Software Design Document

Kaitlin Dosch

Connor May

Dominic Ravagnani

Diane Smith

Julia VanLandingham

Introduction

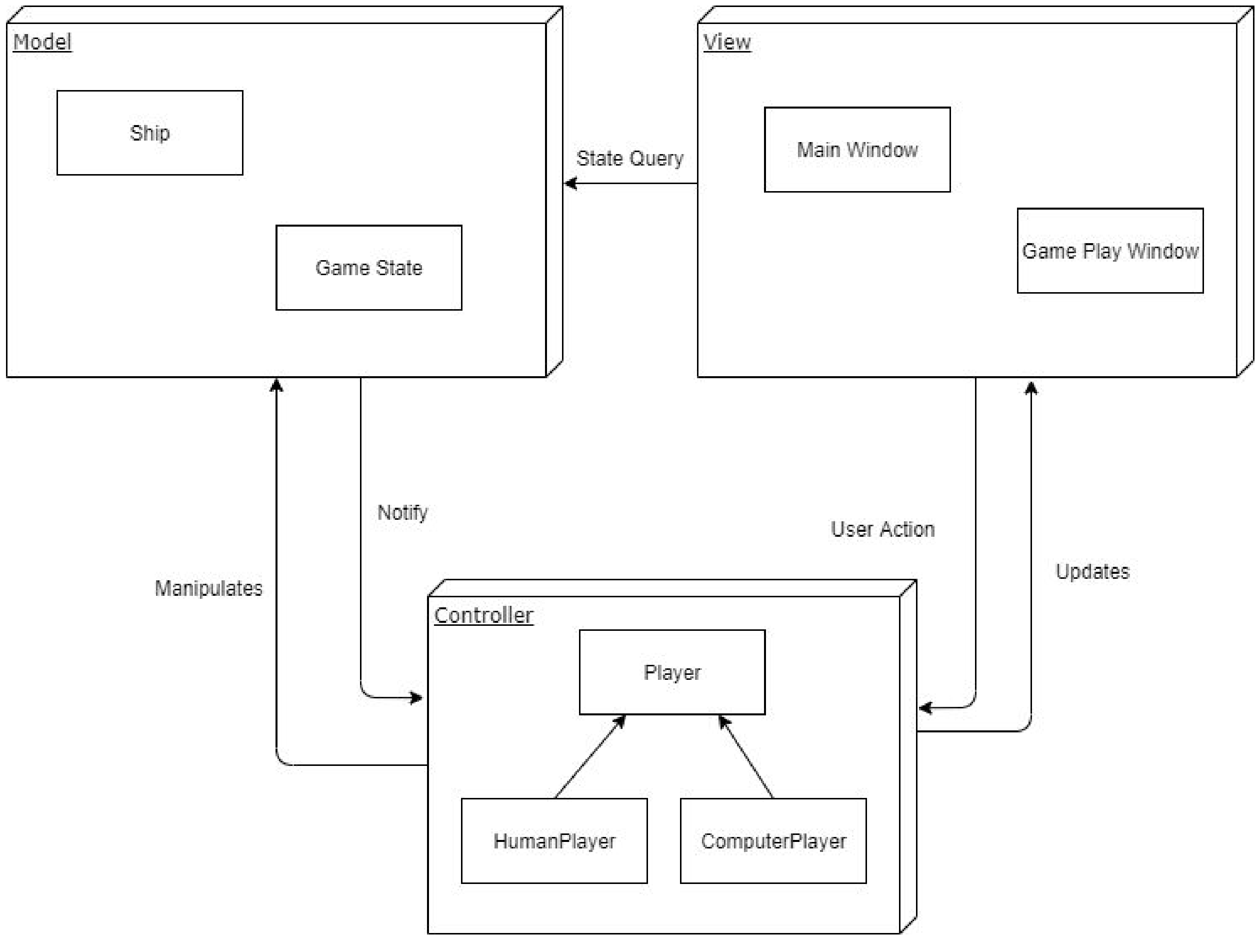
The purpose of this document is to guide the implementation phase of creating an online game of Battleship. The purpose of the product is to create a way for a person to play a game of Battleship by themselves or for two people who are unable to play in person (either due to not having the physical game or not being in the same physical location) to play a game of Battleship together via the Internet. This project is completed in partial fulfillment of the requirements for Otterbein University's COMP 3100 course and has been implemented under the guidance of a university professor. This product is intended for 1-2 players who want to play the game Battleship. This document is intended to aid the development team through the implementation phase.

The remainder of this document outlines the implementation of the various classes that make up this product. We begin with specifying the overall architecture of the system and then go on to specify the design of the data structures and the classes that will be used in the implementation. We also include depictions of the user interface as well as a small working prototype. Lastly, milestones for the implementation phase are laid out and depicted using a Gantt chart.

System Architecture

Overall, the software will be modeled using the Model-View-Controller architecture. The model classes will include Ship, which has all the attributes and methods associated with a ship, and GameState, which represents the current state of the user's, or enemy's, board. The view will include the MainWindow class and the GamePlayWindow class. The MainWindow class is the starting screen of the games (see the Interface Design section for a more detailed explanation). The GamePlayWindow class contains two board objects, each of which is a visual depiction of the user's, or the enemy's, board. This class also controls the ship placement phase. The controllers will be an abstract Player class, which is extended by a HumanPlayer class and a ComputerPlayer class. The Player class will coordinate all the interactions between the model and the view. Both the HumanPlayer and the ComputerPlayer classes house the methods for gameplay decisions and communication between two users. The HumanPlayer class also contains a Networking class that will handle all the network interactions between the two players. Below is a block diagram depicting the basic interactions between the components.

(Block Diagram)

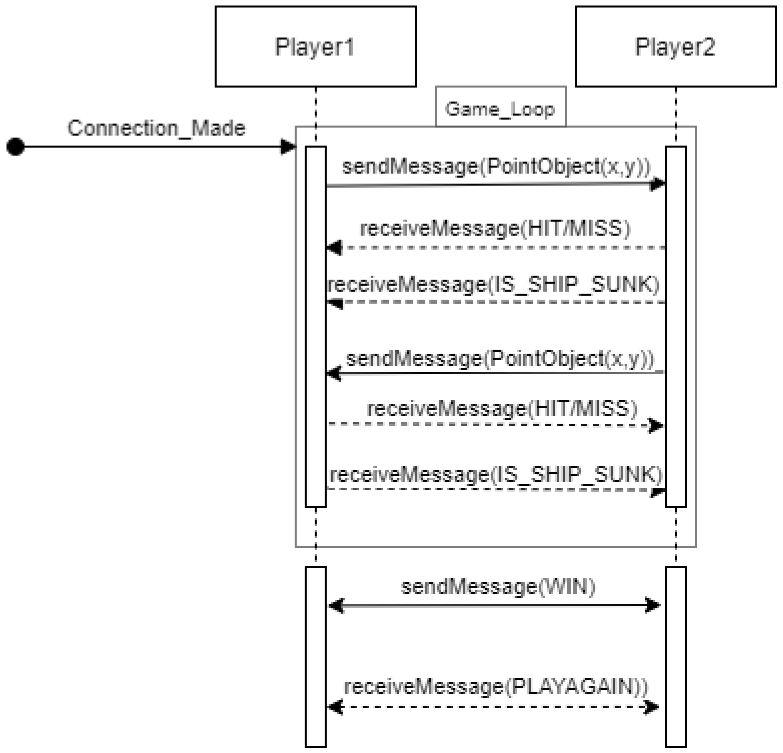


Data Design

This section outlines the basic data structures that will be employed in this product. We define an enum called Tile that has four values that represent the different states that a given position on the board could have. Each Ship class has members of type Point that define its start and end coordinates. In the GameState class, we use a 2D array with Tile values to represent the current state of a player's game. The Board class uses a 2D array of JButtons to visually depict what is stored in the GameState class.

For the two-player mode, the network interaction is described by the figure below. The connection will be created as Player 1 taking the role of server and Player 2 taking the role of the client. After the connection is created and the game is started, Player 1 will start by sending a guess selected by the player across the socket. Player 2's game will respond with whether the guess was a hit or miss. The turn will change and then Player 2 will guess and Player 1 will respond with a hit or miss or if a ship was sunk. This will continue in a loop until there is a winner declared; a player wins when all of their opponent's ships have been sunk. The winner will notify the other player and then wait for a PlayAgain response to start another game or for the connection to end.

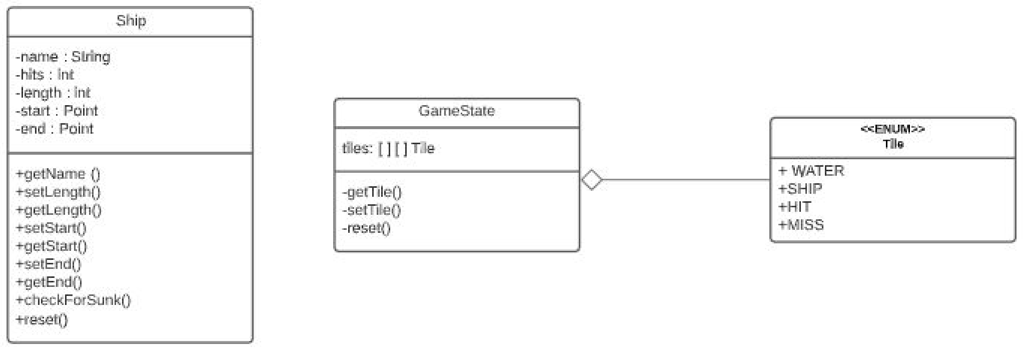
Sequence Diagram


Player1
Player2
Game_Loop
Connection_Made
sendMessage(PointObject(x,y))
receiveMessage(HIT/MISS)
receiveMessage(IS_SHIP_SUNK)
sendMessage(PointObject(x,y))
receiveMessage(HIT/MISS)
receiveMessage(IS_SHIP_SUNK)
sendMessage(WIN)
receiveMessage(PLAYAGAIN))

Component Design

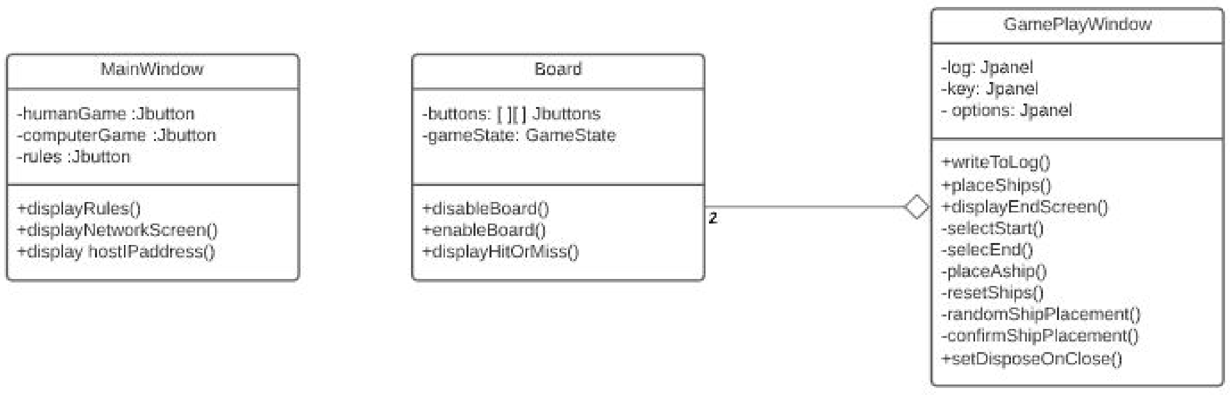
The model holds the Ship class which implements functional requirements 2.10 and 2.12. We have an enum called Tile that represents the states that a position on the board can have. The GameState has a 2D array of Tile values that represents the current state of a player's board and implements functional requirement 2.11.

