CSCI 1933 Project 2

Battleboats Game

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.

- **Due:** The project is due on **Wednesday, March 1st** by **11:55 PM**.
- **Submission:** Please submit a `.zip` or `.tar` file on moodle containing your `src/` folder with all the `.java` files. Even if you work with a partner, *every* student must submit a project on moodle. Submissions in the incorrect format may receive a penalty. Submissions with missing code will also be penalized. Make sure you submit **all** your source code!
- **Partners:** You may work alone or with *one* partner. Please place you and your partner's name, student ID, and x500 in a comment in each `.java` file you submit. Do not share code with students other than your partner.
- **Code Sharing:** If you use online resources to share code with your partner, please ensure the code is private. Public repositories containing your code are prohibited because other students may copy your work.
- **Java Version:** The TAs will grade your code using Java 8. Please ensure your code works in Java 8, which is the version installed on the cselabs computers. Java 6 and Java 7 will work with Java 8 (at least for the purposes of this course).
- **Grading:** 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.

  Grading will be done by the TAs, so please address grading problems to them **privately**.
- **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:** Please post questions related to this project on the moodle forum. **Do not e-mail the TAs or the class email unless you have private questions.** All questions about how to code Java or questions about understanding this instruction document belong on moodle. However, please avoid posting your answer code on moodle, even if it is not working.

Introduction

Battleboat 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 rectangular $m \times n$ 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 input the dimensions m and n at the beginning of the game, and the program should create a board of m rows by n columns. Errors such as nonpositive input should be checked, but you can assume the input is always two integers. The minimum board size is 3×3 and the maximum is 12×12. Assume that the points in the board range from $(0, 0)$ to $(m - 1, n - 1)$ inclusive.

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 board has been 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. All boats should be of length 3. Recall the smallest board is 3×3 and the largest board is 12×12.

Smallest Dimension	Boat Quantity
`width == 3 or height == 3`	1
`3 < width <= 5 or 3 < height <= 5`	2
`5 < width <= 7 or 5 < height <= 7`	3
`7 < width <= 9 or 7 < height <= 9`	4
`9 < width <= 12 or 9 < height <= 12`	6

Use the table above to determine how many boats to place. Recall that the board may be rectangular, so a board that is 9×3 should have just one boat of length 3 (the first case). 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.

- 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.

In addition to normal cannon fire, the user will have the option to utilize a recon "drone" to scan a small area of the board. You can implement this option however you wish, as long as it is plain to the user what must be done to use the drone. The drone will allow the player to see the contents of the squares in a 3×3 area around the input location, but this will not count as a shot. Instead, the user will lose 4 additional turns while waiting for the drone to return from its mission. Therefore, if the user chooses to use the drone, print "recon". If the user sends the drone out of bounds, the user will receive a penalty for a shot out of bounds, wait 4 additional turns for the drone to return, but the board will not be revealed, since the drone will not return to the user with any valuable information.

If the user receives a penalty (for attacking the same spot twice or for attacking somewhere out of bounds), the user's next turn will be skipped. 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 board is 4×4 and just one boat is placed 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
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 sends the drone to $(1, 1)$ and "recon" is printed. The user will now be able to see the contents of the squares $(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1)$, and $(2, 2)$
Turn 9–12: skipped
Turn 13: the user selects $(0, 2)$ and "sunk" is printed. The game ends because the last boat has sunk

The total number of turns is 13, but the total number of cannon shots is only 6 (Turns 1, 2, 3, 5, 6, 13).

Before each turn, a partially visible representation of the board should be printed to the screen. This allows the player to see which squares they have seen, while still obscuring the squares they have not. Printing the board involves printing the locations of boats as well as their identity and which sections have been hit. You can format output as a grid, or simply provide a list of information, but it should be easy to understand.

4 Debug Mode

The sections above should hide the game data from the user so the user cannot cheat. However, for grading purposes, **you are required to add a debug mode**. The game should ask the user whether to run in debug mode and set a flag internally.

If the game is in debug mode, print the game board on the screen before every turn. This is very similar to the board you would print every turn in normal play, but with the entire board revealed to the user. Additionally, debug mode should differentiate between squares that would be visible to the player and squares that would be obscured to the player. Once again you can format output as a grid or just provide a list of information, provided that it is easily understood.

If the game is *not* in debug mode, do *not* print this information.

5 Honors

This section of the project has to be implemented only by a Honors students. **If you are not a Honors student, then this section does not apply to you.** This section will carry 10% of the total grade for this project for Honors students.

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*, *firing shots* 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, firing shots, 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.