Battleship Game Simulator

Program Title: Battleship Game

Author Name: Ian Anderson

ID#: 109166476

Date: December 3rd, 2019

Overview

This program is a simplified version of the game battleship where the user plays the game against the computer. Two 10x10 grids will be filled up with five different ships of varying size, and the two “players” will play the game in rounds, attempting to select a grid square (or fire a torpedo) that is occupied by the enemy’s ships. After each round, the player will be updated on whether they successfully hit a ship or missed, and an updated grid will be displayed to the user showing all of the grid squares or coordinates they have already chosen that game. Once all of a player’s battleships have been “sunk”, the game ends, and the other player is the victor.

Input Requirements

There will be an input file used to determine the location of the user’s ships for the game. The input file must be named exactly “ship\_placement.csv” and must be a comma separate value file. The file cannot contain blank data sections and must contain allocations for exactly five ships (Carrier, Battleship, Cruiser, Submarine, Destroyer). Each data set must contain the ship name, the upper left coordinate of the ship (char: a-j)(int: 1-10), and H or V to denote whether the ship will be placed horizontally or vertically, and must be written in that order. The user may receive an error message if any of the data inputs are incorrect, in the wrong order, are missing, or if any of the ships intersect on the coordinate plane.

The user will also be prompted to give input whenever it is their turn to “shoot a torpedo” and guess a grid square. These coordinates must use a char in the range of (a-j) or (A-J) to denote the x coordinate and must use an integer in the range of (1-10). The user will receive an error message and be prompted again if their input falls outside of this range or is the wrong syntax.

Output Requirements

At the beginning of execution, the user will be prompted with a welcome message showing the filename being accessed.

Error Message: “There was a problem getting the Ships from the input file. Please make sure the filename is correct and the values are in the proper format! Check the ReadMe for more information.”

If the function detects that two of the ships intersect while it is populating, the program will exit, and the user will get an error message.

Error Message: “Two of the ships appear to intersect. Try changing your coordinates and reloading the program.”

Error message: “One or more of your ships doesn’t fit on the grid with the given coordinates. Please rearrange your ships and try again.

While the game is running, two grids will be printed to the screen after each turn. One grid will show the player all the guesses that they have made on the opponent’s grid so far and show successful hits with ‘O’s and misses with ‘X’s. The second grid will show all of the guesses made by the computer opponent demarking hits and misses with ‘O’s and ‘X’s respectively as well.

As well as displaying the grids, the user will also receive a message after each of their turns denoting weather their last hit was a hit or miss.

Hit message: “Your shot was a hit!! (ship type) was sunk, good job!.”

Miss message: ”Your shot was a miss!”

Victory message: “Congratulations! You are the victor of this match.

All the computers ships have been sunk, good job!.”

Defeat message: “The Computer sunk all of your ships! Better luck next time.”

The user will also receive messages based on what the computer did that turn.

Hit message: “The computer hit your (ship type) and sunk it!;

Miss message: “The computer shot and missed!”

Problem Solution

When the program begins execution the first thing we do in the main function is create two GuessGrids and two ShipGrids, each contained in their appropriate child Classes (PlayerGrid and ComputerGrid). Each player will be assigned one GuessGrid and one ShipGrid. The ShipGrid that belongs to the user will be populated based upon the data in the input file. The ShipGrid that belongs to the computer player will be randomly filled using the compGenerate() member function in the ShipGrid class.

After this the main game begins, and the user will be prompted to input a coordinate for their guess. After the user input is validated, the program will check the guess with the opposing player’s ShipGrid (referred to as gridVect in the code) of the computer player and update the user’s GuessGrid with a hit or miss accordingly, after which the user will receive the appropriate message and then the two updated grids will be shown on the screen. The computer will then take their turn using the guess() member function inherited from the Grid class, and will repeat the process described for the human player, except the coordinates for the torpedo will be generated randomly. After each guess made by a player the game checks to see if the victory conditions have been met and if they have, the program will end the game and display the appropriate ending message.

