Joseph DiPalma, Annan Miao, and Ben Xu

May 2, 2017

Professor Brian King

CSCI 205 Final Project

Design Manual

Introduction

User Stories (Implemented)

1. **As a user I want to be able to place my ships on my board so that I can start the game.**

   Relevant classes: Player, Board, View, Ship

   Each player has a board object and an array of Ship objects. Each ship has its own ship type and its locations an ArrayList. The element in the ArrayLists are size 2 arrays to represent 2d coordinates. The view class displays all of this graphically.

2. **As a user I want to be able to view where my opponent has attacked my ships so that I can make my next move.**

   Relevant classes: Player, Board, View

   Each player has their own board keeping track of enemy attacks. The view class displays this graphically.

3. **As a user I want a view of the hits and misses I have made on my opponent's board so that I can plan where to make my next move.**

   Relevant classes: Player, Board, View

   Each player has a enemy board, which keep tracks of all the attacks made toward opponent. Each attack hit is represented as a red square, and miss is represented as a white square. The view class display all of this graphically.

4. **As a user I want to be able to connect to the network so I can play with another person on a different computer.**

   Relevant classes: Server, Client, GameStartView

   The Server will host the game and display its server ip address and wait for the other user who act as client to connect together. This is handled graphically by GameStartView.

5. **As a user I want to be able to use the game without lag so it is an enjoyable experience.**

   Relevant classes: All

   All the tasks are being separated into different threads. The Server, Client, and main GUI are all separated into their own threads to ensure no lag occurs.

OOD

[INSERT CRC CARD AND UML HERE]

Our design is basically break everything down into individual objects that represent the physical objects for Battleship. We started of with listing what "objects" are needed for the game to function as a general concept, which are Board, Ship, Player.

Then we started to consider how each of these objects interact based on our user stores. Since we want to be able keep track of each player's attack, so each Player object have two Board Objects. Since we also need to keep track of the ships, so each Player object have an array of ships, and an enumeration type is used to represent different types of ships.

All of these objects are enough to build the Battleship game, so the MVC design pattern is just using these objects to build the game graphically. In order to have a proper sync, we decided to have one model to manage both players to keep things consistent through the game. Hence, the model takes in two players, and would be pass into the controller, and then this controller would be pass into the server and client to run the game over the network.

We following server/client model using socket api for networking because none of us have any prior experience, and socket api is the only one that made sense to us when we research online. The idea for networking goes like this:

1. User 1 host a game, which display its server ip address.
2. User 2 obtain server ip address, enter it into a text box and connect to the user 1's server host.
3. Once the connection is successful, both users would see their respective views and the game would start.

In order to ensure smooth game play game experience, we incorporate multithreading within the design. Since Battleship is a turn based game, we decided to make game interaction run within the JavaFX thread alongside the GUI while the server and the client runs within their own thread, respectively.