Jason Carrete

Prof. Arias

CMPT 220L-204

4 May 2018

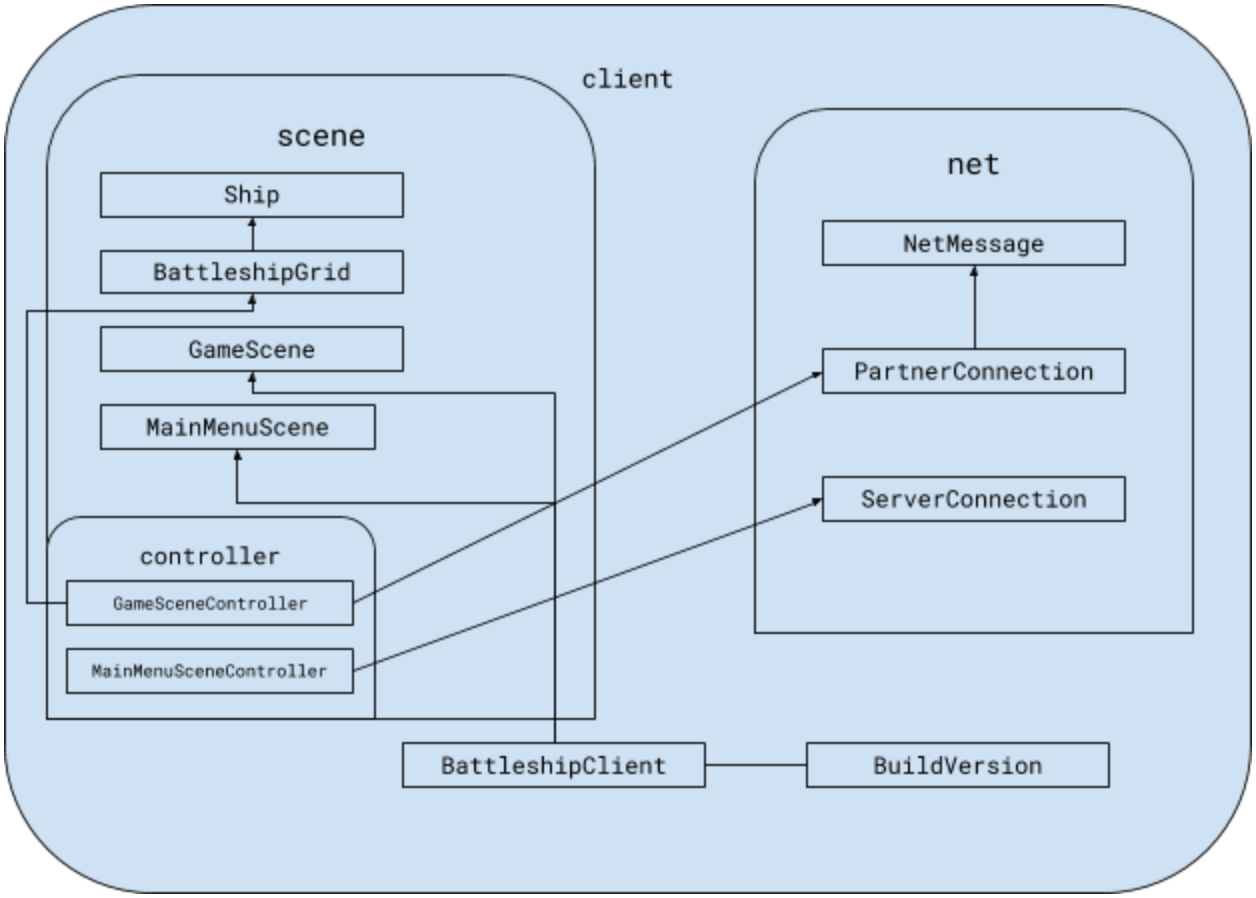
Final Writeup

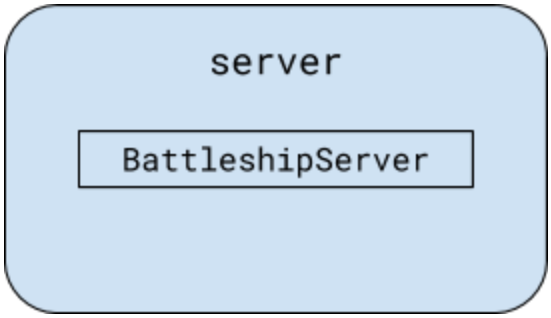
Abstract: My software development project is an implementation of the Battleship™ board game. The game is implemented to work with two players over a network.

Introduction: I chose to make a Battleship™ game because I have never created my own Battleship™ implementation and wanted to work with network sockets. I have had some experience before with network sockets in the past and I wanted to continue to try to use them to further understand how they work and come up with better solutions for handling network messages.

Detailed System Description: My application is split up into two separate modules: client and server. The client module is what a user who wants to play the game would run. The client must be able to communicate with the server in order to find a partner to play the game with. After the client has a partner, then the Battleship™ game can be played between the two clients. The server module runs on a remote computer and facilitates connections between clients. Its only job is to wait for clients to connect and then match two clients together and tell them to connect to each other. After that is done, the server ends communication with the clients as the clients can now communicate with each other without the need of a server. I am able to achieve this through a technique called network hole punching.

When a client receives a message, it is received in a separate thread and put into a queue. Whenever I need to check if a message is received, I attach a callback to a CompletableFuture object which calls my callback with the NetMessage details. The NetMessages are fetched off of the queue and passed along to the CompletableFuture. All of this functionality is neatly wrapped in the PartnerConnection class so as to provide a simple API for the programmer (me) to use for sending and receiving messages from to and from the peer to peer partner. The entirety of the game is played based on a back and forth communication of messages sent between both clients. One client will send a MSG_FIRE and the other will respond with MSG_FIRE_RESULT. When one of the clients realizes they have lost, then they send out a MSG_LOSE to their partner to let them know that the game is over.





(There is a common module that only has some logging utilities)

