# CSCI 2312 - Final Report

1. Title

   Program title: “Battleship: the Computer Game”, author: Xavier Swehla, [redacted] date: December 8th, 2020.

2. Problem Description

   This is a computer game that runs on the console window that simulates playing the single player version of the tabletop game known as “Battleship”. The player specifies their ships’ positions through a file, then proceeds to guess the locations of the computer’s ships. The first one to **land all the hits** on all the opposing team’s ships wins.

   **Note:** I was able to finish adding the extra credit portion for fully sinking the ships (page 6 of the handout) but I was unable to finish the additional logic for the computer, so the fully sinking ships is the only extra credit portion added.

3. Input Requirements

   For our program, we can separate the user/external inputs into two categories: file input, and keyboard input.

   File input

   - The only file input is from a single file called “ship_placement.csv”
     - File contains the name, position, and alignment of each ship
       - Name - will be stored as a string in the Ship object
       - Position - will be stored two integers in the Ship object
       - Alignment - will be stored as a char, either ‘H’ or ‘V’
     - 5 total ship objects

   Keyboard input

   - Main Menu input
     - A single character corresponding to the selection from the menu
     - The user enters a char which will be stored as char
     - Range: a(play game), b(view tutorial), or q(quit)
   - Tutorial menu input
     - A single character representing the selection from the tutorial menu
     - The user enters a character and it is stored as a char

        - Range: n(next page), q(quit)(back to main menu)
- Game input
    - During the execution of the game loop, the user enters a guess for the computer's ship locations, which is a character followed by an integer e.g A1 or a1
    - The data will be stored as a two integers
    - The character (first coordinate) will be translated to an integer using a function called CharToInt()
    - Range:
        - First coordinate: A/a to J/j (the program accepts both lower and uppercase)
        - Second coordinate: 1 to 10
        - Both coordinates will be internally stored as ints from 0-9
    - At any point of time, the user may decide to swap out their firegrid for the computer's shipgrid by typing in "sv_cheats1". This is mainly used for testing purposes, but it seemed like a good option to keep in for a single player computer game. This is one way to ensure that the fully sinking ships feature is working.

## 4. Output Requirements

Output using the console window (std::cout)

- Title of the game and welcome message
- Main Menu
    - Play the game
        - Displays player's FireGrid
        - Displays player's ShipGrid
        - Display prompt for next guess
        - If the shot was a hit or miss
            - Displays Ship hit by displaying a 0
            - Alerts player of miss by an X
        - Repeats until game over
        - Displays winning/losing message
            - Displays computer's ShipGrid if game lost
    - View Tutorial
        - Displays the pages of the rules/info
    - Quit
        - Exits the application

<u>No file output needed</u>

## 5. Problem Solution Discussion

The development of the game is split up into three major steps/algorithms: 1. Creating grids that can store the ships, 2. Adding the hit detection functionality, 3. Generating the computer's ship placement and guesses/shots.

1. For implementing the grids, I used a 2 dimensional vector which stored characters. However, the issue that I ran into was how would we display the ships on the grid and ensure that the ships would not overlap? To display the ships, I placed the letters of the ship's name on the grid according to the ship's size. We can get the occupied cells of each ship by adding its size to its starting position. Then we can check to make sure that each cell the ship will occupy is blank (not taken) and within the boundaries of the grid. All the unused cells/positions can be filled with a blank space or a character (I decided to use tilde ~ as the blank character since it looks like a wave).
2. For detecting hits on the ships, we iterate through the positions occupied by each ship to see if coordinates of the shot is occupied by any of the ships. If the cell is occupied, the shot was a hit and we can subtract one from the health of the ship. We can return the index of the ship that was hit to change the health as needed or return -1 to signify a miss. If we get a return value that is not -1, we can display the shot as a hit on the FireGrid or ShipGrid depending on who took the shot.
3. The implementation for generating the computer's shots and ship placements will make use of the rand() random integers to randomly select the placement of the ships and guesses. We use the same method for detecting overlapping ships as mentioned in 5.1 (two paragraphs up) and if the ships overlap or are out of bounds, we generate new random coordinates until the ship can be placed. For ensuring that the guesses are not repeated, the function will make a guess, check to make sure the coordinates are not marked as hits or misses, generate a new guess until the guess is valid, then call the function for sending an outgoing shot.