MODEL



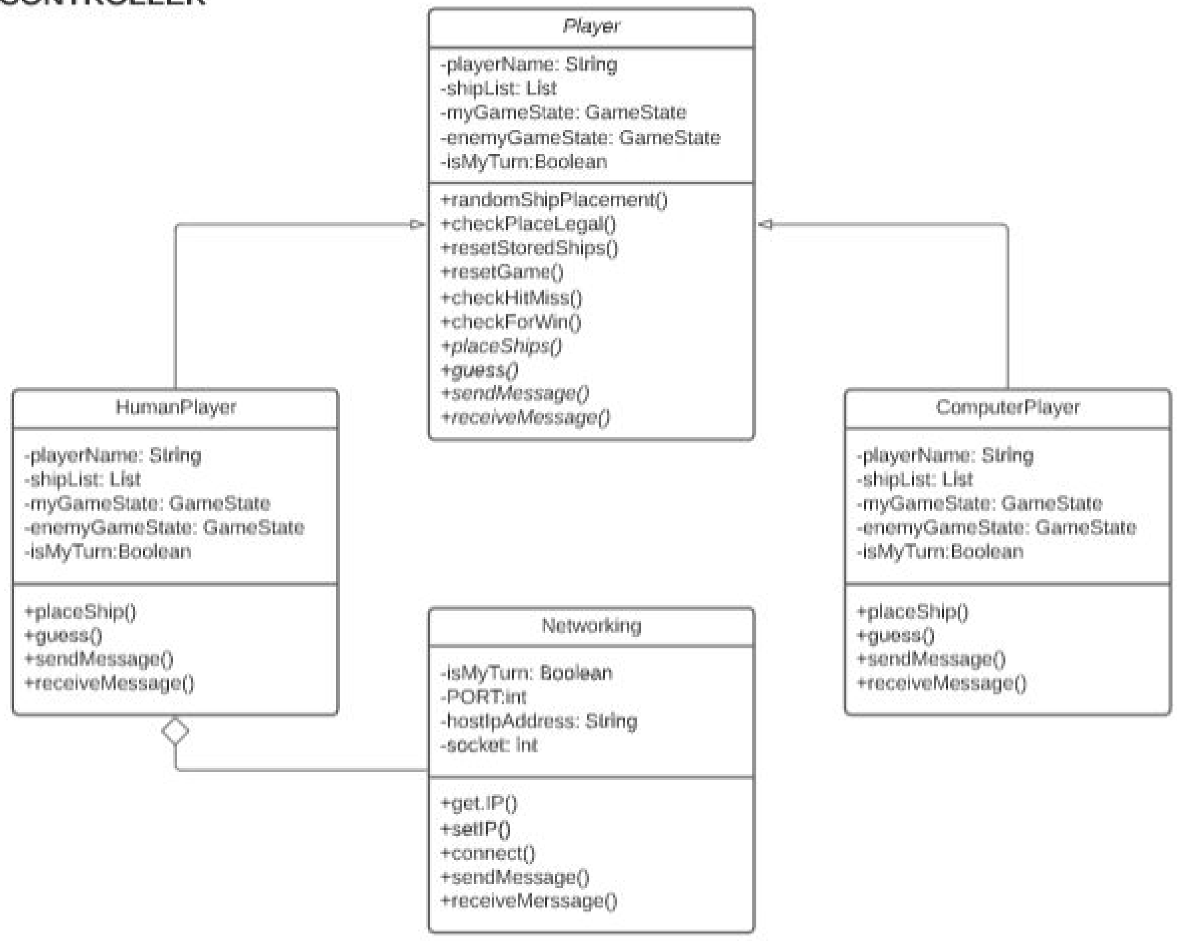
The MainWindow class will run the main menu, start the network connection process, and implement functional requirements 1.1-1.8, and 2.14. The Board class visually represents the current GameState and implements functional requirements 1.21, 2.8, and 2.9. The GamePlayWindow class holds two Board objects; one for the current player's board and one for the opponent's board. This will show all of the results from gameplay. It will also display a window at the end of the game with various options and will implement functional requirements 1.9-1.20, 1.22-1.24, 2.14, and 2.15.

VIEW



The Networking class establishes connections between two players as well as sends and receives messages throughout the game and implements functional requirements 3.1-3.8. The abstract Player class houses all of the view and model classes and controls the interactions between them during the game and implements functional requirements 2.2, 2.4-2.7, 2.13, and 2.15. This class has abstract methods for placing ships, guessing a tile, and sending/receiving messages to/from the opponent. This allows the HumanPlayer and ComputerPlayer classes to implement these functions in different ways that will interact properly with each other. The HumanPlayer class extends the Player class implementing its abstract methods and functional requirement 2.3. It also has a Networking object to handle any network functionality. The ComputerPlayer class extends Player, which implements its abstract methods and a strategy for an AI opponent implementing functional requirements 2.1 and 2.3.