(Classes in the scene.controller package are linked to their respective scenes via fxml)

Requirements: My solution is solving the problem of multiplayer by using a TCP network sockets to communicate between two clients. Before players start playing, each players grid needs to be properly set up. Each player will randomly place their battleships. My algorithm for randomly placing the ships involves choosing a random orientation for each ship and then generating a coordinate for the ship to be places at. If the ship can't be placed there then the ship's orientation and position are chosen again it tries to be placed somewhere else. After both players place their ships, a network message is sent to each client from each other and the game is played starting with the host player.

Literature Survey: I haven't really done any extensive research on other battleship implementations. The only source I have for how the game of Battleship™ works is the wikipedia article for battleship and my own experience playing the game.

User Manual: The application is very simple to use. The client has a simple user interface to find a multiplayer game and play. The UI is pretty self explanatory and if you have ever played the game of Battleship™ you will have no trouble figuring out how to guess squares where you think your opponents ships are.

Before both players can start the game, the ships must be placed. This is done by pressing the random button which will randomly place your ships and let your partner know that you are ready to play. When it is your turn, you must select a cell on the upper half of the BattleshipGrid to fire at. The cell you select as a potential target will highlight orange. Once selected, you then press the fire button to confirm to your partner where you are firing. The cell will then highlight either green or red indicating a hit or miss respectively. When you sink an opponent's ship, you will be notified specifically that a ship has been sunk. The game ends when either you or your opponent have all of your ships sunk. Whoever loses all their ships, loses the game and will be notified.

Conclusion: Overall, my application is just a simple implementation of the once popular board game Battleship™. The only purpose this application serves is to mimic the game of Battleship™ in a more modern digital format.

References/Bibliography:

Hasbro. "Battleship (Game)." Wikipedia, Wikimedia Foundation, 1 Apr. 2018, en.wikipedia.org/wiki/Battleship_(game).