The rest of the implementation is fairly straightforward and considerably less complex.

## 6. Classes, Inheritance, and Data Structures

<u>Classes</u>

- <u>Grid</u>
  - Holds a private vector<vector<char>> (2dvector that holds chars) which can be displayed in grid fashion
  - Holds a private vector<Ship> which will contain the ships for the grid
  - Contains methods of constructing Grid objects, displaying Grid objects, and setting the two vectors

I decided that it would be easiest to build a base class called “grid” so that we can build the two types of grids, the grid containing the ships and the grid for firing torpedoes, that each player will have based off one common class to make setting up the grids relatively easy. The added benefit is that there will only be functions available relating to the type of grid that you are calling which results in simpler implementation in the code. Examples of this are given below.

- ShipGrid
    - Child class of Grid
    - Has three additional functions
        - FillShipGridWithShips(): prepares the grid for the game
        - FireTorpedoAtShipGrid(): updates the visual component of the grid.
        - ShipHit(Point p): checks if the point is occupied by a ship, if the point is occupied, it returns the index of the ship that is in the cell.

The ShipGrid class is a child of the Grid class which will contain the owner’s ship positions and the results of incoming torpedoes. The ShipGrid only manages the owner’s grid of ships from a visual aspect. The logic for hit detection is handled elsewhere.

- FireGrid
    - Child class of Grid
    - Has additional functions to display the firegrid and fire torpedoes at a firegrid
    - Also contains an additional grid to manage the visual component of the grid without accidentally showing the player the computer’s ships
- Ship
    - Holds ship data such as: size, starting position, positions that are occupied by the ship, alignment, health, and type.
    - Contains a vector<Point> to store the positions occupied by the ship.
    - Has various mutator functions to set the value of health, and to get the values for other private members.
        - The health variable is the only variable that is allowed to be changed in run-time.
- Point
    - Holds positional data in the form of two integers, x and y
    - This is mainly used by the Ship class to store the positions of the ships so that we can detect hits/hit registration (commonly referred to as hitreg or hit detection).
- Exceptions
    - Exceptions is a base class for defining various exception classes that have to deal with input file errors.
    - Each exception class has one public member variable, std::string error, which holds the text that will be displayed to alert the user when an exception is thrown.

**Inheritance**

**Grid:** Grid is the base class of ShipGrid and FireGrid. The grid class contains a 10x10 grid which holds a single character in each cell to represent ships, hits, or misses.

**ShipGrid:** ShipGrid is one of the child classes that is derived from the Grid class. The ShipGrid class contains additional functions which are necessary to update the grid to accurately show hits/misses as the game progresses. ShipGrid also contains a function to receive incoming torpedoes and update the values of the cells on the grid to reflect the changes.

