CSCI 1933 Project 2

Battleboats Game: Due March 20

Instructions

Please read and understand these expectations thoroughly. Failure to follow these instructions could negatively impact your grade. Rules detailed in the course syllabus also apply but will not necessarily be repeated here.

- **Identification:** Place you and your partner's x500 in a comment in all files you submit. For example, `//Written by shino012 and hoang159`.
- **Submission:** Submit a `zip` or `tar` archive on Canvas containing all your `java` files. You are allowed to change or modify your submission, so submit early and often, and *verify that all your files are in the submission.*

  Failure to submit the correct files will result in a score of zero for all missing parts. Late submissions and submissions in an abnormal format (such as `.rar` or `.java`) will be penalized. Only submissions made via Canvas are acceptable.
- **Partners:** You may work alone or with *one* partner. **Failure to tell us who is your partner is indistinguishable from cheating and you will both receive a zero**. Ensure all code shared with your partner is private.
- **Code:** We will not be using unit tests to grade this project. You are free to make any design decisions you want, but your code must be reasonably clean, well-designed, and commented thoroughly. Your code may receive a penalty if it is confusing or demonstrates poor knowledge of Java. Code that doesn't compile will receive a significant penalty. Code should be compatible with Java 8, which is installed on the CSE Labs computers.
- **Extra Work:** If you enjoy this project and want to add extra features, please document the extra features thoroughly and allow them to be disabled for grading purposes. Mystery features we don't understand could cause grading issues. Extra work will not result in extra credit, but the TAs will be impressed!
- **Questions:** Questions related to the project can be discussed on Canvas in abstract. This relates to programming in Java, understanding the writeup, and topics covered in lecture and labs. **Do not post any code or solutions on the forum**. Do not e-mail the TAs your questions when they can be asked on Canvas.
- **Grading:** Grading will be done by the TAs, so please address grading problems to them *privately.*

Introduction

Battleboats is a probability-based board game that challenges the user to locate enemy boats hidden on a rectangular grid. The purpose of the game is to locate and destroy every enemy boat in the least number of guesses.

You will be modelling this game in Java. There are no requirements for the actual classes and functions you design, but you **must implement every part of the description below**. Your code may also be judged based on style. This should not be a stressful requirement - it simply means that you must create logically organized, well-named and well-commented code so the TAs can grade it.

You are required to specify which main method we should run in a comment in the file. This is because there may be multiple main methods. A good way to make this obvious is to make a `Main.java` class that contains only a main method.

IMPORTANT: You cannot import anything to help you complete this project. The only exception is importing `Scanner` to handle the I/O. Note that you do not have to explicitly import `Math` because Java already does it for you. In other words, you can use `Math` methods without importing `Math`

Note: You are not required to write tests for your code; however, you will find it useful to write your own tests to prove to yourself that your code works. **Please include any test classes you write with your submission**.

1 Board

The computer will simulate a square $n \times n$ board. **You are required to use a 2-dimensional array to represent the board**. The type of this array is up to the programmer.

The user will select whether they want to play the game on **Standard** or **Expert** mode when the program starts. If **Standard** mode is selected, the program should create a board of **8** rows by **8** columns. If **Expert** mode is selected, the program should create a board of **12** rows by **12** columns. Assume that the points in the normal mode board range from $(0, 0)$ to $(7, 7)$ inclusive, and from $(0, 0)$ to $(11, 11)$ inclusive in the hard mode.

Hint: You may find it useful to make your own `BattleboatsBoard` class that contains a constructor to set up the array and all the methods for setting up boats.

2 Boats

Each boat is represented by a line of consecutive squares on the board. Boats may not overlap other boats, extend outside the game board, or be placed diagonally. They may be horizontal or vertical. A boat is considered "sunk" when all the squares of the boat have been "hit" by the user.

Examples: Valid coordinates for a boat of size 3 are $\{(0,0),(0,1),(0,2)\}$ and $\{(1,1),(2,1),(3,1)\}$. Examples of invalid coordinates are $\{(0,0),(0,1)\}$, which is invalid because there are not enough points. $\{(0,0),(0,2),(0,3)\}$ is invalid because the points are not consecutive. $\{(-1,0),(0,0),(1,0)\}$ is invalid because the first coordinate is out of bounds. Finally, two boats cannot contain the same point because they cannot overlap.

After the game mode has been chosen and the game board has been correctly sized, **the program should place boats randomly on the board**. This requires randomly generating a coordinate (x, y) where the boat will be placed as well as randomly choosing whether the boat should be horizontal or vertical. The quantity of boats is defined by the width and height of the game board.

For the **Standard** game mode you will place **5** boats on the board. These boats will have the following lengths:

- 1 boat of length 5
- 1 boat of length 4
- 2 boats of length 3
- 1 boat of length 2

For the **Expert** game mode you will place **10** boats on the board. These boats will have the following lengths:

- 2 boats of length 5
- 2 boats of length 4
- 4 boats of length 3
- 2 boat of length 2

The user should be told how many boats are on the board when the game begins.

Hint: To randomly place a boat, consider the coordinate in the upper left. If the upper left corner was $(0,0)$, consider how the boat looks if it is horizontal or vertical. What upper left corner coordinates are invalid? The placing of the boats may be the most challenging aspect of this project: see what assumptions you can make to simplify it.

Hint: To generate random numbers, the `Math.random()` method can be used. However, this method returns a `double` in the range 0 to 1. We will need to scale this and then round it to a whole. To do this, use the `Math.floor(x)` function, which takes a `double x` and rounds it down to the nearest integer. For example, `Math.floor(2.9)` is 2.

3 Turns

Each turn, the user will input a location x, y to attack on the board. You can assume that x and y are integers, but you will need to check if the pair x, y is a valid location on the game board later. For this game you can assume that x represents which row to fire on and y is which column to fire on, with the 0th coordinate representing the first column or row. For example, firing on coordinate (1,0) would fire on the spot in the second row and first column.

- If there is no boat at the location, print "miss".
- If the user has already attacked that location or location is out of bounds, print "penalty".
- If there is a boat, print "hit". If the boat sinks, print "sunk". Do not print "hit" again if the user has already "hit" this location.

If the user attacks the same spot twice or attacks a spot out of bounds, the user's next turn will be skipped (meaning that an extra turn will be added to the turn counter. See the example below for clarification). The game ends when all the boats have been sunk. The game should report how many turns it took to sink all the boats, as well as the total number of cannon shots. Lower scores are better!

Example:
Suppose the user is playing on board of size 5x5 with one boat on it. While in the actual game there will be at least 5 boats on the board and a board size of at least 8x8, for this example let's assume that just one 3-length boat is placed on the board with the coordinates $(0, 0), (0, 1), (0, 2)$.
Turn 1: the user selects $(1, 0)$ and "miss" is printed
Turn 2: the user selects $(0, 0)$ and "hit" is printed
Turn 3: the user selects $(0, 0)$ again and is penalized by losing turn 4. "penalty" is printed
Turn 4: skipped
Turn 5: the user selects $(0, 1)$ and "hit" is printed
Turn 6: The user selects $(-1, 0)$ which is out of bounds. Penalty
Turn 7: skipped
Turn 8: the user selects $(0, 2)$ and "sunk" is printed. The game ends because the last boat has sunk
The total number of turns is 8, but the total number of cannon shots is only 6 (Turns 1, 2, 3, 5, 6, 8).

Before each turn, a visual representation of the current board should be printed to the screen. This allows the player to see which squares they have seen (meaning have fired on), while still obscuring the squares they have not. Printing the board involves printing the locations of the current hits and misses, as well as any locations that have been scanned by the drone. You can format output as a grid, or simply provide a list of information, but it should be easy to understand.

Example:
In the example above this is how the board would might be printed out before these turns. You can choose any representation you want to display the board as long as it is readable and accurate.
Before Turn 2:
- - - - -
O - - - -
- - - - -
- - - - -
- - - - -
Before Turn 8:
X X X - -
O - - - -
- - - - -
- - - - -
- - - - -

4 Powers

There are two special powers that you will implement for this project. Each power can be used **once** a game if the game is in **standard** mode, and **twice** if the game is in **expert** mode. Each power will add one turn to the total turn count, but will not count as a shot fired. If the user attempts to use the power again a message will be printed notifying them that they have already used that power as many times as they can. There is no turn penalty for trying to use a power when you have already hit the limit.

Drone

At any point in the game, once the board is initialized and the mode has been chosen, the user can type in "drone" (or whatever way you want them to signify to the program that they wish to use a drone).

The program will then ask the user if they want to scan a row or column. The user will respond, for example by typing in "row". If the user types in an invalid response, the program should alert them until they type in a valid response.

Next, the program will ask the user which row or column they wish to scan. The user will respond by typing in a number, for example 0. If the user types in a number that is out of boundaries, the program should alert them until they type in a valid number.

The program will then "scan" that row or column, and determine how many spots are there that contain a boat. It will then print that information out to the user. For example in the above

example where a user entered "row" and 0, the program would scan the 0th (or first row) and return the number of boat spots in that row. The drone will count boat spots that have already been hit and sunk.

Example:
User: Types in drone
Terminal: Would you like to scan a row or column? Type in r for row or c for column.
User: row
Terminal: Invalid input. Please type in r for row or c for column.
User: r
Terminal: Which row or column would you like to scan?
User: -1
Terminal: Invalid Input. Please type in a number within the boundaries of the board.
User: 0
Terminal: Drone has scanned 2 targets in the specified area.
User: drone
Terminal: Drone has been used the max amount of times.

Note: The above example is just a potential way that the drone would work during a game. However you wish to have the drone interaction work is fine, as long as it is clear what commands a user has to enter to select the necessary arguments, as well as call the drone.

Missile

At any point in the game, once the board is initialized and the mode has been chosen, the user can type in "missile" (or whatever way you want them to signify to the program that they wish to use a missile).

The program will then ask the user for a coordinate. The user will type in two numbers (Example: 0 5). The program will then check if the coordinate (0,5) is valid for the board. If it is not, the program will continue to ask the user to type in valid coordinates until the user does so.

Next, the program will fire a missile in that spot, what that means is that it will fire at a 3x3 square, with the chosen spot being in the very center of the square. If the chosen spot is near the edge of the board, the missile will only hit the spots on the board.

Example:
User: Types in missile
Terminal: Where would you like to launch your missile?
User: 2 2

In the above case, a missile will launch at coordinates (2,2). This will hit spots (1,1),(1,2),(1,3),(2,1),(2,2),

(2,3),(3,1),(3,2), and (3,3).

Another example is if a user launches a missile at coordinates (0,0). In this case, because (0,0) is near the edge of the board, only the spots that are on the board will be hit. In this case spots (0,0) (0,1),(1,0) and (1,1) will be hit. Note: It is fine if the spot to launch the missile has already been fired upon, or even if all the spots the missile will be hit have already been fired upon. The only invalid missile use is firing outside of the board.

5 Standard and Expert Game Modes

At the beginning of the program, the terminal should prompt the user if they want to play the game in standard of expert mode.

Example:
Terminal: "Hello welcome to BattleBoats! Please enter 1 to play in standard mode or 2 to play in expert".
User: standard
Terminal: Invalid input. Please enter 1 to play in standard mode or 2 to play in expert.
User: 2

To recap the differences between Standard and Expert Mode.

Standard Mode:

- 8x8 Board.
- 5 Boats.
- Each power can be used once

Expert Mode:

- 12x12 Board.
- 10 Boats.
- Each power can be used twice

6 Print

At any point when the user types in "print" after the game begins and the board is set, the program should print the entire board to terminal, displaying spots that have been hit and missed, as well as where boats are. It is important that you differentiate the boats in some way. This will be useful for you to make sure that your board is being set up correctly.

Example: After the user types in print the board is printed out
X X X 1 1
O 2 2 2 2
X 3 3 O O
5 - - - -
5 4 X 4 O

Note: Make sure that by looking at the board that you can tell which boats are which, meaning that the symbol to represent boats aren't all the same, but that they are distinct per boat. (i.e. using 1s for boat 1, 2s for boat 2 etc., instead of using a 1 for every boat)

7 Getting Started

There is a lot to do in this project but we recommend starting by making a BattleBoatsBoard class that contains your 2D Array or board. You should first set up the functionality to correctly size the array depending on if the user selects a standard or expert game mode. Then, determine how to randomly place the ships on the board without having them overlap. Implement the print command to help you see the board and make sure that your ships are being correctly placed.

While again there is no strict requirements on what classes you have to implement and how to implement them, we have a possible class design below.

BattleBoatsBoard Class

- Contains a 2D Array (Can contain ints or strings depending on implementation)
- Contains int variables to keep tracks of total shots, turns, and ships remaining.
- Constructor takes in an int size variable as argument (Will be 8 in case of standard mode, 12 in expert), and correctly sizes the array upon construction.
- Has placeBoats() function to randomly place boats on board.
- Has fire(int x, int y) function to handle firing on a coordinate.
- Has display() function to print out the player board state every turn.
- Has print() function to print out the fully revealed board if a player types in the print command
- Has missile(int x, int y) function to fire a missile on a specified coordinate
- Has drone(int direction,int index) function to scan a specific row or column

BattleBoatsGame Class

- Main should be in here. Should create and initialize a BattleBoatsBoard object. Should use scanner to take in user input until game end.

- Can also contain any helper functions that you may find helpful.

8 README file

Make sure to include a README.txt file in with your submission that contains the following information

- Group member's names and x500s
- If in a group, what did each partner do.
- Any additional or special features your project has
- How to run and compile their code
- Any known bugs or defects in the program

9 Honors

This section of the project has to be implemented only by an Honors students. **If you are not an Honors student, then this section does not apply to you.** This section will carry 5

You must provide a write up of your project detailing the time complexity of the functions you have written. You are expected to write and explain the run time of the functions which deal with *building the board*, *taking a turn (firing a shot)* and *sending the drone*. A careful worded proof of the time complexity must be presented to achieve full credit for this section. A direct answer without an explanation or without a complete proof will not achieve full credit. The writeup must be a *.pdf* or *.txt* file. Submitting the writeup in any other file format may incur a penalty.

IMPORTANT: We will not grade analysis for methods that are not related to building the board, taking a turn, and sending the drone. You can choose to have one single function implementing the above processes or you can have separate functions for each process. Either way, you must provide a run time analysis.