**Battleship on the Console Design Document**

By Gabriel Jones

ID: 107189922

2/28/18

NOTE: This program has implemented the extra credit of requiring all spots on a ship to be hit to be sunk, as well as a little additional logic for computer attacks.

The program currently compiles on the csegrid and has all features implemented.

NOTE: There is one current error that may rarely occur, which occurs when the computer attempts find a direction in a square that’s surrounded. It simply won’t be able to escape this situation currently.

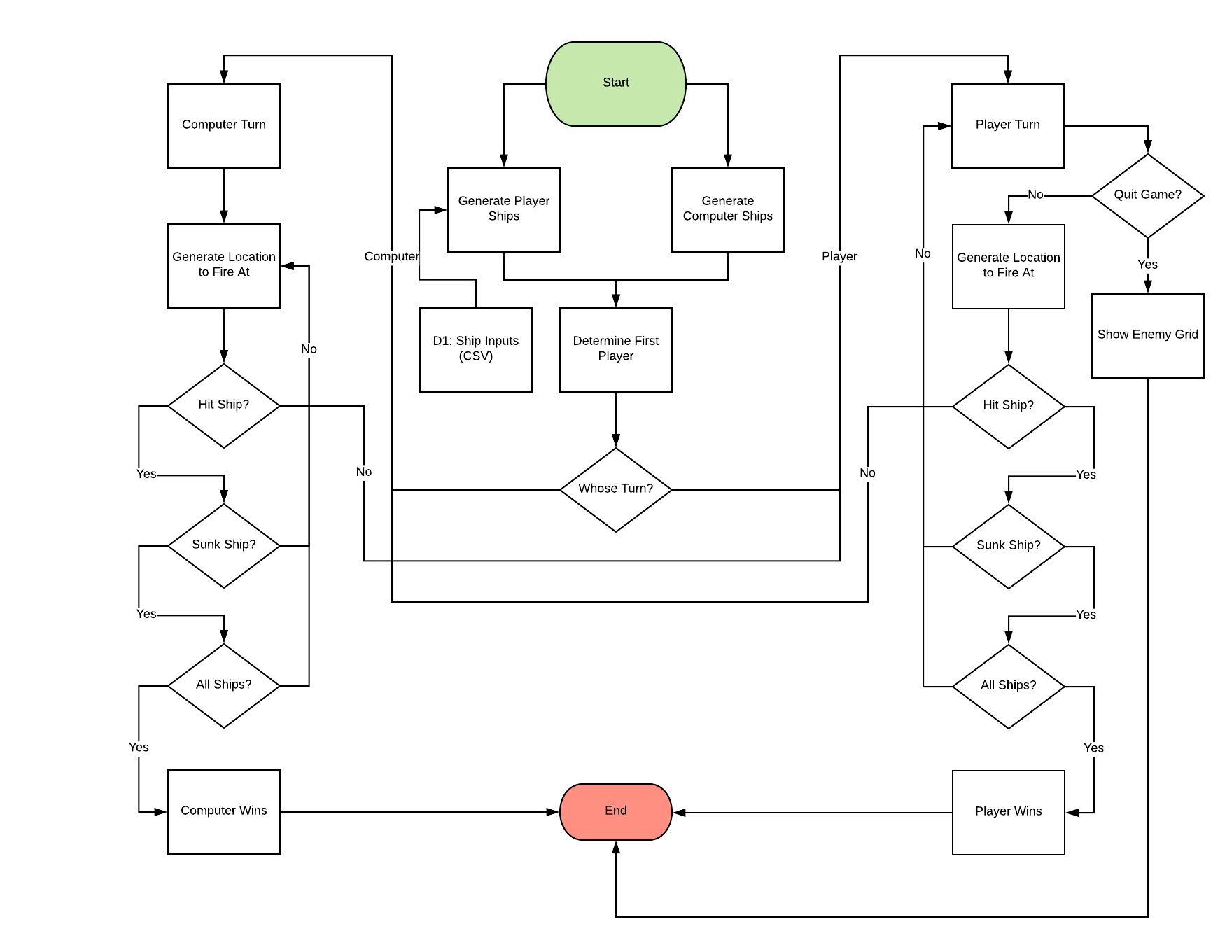
Brief Description:

Battleship is traditionally a board game played by 2 people. Players place 5 ships of lengths 5, 4, 3, 3, and 2 onto various positions of a 10x10 grid hidden from the other player. Ships can be placed horizontally or vertically but may not overlap or go off of the grid. Players then take turns trying to guess where the other player has put their ships by calling out a grid location, such as A5 or G7. If the player misses, the next player takes their turn. If the player gets a hit on a ship however, they are able to guess again until they either miss or get another hit. This continues until one player sinks all of the other player’s ships, and they are declared the winner.

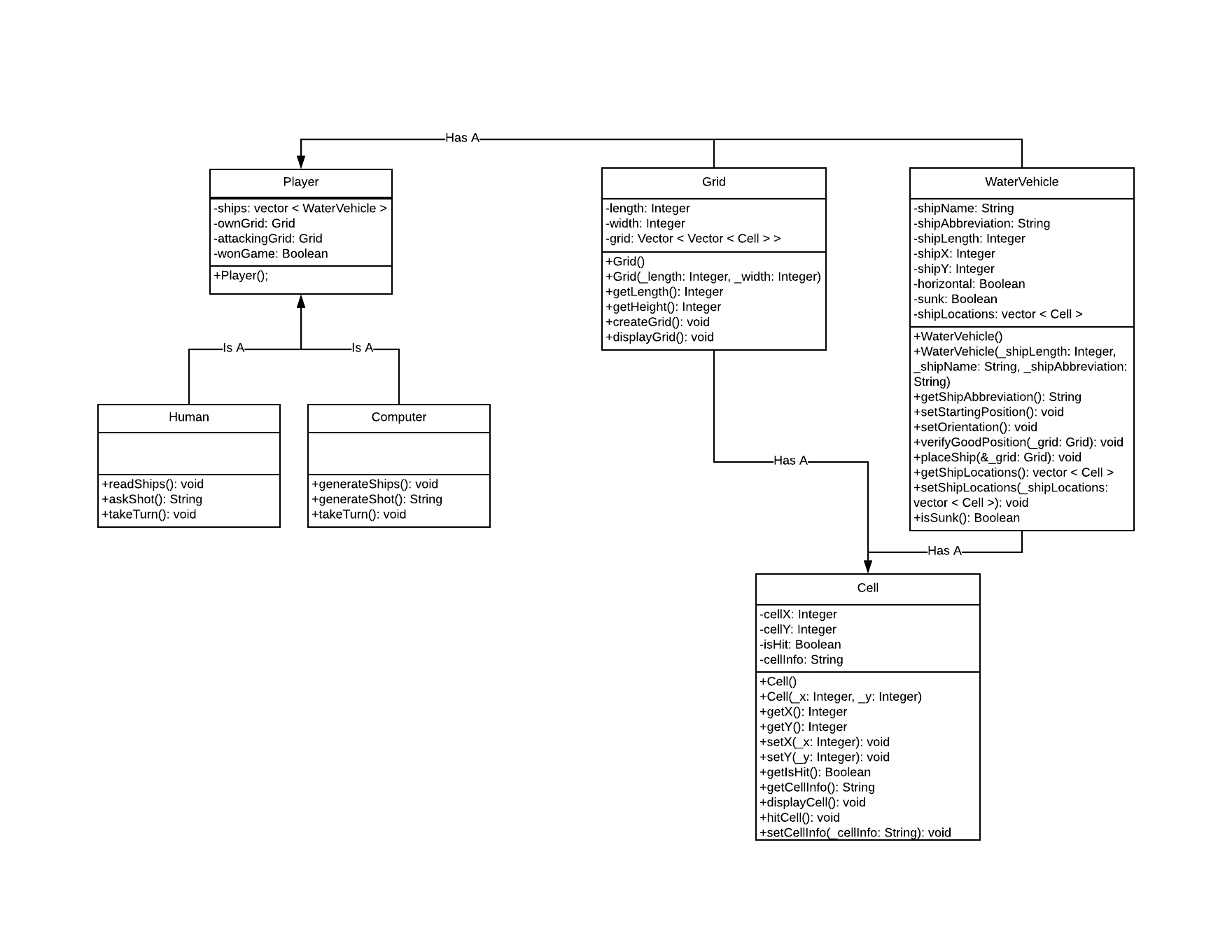
Overall Software Architecture:

The program starts off by generating all of the ships for both the player and the computer. Player’s ships are created by reading in from a csv file whereas the computer’s ships are created by randomly generating a location and orientation to place them. Two grids are created for each player, one to view their own ships, and one to view the attack grid. The ships are then placed on the ship grids using various functions to verify whether they are valid and then to actually edit the grid itself. After that, the actual game begins, and players alternate taking shots. If a player’s shot hits a ship, they get an additional turn to try and hit the ship again. If the player’s shot is a miss, the other player begins their turn. Play will continue until one player’s ships are all destroyed.

Activity Diagram:



Class Diagram:



Input Requirements:

* External
  + CSV File: This is used for reading in the human player’s ships provided in the format of TypeOfShip,Location,HorizontalOrVertical
  + TypeOfShip is a string which will be used to provide the name of the ship for display purposes.
  + Location is in the form of A-J,1-10, to describe the starting grid location for the tip of the ship.
  + HorizontalOrVertical describes the orientation of the ship from the starting location. Horizontal will fill the spaces to the right of location, and Vertical will fill the spaces downward. This will be of type string.
* Internal
  + At the start of each turn, the player will be given the option to either quit the game now, or take a shot at the other player’s ships.
  + If the player enters Q, which will be a string, the game will end.
  + If the player enters A-J,1-10, a coordinate point on the board, also described as a string, the game will check whether that input is a hit or not.
  + That input will first be checked if it’s valid. If it doesn’t match the format of LETTER,NUMBER, it will be rejected. This will also happen if the player chooses a number < 0 or > 10. Similarly, choosing a letter after J.
  + The input will then be checked to see if the player has already fired at that spot. If they have it will be rejected.
  + The input will also be rejected if it’s simply not a string at all.
  + If an input gets rejected, it will simply get asked again until it’s valid. There will always be a case where it’s valid.