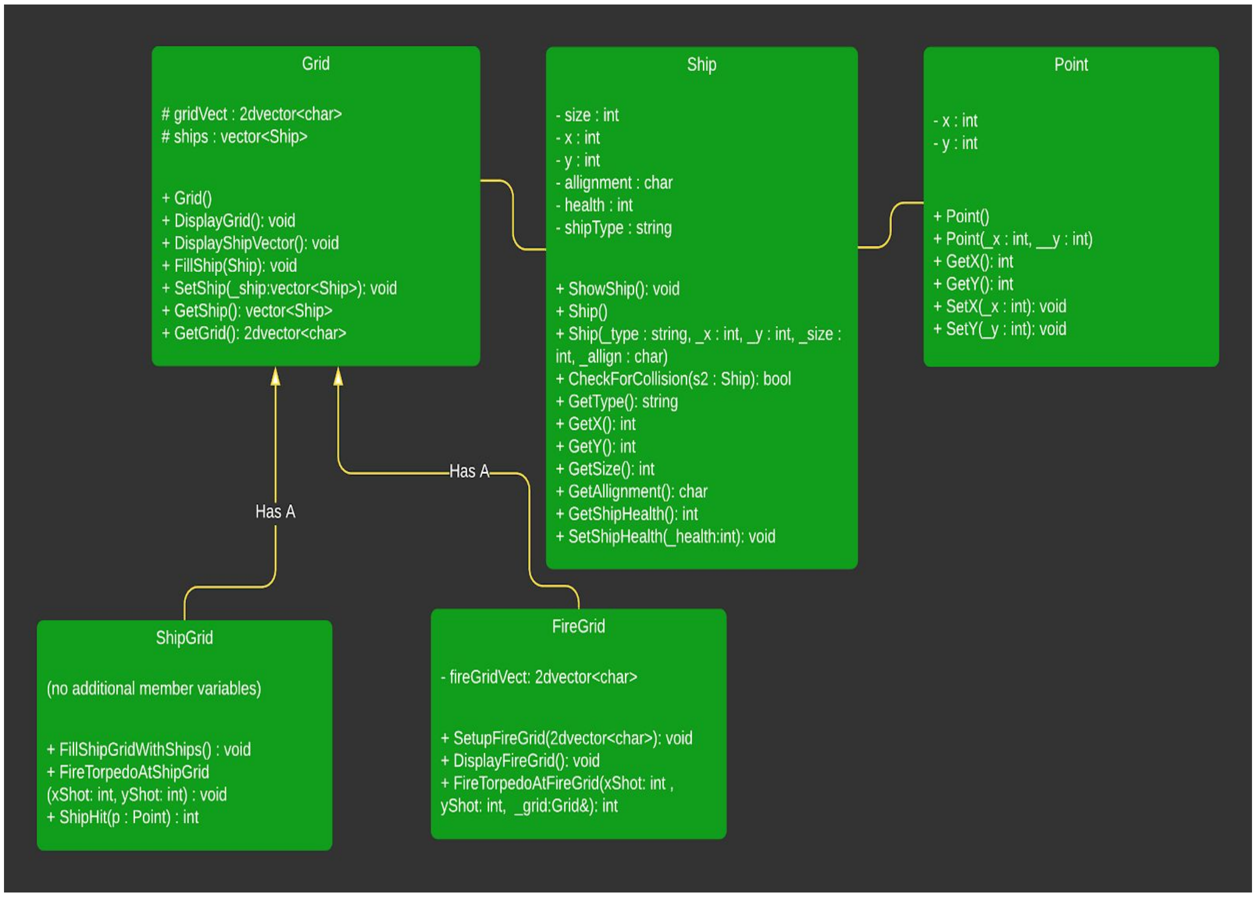
**FireGrid:** FireGrid is the other child class that is derived from the Grid class. The FireGrid class contains additional functions to accommodate the player sending outgoing torpedo shots and reflects whether or not it was a hit or miss.

I decided to split up the grids into the two types (shipgrids and firegrids) because I started working on developing the code for the project (with just one grid class) and quickly found that it would be beneficial to split up the grids into types based on their functionality and use case in the code. After splitting up the grids, I found that the code was less confusing to read and work with overall.

*Data Structures*

*Vector<vector<char>>*: The entire program deals mainly with vectors. For example, grids use 2 dimensional vectors that hold characters, Ship data is stored into Ship class objects and then stored in a vector. I decided to use vectors instead of arrays or other data structures because vectors are very easy to work with in my experience and are very easy to iterate through thanks to the built in .size() function. Another benefit of working with vectors is that they are dynamic in size, so it is easy to iterate through a loop and add items to the vector.