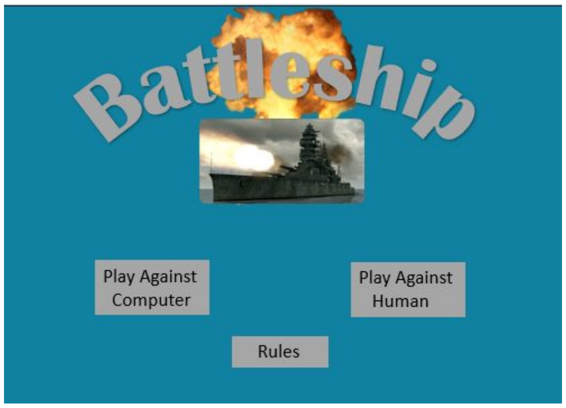
CONTROLLER



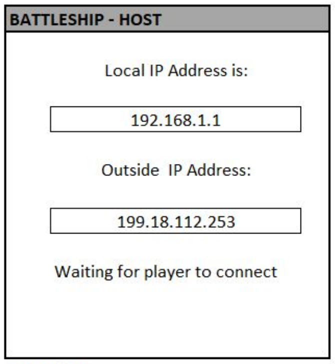
Interface Design

On the menu screen, when the “Rules” button is clicked a separate window will appear displaying the rules. When the “Play Against Computer” button is clicked a new game between the user and an AI opponent will begin. When the “Play Against Human” button is clicked the user will be redirected to the networking screen. A pop-up window will show up, giving the user the option to be the client or the host (screen not depicted). The following screen will be based upon the user’s decision and is shown below. On the client’s networking screen, a “Connect” button will establish the connection between the two users and start the game.

(menu screen and networking screen)



Networking Screen


BATTLESHIP - HOST

Local IP Address is:

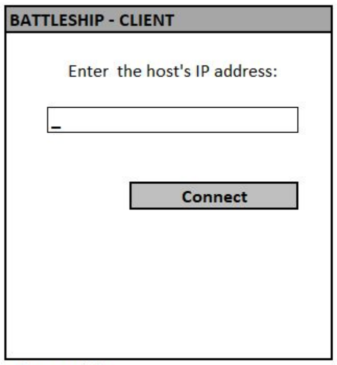
192.168.1.1

Outside IP Address:

199.18.112.253

Waiting for player to connect

Host View


BATTLESHIP - CLIENT

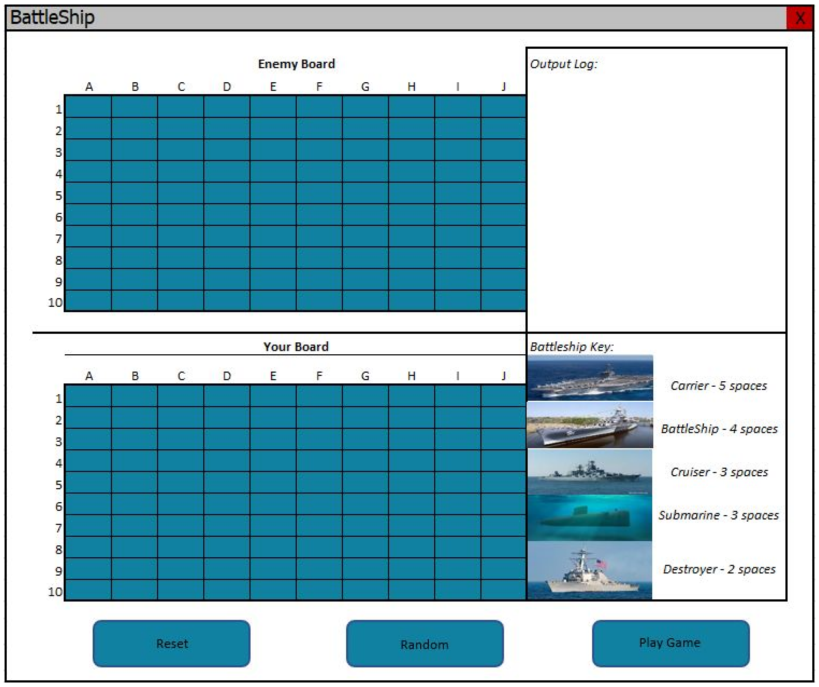
Enter the host's IP address:

_

Player to join

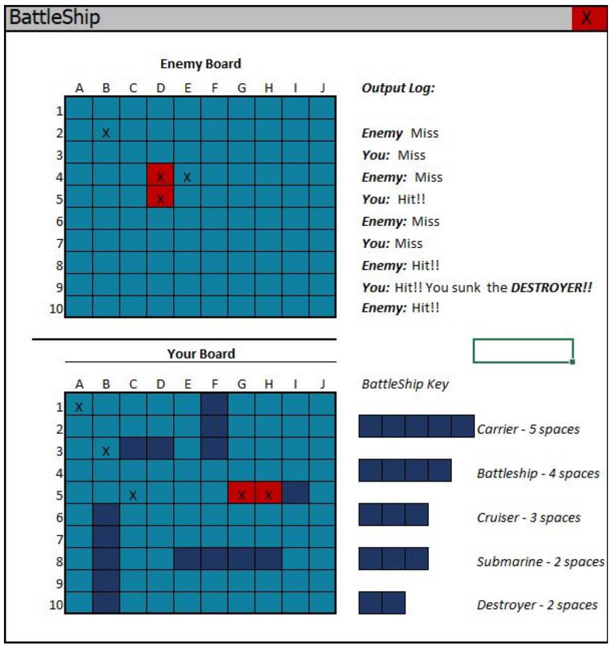
During the ship placement phase, the buttons on the enemy board are all disabled. The buttons on the player's board are enabled so that the player can place their ships on the board. They are then disabled when a ship is placed so that no ships are placed on top of each other.

Game board during ship placement



During gameplay, all of the buttons on the player's board are disabled. The buttons on the enemy's board are enabled on the player's turn and disabled on the enemy's turn. When the player clicks on the enemy's board to make a guess, the icon on the clicked button changes to show whether it was a hit or a miss. The clicked button then permanently disables. When the enemy makes a guess, the icon of the corresponding button on the player's board changes to reflect whether the guess was a hit or a miss.

Game board during gameplay



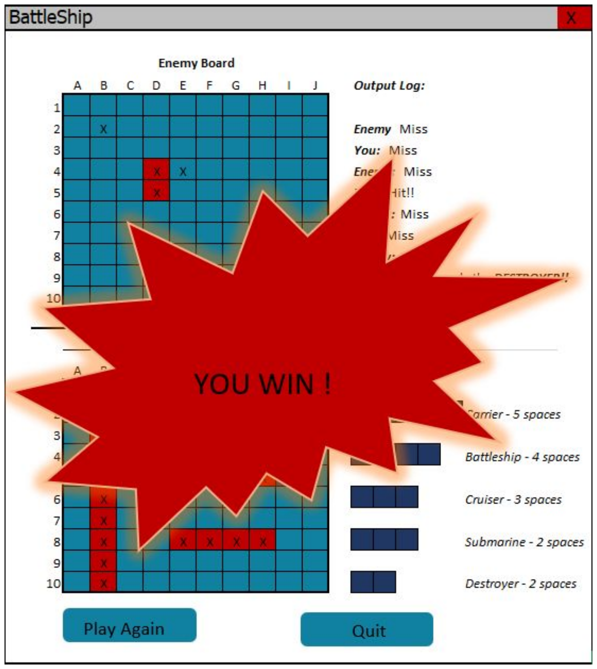
Output Log:

Enemy Miss
You: Miss
Enemy: Miss
You: Hit!!
Enemy: Miss
You: Miss
Enemy: Hit!!
You: Hit!! You sunk the ***DESTROYER!!***
Enemy: Hit!!

BattleShip Key

Carrier - 5 spaces
Battleship - 4 spaces
Cruiser - 3 spaces
Submarine - 2 spaces
Destroyer - 2 spaces

Game Board - Win view



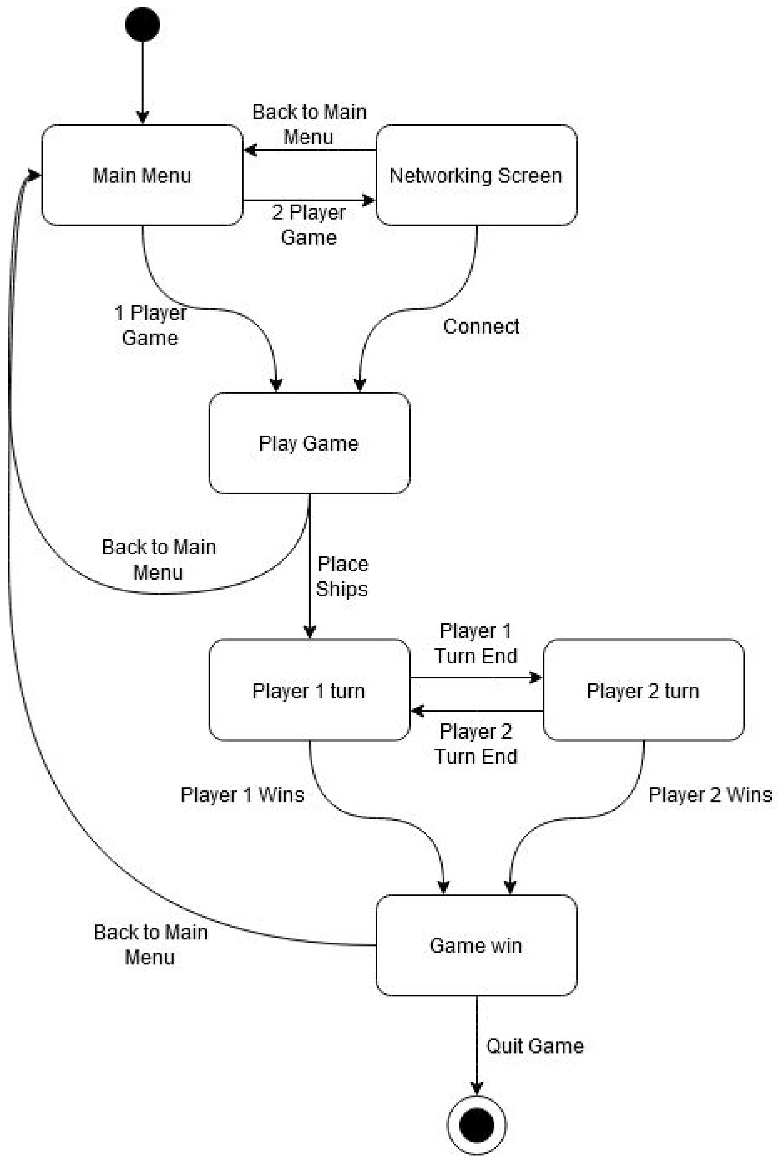
Output Log:

Enemy Miss
You: Miss
Ene Miss
Hit!!
: Miss
Miss

Carrier - 5 spaces
Battleship - 4 spaces
Cruiser - 3 spaces
Submarine - 2 spaces
Destroyer - 2 spaces

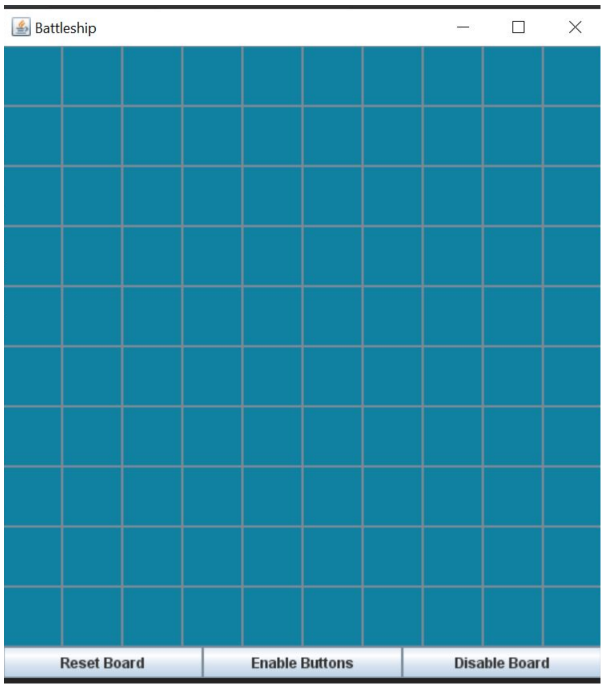
This diagram maps the states that the program can be in and which states transition into each other.

