CSCI 1933 Project 1

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 **Friday, February 16th** by **11:55 PM**.
- **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 Moodle 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 Moodle 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 be using unit tests to grade parts of this project. This means you must follow and specified modifiers and signatures exactly as they are specified. Also 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 may result in a maximum of 5 percent extra credit on this project grade – at the discretion of your TAs and professor.
- **Questions:** Questions related to the project can be discussed on Moodle 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 Moodle.
- **Grading:** Grading will be done by the TAs, so please address grading problems to them (e.g. via the email alias or at office hours).

Introduction

Battleboat is a probability-based boardgame 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. **You must implement every part of the description below. This means you must follow and specified modifiers and signatures exactly as they are specified.** Examples of changes that could lose you points: changing a variable to public when it is listed as private, changing the return type of a function, etc.

Your code will 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.

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: Writing useful helper methods for the functions listed above is allowed and encouraged! Remember part of your grade comes from style and readability of the code you write.

Note: 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 Cell Class

The Battleboats game board is composed of many cells. You are required to create a `Cell` class that contains the following private attributes:

- `private int row`: indicates the row value of the `Cell`
- `private int col`: indicates the column value of the `Cell`
- `private char status`: character indicating the status of the `Cell`. There are four different possibilities for this field:

Character	Conditions
' ' (space)	Has not been guessed, no boat present
'B'	Has not been guessed, boat present
'H'	Has been guessed, boat present
'M'	Has been guessed, no boat present

In addition, you are required to implement the following functions:

- `public char get_status()`: getter method for status attribute
- `public void set_status(char c)`: setter method for status attribute
- `public Cell(int row, int col, char status)`: `Cell` class constructor

2 Battleboat Class

A `Battleboat` object will have the following attributes:

- `private int size`: indicates the number of `Cell` objects a `Battleboat` spans. Default this value to 3
- `private boolean orientation`: indicates the orientation of the `Battleboat` (horizontal or vertical, can be randomly decided)
- `private Cell[] spaces`: array of the `Cell` objects associated with the `Battleboat`

And the following functions:

- `public boolean get_orientation()`: getter method for orientation attribute
- `public boolean get_size()`: setter method for size attribute
- `public boolean get_spaces()`: setter method for spaces attribute
- `public Battleboat()`: Battleboat class constructor

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 Board Class

The computer will simulate a rectangular $m \times n$ board. A `Board` object will contain the following:

- `private int num_rows`: indicates the number of rows a `Board` has
- `private int num_columns`: indicates the number of columns a `Board` has
- `private int num_boats`: indicates the number of `Battleboat` objects a `Board` has
- `private Battleboat[] boats`: array of all the `Battleboat` objects associated with a `Board` object
- `private Cell[][] board`: 2-dimensional `Cell` array required to represent the `Board`
- `private boolean debugMode`: flag to indicate if `Board` should be printed in debugMode

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.

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 gameboard 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. As mentioned prior, 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.

Required functions:

- `public Board(int m , int n, boolean debugMode)`: constructor for the`Board`class. This method should assign appropriate number of boats to `num_boats` variable, initialize the`Board` as a 2-D `Cell` array, initialize boats as a `Battleboat` array, place `Battleboat` objects appropriately on the `Board`, and add them to the board's `boats`
- `public int guess(int r, int c)`: returns an int based on the guess for the cell. The statuses of the `Cell` must also be changed according reflect the statuses from the table in the `Cell` class portion of this writeup
- `public int unsunkBoats()`: calculates the number of unsunk boats on the `Board`

4 The main method and debug mode

A main method is provided to you in the `Game` class which will run the game once all the functions are implemented as specified. If the `debugMode` flag is set to false, the view of the board will be obscured during play so that the player only sees the spots they have already guessed. Setting `debugMode` to true may be helpful to you while debugging and testing your code.