CSCI 2312: Object Oriented Programming

Project

Part 1 Due: March 25, 2019 at 11:59 PM

Full Project Due: April 08, 2019 at 11:59 PM

For many of us old school gamers, we started out playing good old fashioned board games. Many of us learned strategy from games like Risk and Battleship. For your project, you will be developing a simple battleship type game to test your knowledge of the concepts we have learned in Object Oriented Programming.

The purpose of this project is to have you design of a fairly complicated project using concepts we have learned and then implement the solution using some of the code that we wrote in the previous projects along with new code, and then test your game.

You need to first start with the design of the project. If your design is carefully thought through, the coding of the game should be relatively straightforward since many of the classes and concepts closely match previous assignments. You will need to first read these requirements and make a design document (ensuring that all the requirements are met in the design). A sample design document is posted in Canvas and should be used as your design TEMPLATE (example). Create a design document complete with the class diagram and activity (flow) diagram, as well as any decisions you made on the best use of classes, inheritance, polymorphism, and exception handling.

After you have completed your design, then you will be ready to implement the game and test. I cannot stress enough that a good design document and understanding of the requirements will make the actual coding of the game much faster and more simple.

Please don't leave things until the last two weeks. Get started now, and please ask your instructor for help BEFORE you get too lost. Get the big picture done first. Worry about the structure and implementation of the major functionality. Then if you have time, work on the little details, and minor error checking.

So now onto the requirements. You sunk my battleship!….

For your CSCI 2312 Project, you will develop a simple battleship game. Battleship is a guessing game for two players. It is played on four grids. Two grids (one for each player) are used to mark each players' fleets of ships (including battleships). The locations of the fleet (these first two grids) are concealed from the other player so that they do not know the locations of the opponent's ships. Players alternate turns by 'firing torpedoes' at the other player's ships. The objective of the game is to destroy the opposing player's entire fleet. In our game, 'firing a torpedo' will be allowing the player to take a guess at where on the grid their opponent may have placed a ship.

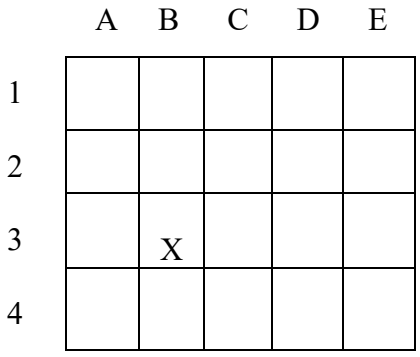
In the requirements, we will set forth other simplifying rules to limit the scope of this project.

Requirements

Given the requirements as a rough specification, you are to design the classes and implement the game. In our imaginary game company, the requirements below were developed by the Product Development Team and your instructor is the Product Owner. You are in full control of the choice of classes (please use classes appropriately or points will be deducted), data structures, algorithms, internal file format, detailed user interface scheme, or any other pertinent design decisions you need to make. As the Product owner, I care that it compiles and runs like it is supposed to, meets all the functionality and requirements I have set forth, and is easy to play and understand.

The Battleship game you are designing and implementing is a simplified version of the electronic Battleship game played in one player mode.

The game is played on four grids, two for each player. The grids are typically square and in our case will be 10 by 10. The individual squares in the grid are identified by the x coordinate (indicated by a letter) followed by the y coordinate (indicated by a number). The following is an example of a 5 by 4 grid with an X in the position B3.



Each player uses two grids. Each player uses one of their grids to arrange their ships and record the torpedoes fired by the opponent. On the other grid, the player records their own shots and whether they hit or missed.

Before play begins, each player secretly arranges their ships on their primary grid. Each ship occupies a certain number of consecutive squares on the grid (sizes of ships are in the following table), arranged either horizontally or vertically. The number of squares for each ship is determined by the type of the ship. The ships cannot overlap so only one ship can occupy any given square in the grid. The types and numbers of ships allowed are the same for each player.

Ship Type	Number of Grid

	Squares
Carrier	5
Battleship	4
Cruiser	3
Submarine	3
Destroyer	2

The game is played in rounds. In each round, each player takes a turn to fire a torpedo at a target square in the opponent's grid. The opponent then indicates whether the shot was a hit (a ship occupied the square) or a miss (there was not ship in the square). If the shot is a “miss", the player marks their primary grid with a white peg (X in our game); if a "hit" they mark this on their own primary grid with a red peg (O in our game). The attacking player then indicates the hit or miss on their own "tracking" grid with the appropriate color peg (red (0) for "hit", white (X) for "miss") so that they can understand where the opponent’s ship might be.

In the board game, once all of the coordinates of a ship have been hit, the ship is sunk, and the ship's owner announces “You sunk my battleship! (Or whatever the particular ship that was destroyed). For our purposes, we will consider a battleship sunk if the opponent has a single hit. When all of one player’s ships are sunk, the other player wins the game.

For your game, you will create a one-person version of the game where ‘the computer’ will play for the second player.