State Diagram

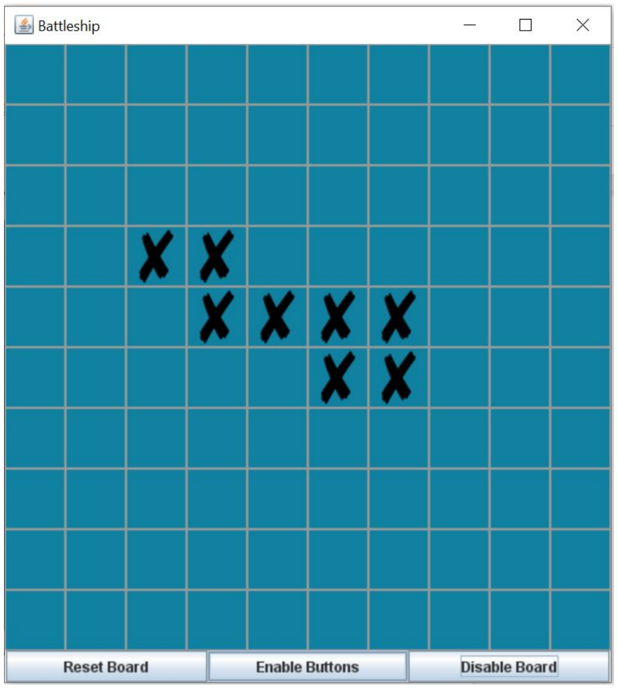


Prototype

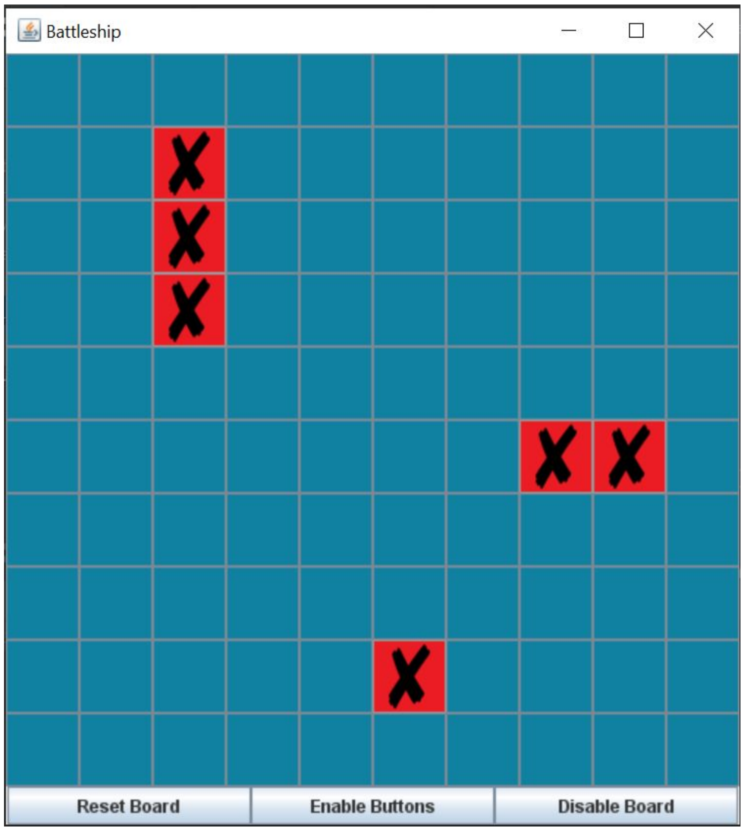
The prototype models the functions of a single Board object and its interactions with the GameState and Player objects. The Player class is left unimplemented for this prototype. The Board class houses the buttons that make up a player's board, a reference to the player's GameState, and the methods for enabling and disabling buttons. The enableButtons method enables only the buttons that have not already been guessed. For this prototype, a panel of options has been added for interaction with the board. The checkForHit method is left unimplemented and thus in its current form will only ever display hits or misses depending on if the return value is set to true or false respectively. Below are screenshots of the prototype in different states. The source code can be found in our repository on GitHub and can be run using the main method in the Player class.



Board at startup



Board with some spaces missed



Board with some spaces hit

Tasks & Milestones

Task 1 - 2.10, 2.12 - Make ship class	10/16/2020
Task 2 - 1.21, 2.8, 2.9, 2.11 - GameState and Board classes	10/16/2020
Task 3 - 1.1-1.8, 2.14 - Menu window with popups	10/16/2020
Task 4 - 1.9, 1.11-1.13, 1.15-1.19, 2.4-2.6, 2.15 - Ship placement	10/23/2020
Task 5 - 3.1, 3.2 - Network connection	10/23/2020
Milestone 1 - Place Ships on Board	
Task 6 - 1.10, 1.20-1.22, 2.2, 2.7, 2.13 - Game play	10/30/2020
Task 7 - 2.1, 2.3, 3.3-3.8 - Finish HumanPlayer and ComputerPlayer 10/30/2020	
Milestone 2 - Playable game	
Task 8 - 1.14, 1.23, 1.24 - End game options	11/6/2020
Milestone 3 - Finished Implementation	

Battleship Gantt Chart

1

▨ Plan Duration

ACTIVITY	PLAN START	PLAN DURATION	Weeks				
			1	2	3	4	5
Task 1	1	1	▨				
Task 2	1	1	▨				
Task 3	1	1	▨				
Task 4	2	1		▨			
Task 5	2	1		▨			
Task 6	3	1			▨		
Task 7	3	1			▨		
Task 8	4	1				▨	