**UML Design Document**


Grid
# gridVect : 2dvector<char>
# ships : vector<Ship>
+ Grid()
+ DisplayGrid(): void
+ DisplayShipVector(): void
+ FillShip(Ship): void
+ SetShip(_ship:vector<Ship>): void
+ GetShip(): vector<Ship>
+ GetGrid(): 2dvector<char>
Ship
- size : int
- x : int
- y : int
- allignment : char
- health : int
- shipType : string
+ ShowShip(): void
+ Ship()
+ Ship(_type : string, _x : int, _y : int, _size : int, _allign : char)
+ CheckForCollision(s2 : Ship): bool
+ GetType(): string
+ GetX(): int
+ GetY(): int
+ GetSize(): int
+ GetAllignment(): char
+ GetShipHealth(): int
+ SetShipHealth(_health:int): void
Point
- x : int
- y : int
+ Point()
+ Point(_x : int, __y : int)
+ GetX(): int
+ GetY(): int
+ SetX(_x : int): void
+ SetY(_y : int): void
Has A
Has A
ShipGrid
(no additional member variables)
+ FillShipGridWithShips() : void
+ FireTorpedoAtShipGrid (xShot: int, yShot: int) : void
+ ShipHit(p : Point) : int
FireGrid
- fireGridVect: 2dvector<char>
+ SetupFireGrid(2dvector<char>): void
+ DisplayFireGrid(): void
+ FireTorpedoAtFireGrid(xShot: int , yShot: int, _grid:Grid&): int

The UML document is also included in the assignment submission in case of viewing issues

## 7. Overall Software Architecture

**Functions**

a. Main
   i. Main no longer reads in the ship data from the csv file, it has a while loop which simply calls the function to input the ship data from the csv file, then calls the DisplayMainMenu function to prompt the player to choose a selection. If the player chooses to play the game, main passes the grid objects to the DriverFunction in GameLogic.cpp

b. ReadInShipFromCSV
   i. This is the function that gets the ship data from the csv file and it takes the input file as an argument. Unfortunately this function is around 90 lines long, and that is due mostly to the exception handling that occurs while reading in the data. Most of the testing for invalid input has to occur by a certain time, or certain functions will call unexpected(). For example, if we try to use string.front() on an empty string, unexpected() is called and the program terminates.

c. DisplayMainMenu

   i. This function displays the options that the user can choose from. The important options are “play the game” and “quit”. If the player chooses to play the game, the function will return the char ‘a’ back to main() and the DriverFunction will be called. The user also has the option to view a tutorial which is just a quick explanation of the Battleship game.

d. DriverFunction
   i. This function is the “main” function for the game logic. It takes in the player’s grids as parameters and loops through the core game logic until the player wins or loses.
   ii. DriverFunction makes use of several smaller functions to perform tasks such as validate the user’s input, generate the computer’s ship placement, generate the computer’s guess and so on.

e. ShipDestroyed
   i. This function is short, but important. This function contains the logic which determines if a ship was destroyed. If the ship was destroyed, the function then checks if all of the ships are now sunk, and if so, it ends the core game loop (DriverFunction) and calls the function to alert the user that the game is over.

**Classes**

a. There are six main classes in this program, Exceptions, Grid, ShipGrid, FireGrid, Ship, and Point.
b. Exceptions has many child classes with a single member variable. All the classes under Exceptions are used for generating exceptions on retrieving invalid data from the input csv file.
c. Grid is the base class of ShipGrid and FireGrid and contains the members and functions that all grids can make use of. Every grid has a collection of Ships that are stored in a vector.
d. ShipGrid is the class that will contain the ship placement and keep track of the ships’ health. ShipGrid contains functions to place the ships on the grid, and receive an incoming torpedo.
e. FireGrid is the grid where we keep track of our own guesses. FireGrid has another grid which is derived from the ship placement of the opponents ships so that we can easily track which shots are hits and misses. FireGrid contains functions to send outgoing shots, set up the FireGrid, and display the grid.
f. Ship contains the information regarding a single ship such as health, the placement of the ship which is stored as a vector of Points, the ship’s health, the alignment, the size, and the ship’s type. Ship also contains functions to get all of the private variables and set the ship’s health.
g. Point holds two integers, an x coordinate and a y coordinate. Point represents a single cell on the grid.

## 8. Program Status

a. The program compiles and runs on Visual Studio and csegrid.
b. All of the required features have been implemented as well as a portion of the extra credit: fully sinking ships. To sink a ship, you must hit all of the spots that the ship occupies. To test this, please see section 3. Keyboard Input > Game Input > last bullet point.