation of the spec implements the stateful requirement in two different places. Protocol state must both be stored on the client, and on the server. Therefore, I made an Enum to represent the protocol state (located in internal_types/states.py). This enum is used in both the client and server applications. In the client, the current Protocol State is stored as the attribute __protocol_state on the Client Object (client/client.py). In the server, the current Protocol State is stored as the attribute __current_state on the Session object (server/user_sesion.py). Both the Session and the Client objects have an __on_data method which call different callbacks when the socket gets data, based on which protocol state they are in. If a message is received in an invalid protocol state, the connection is closed in most cases (there is one case, the server receiving MakeMove while in GameEnd which does not constitute an error and is not part of the DFA because it only happens because the client and server are out of sync).

Concurrent

The server is using a select model in order to be concurrent. I am using the library pyuv (based on the C library libuv) to create an asynchronous event loop and bind sockets to it. The TCP objects (sockets) are bound to an event loop. The event binding is done when the .start() method is called on various objects, the TCP sockets are bound in the Server object (server/server.py) and the Session objects (server/user_session.py). One of these objects receiving a connection or data constitutes an “event” happening, and calls a function or a “callback”. Because of this model, the server is never waiting on a single client to connect - it only reacts to events rather than expecting them. This allows many users to connect and send or receive messages effectively at the same time. When many users make many requests, it alternates between serving different users based on which ones have data available from the network (using select).

Service

The server binds to a default port of TCP 8864. This value is picked by passing the “port” argument to the Server object, and defaults to 8864 in the program options (server/__main__.py). The connection is actually bound to the port in the method Server.start() (server/server.py). The client connects to this port in the Client.start() method (client/client.py).

Client

The Client class is in the client folder, specifically in the file client/client.py. When running the “client” command to run the client, the argument --host-addr can either take an ip address, a hostname or can not be specified to use service discovery to find the server.

UI

There is a very basic user interface for my application in the Interface class (client/interface.py). Both the client and the server have logging as well which for the server defaults to INFO level (showing all errors, warnings and informational messages) and for the client defaults to WARNING level (showing errors and warnings only). If the --verbose option is added to either it will set the logging level to DEBUG (show all log messages, including logging all messages sent back and forth between the client and server). The server can also specify the --quiet option to set the level to WARNING.