CSCI 2312 Final Project - Battleship

Tobby Lie
Student ID - 103385744
10/18/17

Problem Description:

This program will utilize object oriented programming concepts such as inheritance, polymorphism and exception handling in order to read in a file that specifies types of ships, grid placement and orientation. The user will then play a game of Battleship with a computer that randomly positions five ships on a grid of it's own. The user and the computer will take turns generating torpedo shots on the other player's grid. Each turn the game will determine if a ship of either player has been hit or sunk.

Status reports of the cpu board will need to be displayed at every turn and also when the game ends either by victory or program exit. If it ends by victory or program exit, then the status needs to be displayed as well as the cpu board to reveal where their ships had been placed.

The basic gameplay flow of the program will be a loop. At each turn the player will be able to quit by inputting Q or to continue playing by entering any other key. The user will then be asked to input coordinates on the board to guess the cpu ship location. If their guess was a hit, it will display an O on a separate grid that they can see which displays all of their guesses: X means miss, O means hit. After their turn, a status report will print notifying if the player hit or sunk any enemy ships. The cpu will then randomize a torpedo shot which will appear on the players grid, a cpu report will then print displaying the status of their guesses and how they impacted the user grid.

Overall Software Architecture:

(At bottom of document)

Input Requirements:

Input file: ship_placement.csv

- This file will have information about five different ships to be included in the game.

This information will consist of the type of ship which will be read into a string for processing, the location of the ship which will be two coordinates for the grid read into

two individual int variables, and a horizontal or vertical orientation which will be represented by a V or H which will be read into a char type. This input will need to be checked for validity.

- Specifically the combination of ship length, orientation and start position will need to be checked so that the ship is placed within the bounds of the grid.
- Various exception handling tests will need to be done in order to handle exceptions such as negative input for starting position or a letter that is not V or H for HorizOrVert.

- TypeOfShip: string
- Location_x: char
- Location_y: int
- HorizOrVert: char

User input: keyboard

- The user will also provide input through the keyboard which will represent the x and y coordinates of a torpedo shot on the grid. The x and y will be stored in two separate variables, one int and one char to represent an input such as A10 since the grid will have x coordinates A - J and y coordinates 1 - 10. The user may also input Q to exit the game at any time. At each turn, the game

ask the user if they would like to continue placing torpedo shots or if they would like to exit.

- torpedo_x: char
- torpedo_y: int
- quitGame: char

Output Requirements:

This program does not require any file output, only an output to screen. The program will print out a message if either player has had a ship hit or sunk. The program will also print out a message if the player either quits or the game ends by one player winning. This message will output a grid with all of the player's guesses and the locations of the computer's ships on the grid. The game will be played on four grids but for the user only two grids will be visible. These grids being the grid the user is using with all of their ships placed and the torpedo shots placed on the computer's grid signaling if that coordinate is a hit or a miss, which will be represented as O's for miss and X's for hits. These two grids will be updated and printed to the screen for the player to see at every turn. The grid will be a 10 x 10 grid which will have x coordinates that range from A - J and y coordinates that range from 1 - 10. If user chooses to quit game at any point, the grid with the user's guesses and computer's ships will be printed out.

Output for beginning of a round:
- user will be prompted for torpedo coordinates
- will also have the option to input Q to quit game
- user's guesses displayed on grid, X meaning hit and O meaning a miss.
- user's own grid with ships displayed and computer's torpedo shots

Output for end of a round:
- updated grid with user's torpedo shot, signaling a hit or miss.
- a status report saying whether any hits were taken from the computer and if any ships had been sunk.

- a status report saying whether computer had taken any hits or if any computer ships had been sunk.

Output for if a winner occurs:
- message displayed announcing who the winner is

Output for when user quits:
- computer grid is outputted with ship placements revealed.

Problem Solution Discussion:

The overall flow of this program will essentially be based on a loop that continues to play the game until either the user or computer has won the game, meaning that either one has sunk all of the other side's ships. This loop can also terminate by inputting Q to exit at any turn.

Something important to note would be the error checking for input coming in from the input file. The input from the file must be checked to be sure that the field is first of all available, and then must be checked for valid input. Valid meaning input that makes sense for that field. For example coordinates for the grid must be positive integers and indication of orientation must be either V or H (lowercase will work as well).

The game will then place the ships on the grid based on the file information. This is essential as well, as some sort of bounds checking must be utilized to tell if a ship of a certain type in combination with it's length, starting point and orientation will correctly be placed on the grid, meaning no part of the ship is hanging off the grid, if there is error in this regard, the program can exit with an error statement.

For the torpedo shots, the user and computer will generate coordinates on the grid for which the torpedo shots will be placed. Two things must be checked for this. One being if the coordinates for the shot have already been used, if the position already has a torpedo shot, then the game must either ask the user for new coordinates or in the computer's case the computer must generate two new random coordinates until a spot that has not received a torpedo shot is chosen.

At every turn, the game will need to determine if a ship was hit or sunk. A hit means that any point in a ships length has been hit by a torpedo shot, while a sink means that every point in the ship length has been hit. A hit can be determined by creating a hit counter for each ship, there will then be a function that increments the hit counter for a specific ship if it was hit. If the hit counter matches the length of the ship then we know a ship has sunk.

At each turn, each ship will be checked in order to gain some sort of status of it's condition. This will be done by tracking the condition of the ship at each turn so that every time a point at a ship's length is at the same point that a torpedo shot is generated, it's hit counter will be incremented. In order to make sure that the hit counter does not increment at every round we must also create a counter to count the torpedo shots that coincide with ships length. If this

counter is one less than the ship's hit counter, we know that we need to increment the ship's hit counter because this signifies that a new torpedo shot has hit the ship. A hit counter will tell us how many points in the ship have been hit, if any. This will be checked at every round as well as if the hit counter matches the ship's length. If it does, then we know the ship has been hit at all of it's points, thus the ship has been sunk.

Another important aspect to consider is understanding what will be output for the user. The user will be able to see it's grid with ships placed and also the enemy grid without ships, that displays torpedo shots that were hit (X) or missed (O). As stated before, at every turn the condition of each ship is checked as to whether it has been hit or not. If a hit on computer grid occurs, then this will be available and printed for user view.

When the user x coordinate input is taken in, since the input will be char values, we must somehow convert this char input as numeric input as the multidimensional array will be more easily navigated within the implementation with numeric values. For this reason we will use a series of switch statements that char inputs can be sent through to determine the numeric value of that portion of position.

For example:

```
char input_x;
int numInput_x;

switch(input_x){
        case 'A':
        numInput_x = 1;
        break;
…
}
```

Data Structures:

Since the size of the grid is fixed to 10 x 10, an array was chosen for the implementation of this grid. A vector could very well have been used, however for this program the grid implementation is rather simple and fixed which allows us to just utilize an array of fixed multidimensional size.

User Interface Scheme:

User interface will be relatively simple. At each turn of the game, the user can either choose to place a torpedo shot on the computer grid or exit the game by pressing Q. Alternatively, the game ends if either the computer or the player has won the game by sinking all of the other players' ships. This will then prompt if the user would like to play again, in which case the user can choose to play again or quit.

Overall Software Architecture: