**CSCI 2312 - Design Document**

1. **Title**: Battleship Game

**Name**: Daniel Schlatter

**ID**: 108603919

**Date:** 11/1/19

**Status of testing on CSEgrid**: Program works excellently, no known scenarios that would cause the program to crash (Might have missed a few minor things but I did catch every error I could think of).

**Status of Program**: All functionalities fully implemented.

1. **Problem Description**The program simulates a game of battleship between a user and the computer by taking input from the keyboard for the user and generating random inputs for the computer, then producing outputs of grids with values based on the inputs given. The game is won when either all the computers or all the players ships are sunk.
2. **Input Requirements**:

The program takes an external input of a ships\_placement.csv file. Lines in the csv file are in the format of ShipName, LocationOnGrid, Orientation. All fields are strings. A valid ship name is one of: Battleship, Cruiser, Carrier, Destroyer, or Submarine. A valid LocationOnGrid is a letter between A and J followed by a number between 1 and 10. A valid orientation is either the letter V or H.

A valid file has 5 lines of input, none of the ShipNames are the same and none of the ships occupy any of the same positions in the grid.

Error checking is in place so that if any field on any line is invalid, that line is skipped over and the remaining ship that did not get placed is placed somewhere randomly on the grid. Error checking is in place to also only take 5 valid lines of input at most, if less than 5 valid lines of input exist, then as many ships as are needed will be placed randomly on the grid.

Error checking allows for even an empty file to successfully be entered into the program, in this case, all ships will be placed randomly.

For each of the players turn, external input from the keyboard is taken, as a string, either in the form of a valid LocationOnGrid (discussed in previous paragraph) or a Q if the player wishes to quit the game.

1. **Output Requirements**

Every turn the computer takes, the players grid, complete with the locations (in the form of Xs and Os) of the computer’s hits and misses, abbreviations of the ship’s names representing where each ship is, along with an S beside the abbreviation if the ship has been sunk, is printed to the console. A legend to define each symbol is printed every time a grid is also printed. The computers grid is printed every time the player takes a turn. The computers grid displays all the same things as the players grid, except if a ship has not been sunk yet than it will not be displayed. Upon game over, the computers grid with all ships revealed is displayed.

1. **Problem Solution Discussion**

**Problem: Dealing with incorrect input files**

I dealt with this problem by having a large amount of if statements enclosed in a while loop, and if any of the if statements were triggered, a continue statement was called to force the next iteration of the loop, effectively throwing away the whole line of input if any of the data fields had an invalid value. Then a for loop after the while loop would go through and ensure all ships had been placed on the grid, and if they hadn’t, then the ship would be placed using the placeShipsRandomly function. No special algorithm was used.

**Problem: Dealing with incorrect user input**

Again, I dealt with this problem by having several if statements enclosed inside a while loop, when the if statements are triggered, they produce an error message asking the user to enter a new input. In that case that the user entered an input already entered, or of a coordinate containing a sunken ship, I called the function recursively to repeat asking the user for input.

**Problem: Placing ships randomly**

All I did to place a ship randomly was generate a rand somewhere in between 0 and width\*height (100 in this case), and then checked if the gridsquare at that position in the grid was occupied or not. If it was, generate a new rand. If it wasn’t, I checked the length of the ship needing to be placed spaces to the left and all the spaces in between for if any of the spaces were occupied (or out of bounds) by another ship. If not, then the ship would be placed there, but if there was some ship(s) occupying those spaces then I would proceed to check to the right, then above, then below. If none of these directions were suitable for placing the ship, then a new rand would be generated, and the process would be repeated.

1. **Classes, Inheritance, and Data Structures**

**Class: Grid**

Grid is a child of the Gridsquares class and holds a 2d array of gridsquares, each gridsquare representing a coordinate on the grid. Grid also holds a vector of ships, I chose vector because the container was not super important, I just needed a container to hold information on each ship for access. I chose a 2d array because I would not need to be changing its size at all after I initialized it.

**Class: GridSquare**

GridSquare is the parent of Grid, and mostly just holds information for the grid functions. Has a couple strings, a bool and an int, no large data structures. Grid made sense as a child of GridSquares because a large part of grid is the container for the gridsquares.

**Class: Ship**

The ship class also mostly just holds information for the grid functions. These values are all constant after being initialized, except for the Boolean representing whether the ship has been sunk.

1. **Overall Software Architecture/Function Summaries**

Major functions in the program: placeShipRandomly, playerTurn, computerTurn, placeComputerShips, placePlayerShips, and printGrid.

**placeShipsRandomly:** this function will generate a random position on the grid, and if this position does not contain a ship, it will check every direction to see if all the spaces in that direction needed to fit the ship in are unoccupied and within bounds. This function is called by the placeComputerShips function to place all its ships, and it can be called by placePlayerShips function if the ships\_placement.csv file has an invalid line of input

**placeComputerShips:** This function just calls placeShipsRandomly for all 5 ships to populate the computers grid with ships. Not called by anything except main.

**placePlayerShips:** This function takes the input file and places ships according to each valid line of inputs LocationOnGrid data field. Performs many checks to ensure each line of input is valid and if one isn’t, the placeShipsRandomly function will be called.

**playerTurn:** This function takes keyboard input, performs a few checks to make sure it is not in invalid form, and not a position where a previously sunken ship is located. If it does not pass these checks, input is asked for again, and if it does pass these checks, then either the game is ended when the input is Q, or a hit or miss message is printed to the console depending on whether a ship was located at the inputted position and updates the computers grid accordingly. This function is called by the main.

**computerTurn:** This function generates a random position not previously fired at and not the location of a sunken ship and prints a hit message if a ship is located at the position, or a miss if the position is empty, and updates the players grid accordingly. This function is called by the main.

**printGrid:** This function prints a grid to the console, complete with hit or mis information on each coordinate that has been fired upon, and if asked to display the ships, it will display all the ships, represented by their two-letter abbreviation in each square they occupy. If not asked to display the ships, it will still display a ship if the ship has been sunk, indicating so by an S next to the abbreviation.

**Classes:** GridSquares, Ship, Grid

The Grid Class contains a vector of Ships and a 2d array of GridSquares. The grid class contains all the important functions previously listed, as well as some helper functions for some of those functions. The vector of ships is mostly used to hold information about each ship for access by functions in the grid class. The Gridsquares array holds gridsquares which contain information about whether they have been hit, whether a ship is located at that square and what ship it is as well as the string of the squares coordinate. Gridsquares mostly holds information for use by the grid class functions.

Both the Ship and GridSquare classes contain several strings, a bool and an int, information accessible by the grid. Their class functions consist of mutators and accessors for each member variable.