Output Requirements:

* Internal
  + Before every turn, the game will display the grid which the player is firing shots onto, as well as the grid showing the player’s own ships and enemy shots.
  + Both grids are 10x10, with the x axis displaying the letters A-J, and the y axis displaying 1-10.
  + Each space on the grid will by default display an ‘O’, meaning the cell is open/empty. A hit on a ship will be described as an ‘X’, whereas a miss an ‘M’. On the player’s ship grid, if they ship isn’t already hit, it will show the ship abbreviation for that particular ship, such as ‘B’ or ‘Cr’.
  + Each turn will prompt the player to either make a shot or quit the game.
  + Once the game ends, if the player lost, they will be able to see the grid showing the locations of the remaining ships.

Problem Solution Discussion:

The program starts off by creating a human player and a computer player. The player has their ships read in from a csv file, which constantly will check if inputs are valid ship locations, using the placeShip function in WaterVehicle. PlaceShip will use 3 sub-functions to get the ship orientation, ship location, and the last to check if it overlaps another ship or goes over the edge of the grid. These work by constantly checking all the cells the ship should occupy and seeing if they both exist and are all empty. If the game is able to successfully edit all the cell locations on the grid and place all 5 ships, the game will start. The computer has a similar algorithm, just randomly generating locations and orientations until it can place all 5 ships.

The game will then start, and pick a player to start randomly. The game will ask for a shot input from the player, and check that against the grid to see if it’s a valid input. It will then return that it was either a hit or miss. If it’s a miss, the other player gets a turn, otherwise, the player gets to take another turn. Gameplay continues until one player destroys all of the other’s ships (or rather, each cell the ships are occupying is marked as being hit).

At this point, the player can choose to play the game again with the same set of ships and different computer ships if they so choose.

Classes, Inheritance, and Data Structures:

* Cell Class
  + The cell class is used to store information about each part of the grid.
  + The cell contains an x and y position, as well as what ship is occupying it, if any.
  + The cell will also contain if it’s gotten hit by one of the torpedo shots fired by the other player, and accurately display a hit or miss.
  + It has functions for displaying its contents, in which a hit or miss will take precedent over the ship it’s containing when displaying for the game.
* Grid Class
  + The grid class is used to display ship locations, hits and misses, and open spaces, by making use of many cells.
  + Upon creation, the grid makes a 10x10 matrix of cell objects, with the x coordinates being A-J, and the y coordinates 1-10.
  + The grid has functions for displaying the contents of all of the cells it contains, neatly in the 10x10 format.
* WaterVehicle Class
  + The WaterVehicle class is used to place a ship on the grid as well as check if it’s gotten sunk.
  + It takes in a name, an abbreviation to display on the grid, a starting x, a starting y, and an orientation.
  + The main driver of this class is the placeShip function, which is get the starting position, orientation, and check whether or not the input is valid by checking it against the grid object.
  + If it finds that the input is valid, it will edit the cells on the grid of the specified locations to have cell information of that ship abbreviation.
  + The ship object will also keep track of which cells it has edited the information of (parallel in a sense with the grid).
  + The ship also keeps track of whether or not it has been sunk, by checking if any of it’s cells have gotten hit after each turn, and displaying it if has.
* Player Class
  + The Player class is the generic class for the different type of player.
  + It holds two grids, one for displaying shots on the opponent’s grid, and one for the player’s own ships.
  + It also holds a list of the ships that the user has for checking whenever a ship gets hit on the grid.
  + This class is used for inheritance as the Human and Computer both share the same variables and objects, just with slightly different functionality.
* Human Class (inherited from Player)
  + The Human class inherits from player, as it has slightly different functions from the computer for creating its ships and making a turn.
  + The ships are read in from a csv file, as described above in input.
  + The Human for each turn will receive an input from the console to make a shot on the grid, making sure to continue after each successful hit. Whenever it makes a miss, the Computer gets its turn.
  + The Human also has the option to quit at any time if it so desires. This will display the current enemy’s ships grid and end the game.
* Computer Class (inherited from Player)
  + The Computer class inherits from player, as it has slightly different functions for the human for creating its ships and making a turn.
  + The ships are created through random generation of starting points and orientations, which are checked against the grid until the ship can be successfully placed.
  + The Computer’s turns are also randomized, making a random shot if it hasn’t gotten a hit in the turn before. If it misses, it trades off the turn to the Human. However, whenever it gets a hit, it will randomly pick one of the 4 adjacent squares to that hit that haven’t already been hit to try and hit the ship again. It will continue hitting in this direction until it encounters either the border or misses, from then it will pick another random direction.

The main Data Structure used in this program is a vector. This is mostly because it’s somewhat difficult to augment the size of an array without specifically using a dynamic array, and the vector handles all that logic for us. We use this for storing all the cells in the grid (as a vector in a vector), as well as the list of all the ships in the game.