Part 1: Ship assignment in the Grid (Due Date: March 25, 2019)

In part 1, you are required to submit the design document (I will provide the document template). Also, as a part of part 1, you will read a file called ship_placement.csv which contains the type of ship, the first grid square for the ship placement, and whether the ship is placed vertically or horizontally (V or H in the field). The file will be in csv format (comma separated values). This is a common format and is comma separated (instead of being on separate lines). There will be commas between the values. Blank values will just have a comma noting to go to the next field (the game input should not have blank fields so you should handle the case where a field is blank). If you want to view the file, often this will be opened by a spreadsheet unless you specifically open it with a text editor. Do not open it with Microsoft Word, as this may change the format. The first line of a CSV file notes the data descriptions as follows:
TypeOfShip,Location,HorizOrVert

I have provided several sample files which contain good scenarios and scenarios with placement issues that you will need to handle using exception handling. Your game should run with any of these files, but should also be able to run with any valid file in the correct format. You will need to check whether all ships were included in the input file (and appropriate action to take if not), whether all ships have been placed, whether they fit on the board in the configuration given, and whether more than one ship occupies a

space (which is not allowed) when you read the input file from the user and how to recover if an error occurs.

You will then need to randomly position the computer's ships on the grid taking into consideration the same factors as you did for the player's input.

Part 2: Playing the Game: (Due Date: April 8, 2019)

You will need to prompt for and allow for the user to input their next guess in the form of a letter (A through J) and a number (1 – 10) indicating where they are targeting for their torpedo and you should error check the input. In our simplified game, you will determine if the torpedo shot was a hit or a miss. If the shot was a hit, consider the ship to be sunk. You should display a hit or miss, whether the ship was sunk and which one, and display their tracking grid so they know what they have guessed and where they have made hits. The entire ship which was hit will display as sunk.

After the user takes their turn, you must have the computer randomly select a shot that they have not previously taken. Then you must display to the user what the computer guessed, whether it hit any of the player's ships, whether a ship was sunk, and then display the player's placement grid showing where ships are located and what has been hit.

You should continue this until someone wins or quits the game – meaning you should allow the player to gracefully quit at any turn.

At the end of the game, you should indicate the game is over and who the winner was. You should also allow the user to quit the game by entering a Q when prompted for their next guess. If a player decides to quit the game, the grid with all of their guesses and the locations of the computer's ships should be displayed.

Overall System Design

1. You must have two different classes in your design.
2. You must use inheritance in one of the classes.
3. When reading from a data file, your program should test the input file to ensure that data is of valid format (basic error detection) using Exception Handling.
4. You should consider using the Grids from Assignment 2 to make this easier. You do not need to have 4 grids for this but if you decide to use only two grids, you need to make sure you do not show the player the computer's ship location when you display the grid after each turn.
5. Each component of the overall program should be modular.
6. Program should be fairly fault tolerant of user input and the appropriate user prompts and on-screen directions should be displayed
7. Split the program into multiple files based on the roughly categorized functionality or classes.

Submission Guideline

You need to submit following items (all zipped together):

1. Source code with reasonable comments
2. Makefile that works (and is tested) on the csegrid.
3. A single final report that includes (use template on Canvas):
    - Summary of provided functions. This should be matched with the requirements
    - Design that shows the overall program structures, and the explanation of key algorithms. A description of user interface scheme is required to explain the menu items at top level and items in sub menus and how to navigate through menus. A detailed instruction and sample skeleton is available from the Design Document in Files on Canvas
    - Accurate status of the program, what's done, and what's not completely implemented.
    - Accurate status of testing on the csegrid.
    - The final report should be in MS Word, or PDF format.

Grading Criteria

A. Submitting a working program that provides all of the required features will result in a maximum grade of 80%.
B. Documentation explained above will result in 20% of the grade. Note if you spend time getting it right for the Design Document Homework, you may not have anything to do for the project.
C. Any or all of the following will result in point deductions of up to 5% for *each* infraction.
    1. Poor and/or inconsistent programming style. This includes the following:
        a. Improper use of indentation.
        b. Overuse of global variables.
        c. Failure to keep functions modular and reusable (possibly applicable to other programs).
        d. Insufficient comments.
        e. Failure to use classes and inheritance.
        f. Failure to use exception handling.
    2. Insufficient prompts and display of results to the user
    3. Program is not *reasonably (not absolutely)* fault tolerant.
        a. Test to ensure that your program cannot be crashed or sent into an infinite loop by a user who is not following directions.
        b. Include a reasonable input file integrity check. Rejecting any non-conforming file is fine but may not be optimal for game play.
D. Partial credit may be awarded.
    1. You may get partial credit for non-working modules (functions) by explaining (in the separate document) where you think the problem lies.

2. Up to 10% could be lost for each required feature that is not provided.

E. Submitting a program that does not compile on csegrid.ucdenver.pvt may result in a deduction of at least 20%. Additional points will be lost for each required feature that is not adequately addressed.

Extra Challenge / Extra Credit

Some of you may want an extra challenge to boost your abilities and have some interesting resume material. If you have completed all of the requirements above and want an additional challenge, you can incorporate fully sinking the ships and additional logic to be used by the computer when it makes a hit. After determining whether the shot was a hit or miss, you will then need to determine if the ship was sunk. In the simplified version of the game above, one torpedo hit sunk the ship. For the extra credit, you will need to accurately track all hits on a ship to determine if it was sunk. If it was sunk, you will need to tell the player, "you sunk my XXXship" where XXX indicates which type of ship and then displays the information as described in the simplified version above. Additionally, you will need to add logic to the random selection of torpedo shots for the computer. When the computer takes a random shot that hits a ship, the next series of shots will need to be logically selected until the hit ship is sunk.

If you do the extra credit, please note this in your documentation and make it clear in the running of the program, so we can give you up to 15% extra credit. (meaning you could get 115% on the project). Note: Since this is extra credit, it needs to meet a higher standard for full extra credit).