# Server Client Battleship

## By: Andrew Gazeley

## For: CS313 Computer Networks

## Objectives

The objective of my project was to create a client-server based application for playing battleship across a network. Using only lower level libraries in python to send and receive commands such as connection info, requests for open lobbies, join/create game requests and move request. The server handles the back-end game logic and then the client is responsible for the generation and display of the info in an organized and hopefully graphical manner.

## Plan

My plan is to build two scripts relying on UDP socket based communication to send info back and forth. I am sending the data in JSON notation so that it can be serialized back into dictionary objects on the end-points. The message format will be as show below in the table.

Client Requests

|  |  |  |  |
| --- | --- | --- | --- |
| Game\_id | Player | Req\_type | Req\_msg |
| Game you are inquiring about | Player inquiring | What category of request does it fit into | The content of the reques |

Server Responses

|  |  |  |
| --- | --- | --- |
| Game\_id | Type | Msg |
| This field lets the client know what game the server is talking about. Is used as a key between server/client | What type of info is the server sending back | Info to be handled by client. Usually adjusts view of client somehow. |

Samples:

Game made confirmation example

Client request



Server responds with confirmation



There are eight different request types that the server can handle from the client. These are “new\_game”, ”connect”, “data”, ”join\_game” ,”move”, ”lobby\_rdy”, ”lobby\_exit”, and “board\_setup”. Each of these creates a specific message to reply to the client with or do nothing and wait for more data.

There were five components to my project that I wanted to incorporate and those were simple connectivity, a lobby system for joining games stored on the server, and playing games. Each of these protocols plays into the others. By that I mean that playing a game requires creating a new game which initiates the lobby protocol when the server creates the new game within its own lobby. See below for how individual protocols behave.

## Protocols

Client Connect Process

Client Connects to Server Socket on Host:Port

Client sends ‘connect’ message

Server sends ‘conn\_request’ reply with either 0 or 1 for confirmation.

Client Sends ‘Data’ message with player info.

Server sends back current lobby data for the server. List of games in (id,# players) form.

End

New Game Process

Client sends ‘new\_game’ request

Server creates lobby for game with new unique game\_id

Server sends reply of ‘game\_made’ with either a 0 or 1 for confirmation.

Start Lobby Rdy process

Join Game Process

Client sends ‘join\_game’ request with game\_id and player info of the requested game to join

Server sees if game has opening in it. Adds player to that game lobby.

Server sends reply of ‘join\_result’ with either 0 or 1 for confirmation

Start Lobby Rdy process

Lobby\_Rdy Process

Path 1:

Client sends ‘lobby\_rdy’ request to server.

Server responds with either ‘game\_start’ or ‘lobby-resp’ and info on what players are ready.

Client waits for other players to be ready. Client is stuck here until game starts currently.

Start Board Setup process

Path 2:

Client sends ‘lobby\_exit’ request to server.

Server responds with new list of lobbies to join and removes player from the lobby and game. If lobby is empty the server deletes the lobby.

Board Setup Process

Client receives ‘game\_start’ request from server.

Client chooses a board setup. Need to implement a board editor module but right now we have two static boards.

Client sends ‘board\_setup’ request to server with array of board info.

Server waits til both boards are in and then starts ‘move’ process with client1.

Move Process

Client1 receives ‘move\_req’ or ‘turn’ from server.

Client1 sends ‘move’ to server.

Server computes what happened during the move and sends ‘move\_result’ request to Client1 and sends ‘turn’ request to client2. If the server determines someone won it sends ‘win’ request to winner and ‘lose’ request to loser clients. Client returns to menu after this by sending a ‘data’ request to server. Server deletes game and lobby.

## What I learned

During the process of creating this application I learned quite a lot about the design and implementation of client server applications. The first lesson I had was that TCP is not ideal for dealing with multiple clients as it is connection based. I first implemented this app using TCP and then had to change to UDP when I realized how much work it would be to implement handling of the connections as opposed to just handling the games and users. The second thing I learned is that you need very defined protocols and formats for these protocols. I was pretty confused with my own code until I sat down and made a whole matrix and flow pattern for the individual protocols. The third and final thing I learned is that implementing a network protocol is a lot easier than combining that network protocol with a front end graphics service. I attempted to add a GUI for the client but ended up spending two whole days trying to learn graphics libraries when I could have spent that making more robust protocols and adding features to the server.

## Going forward

My plan for this project going forward is to use it to explore and learn graphics. I will be going to England for the Tech. in UK Trip and we will be working on games while we are there, so I figured I could complete the joining of client logic with client graphics. It needs a few more protocols and the ones that are implemented could me modulated and tested more thoroughly.