Algorithm Analysis

Point

Basic class used to store coordinate points for use during gameplay.

Grid

*Void fill() :* An abstract class used to populate the grids initially. Is redefined in PlayerGrid and GuessGrid.

*Bool addShiptoGrid( int shipSize, Point topPoint, char direction, bool player ):*  This function will place the ship onto the grid given the passed in parameters. While doing so, the function will check to see if any ships overlap, or if any ship placements overflow off the side of the grid. If either of those problems occur, an appropriate error message is displayed (as long as player Boolean = true ) and the function will end prematurely so that the errors can be remedied.

*Void PrintGrid ( bool isGuessGrid ):*  Calls the printShips() or printGuess() functions depending on the boolean value passed in.

*Void printShips():* Prints the grid that is populated with the ships to the console.

*Void printGuess():* Prints the grid that is populated with the guesses so far to the console.

*Int hitDetection( Grid& opponentGrid, Point guessPoint ):* uses the passed in opponentGrid to determine if the torpedo fired was a successful hit or a miss, then updates both the ShipGrid and GuessGrid as needed.

*Bool VictoryDetection( bool player )*: Checks to see if either player has won by checking to see if there are any “unsunken” ships left on either shipGrid. If victory is detected, it prints the appropriate victory message to the console and ends the program.

PlayerGrid

*bool fileValidityCheck( string filename ):*  Opens the file with the passed in filename. Checks to make sure that the syntax of the file is correct. This includes checking to see if all 5 ships are included in the file and also checks to see if the proposed coordinates are the correct syntax as well. If anything is incorrect in the file, the function prints the appropriate error message to the screen.

*bool fill () :* Opens file “ship\_placement.csv” (filename can be changed in the function definition). Converts all the data from the files into the proper formats and passes them into the addShiptoGrid() function. As long as no problems occur while adding the ships to the grid, the function will end with a success message. If not the function will end prematurely and print the appropriate error message so that the user can fix any issues.

*Void guess( Grid& opponentGrid ):* Prompts the user to enter in x-coordinate(char) and a y-coordinate(int) and deals with any incorrect inputs with a re-prompt and an error message. After an acceptable Point has been generated from the user, it passes in Point in hitDetection() which updates the grids as needed.

*Bool VictoryDetection():*  checks for victory by seeing if any ships are left on the opponents grid and the prints the victory message and ends the program if victory is detected.

ComputerGrid

*Void fill() :* This will populate the grid with the five boats with starting coordinates chosen at random as well as the horizontal or vertical character being chosen at random. If the boats happen to intersect or if the ship will not fit onto the grid, the function regenerates new random values and does not place the original ship. Will continue this process until all 5 ships have been placed. Whenever a ship is successfully places, it’s placement info is displayed to the screen.

*Bool VictoryDetection():*  checks for victory by seeing if any ships are left on the opponents grid and the prints the victory message and ends the program if victory is detected.

*Void guess( Grid& opponentGrid ):* Randomly generates a point that has not been previously guessed. After an acceptable Point has been generated from the user, it passes in Point in hitDetection() which updates the grids as needed.

Software Architecture

We start by initializing one PlayerGrid and one ComputerGrid, each of which will contain a ShipGrid and a GuessGrid. The Player’s grid is then populated using the data given in the input file “ship\_placement.csv”. The computers grid is populated using random values generated from the fill() function.

After this, the game begins. Each turn, the user’s input gets stored into a char variable Xcoor and an int variable Ycoor. These values are passed into guess() and then the printGrid and the VictoryDetection() functions are called. For each of the computer’s turns we call guess() and then call VictoryDetection() as well. Turns are taken in succession until someone has won the game.

Functionality

The game is fully functional. All required functionality described in the assignment description has been implemented.

The program compiles and runs successfully on Clion using MinGW version 5.2. Unfortunately, I Did not get to test the program on the CSE grid.

A close up of text on a white background

Description automatically generated