CSCI 1933 Project 2: Battleboats

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 **Monday, October 25th** by **11:55 PM** on **Canvas**.
- **Submission:** Please submit a `.zip` or `.tar` file on Canvas containing your `src/` folder with all the `.java` files and the README file. If you are working with a partner, only one student should submit a project on Canvas. 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 your partner's and your names and x500s 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 version 11. Please ensure you have this version of Java installed.
- **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 Discord or the Piazza forum.

Introduction

Battleboats is a probability-based board game that challenges the user to locate enemy boats hidden on a 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**, but the rest of your class and function design is up to you. 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.

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.

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 a game mode at the beginning of the game, and the program should create a board with corresponding dimensions.

- **Beginner**: 3×3
- **Intermediate**: 6×6
- **Expert**: 9×9

Assume that the points in the board range from $(0,0)$ to $(n-1, n-1)$ inclusive.

Note: The `Board` class is also in charge of placing boats. In a regular game, boats should be placed randomly on the board, but if you are running into issues with this method of placement, you may place the boats manually to get the other parts of the game working. In that case you will wait until the rest of the game works before going back and changing the boat placement to random.

IMPORTANT: You are **required** to make a distinct `Board` class as well as classes for the entities `Cell` and `Boat`. Another class `Game` which will contain your `main` method is also required. We recommend using these names, but you can name these classes whatever you like as long as it is easy to understand.

The reason that it is required for `Cell` and `Boat` to be created as separate classes is to allow more control over each cell, as well as the game board as a whole. By creating a separate `Boat` class, it will allow you to differentiate which cells are occupied by which boat, which will be important in determining if a certain `Boat` has been sunk.

Cells

The Battleboats game board is composed of cells. The cell class should contain the following 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 three different possibilities for this field, as shown in the table below.

The following functions should also be implemented:

- `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

'-'	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

Boats

Each boat is represented by a line of consecutive cells 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 cells of the boat have been "hit" by the user.

Example: Given a boat of size 3, valid coordinates for it can be $\{(0,0),(0,1),(0,2)\}$ and $\{(1,1),(2,1),(3,1)\}$. An example of invalid coordinates is $\{(0,0),(0,1)\}$, 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 board has been sized appropriately, **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 number and size of boats are defined by the size of the board (which is defined by the game mode).

Game Mode	Boat Sizes
Beginner (3×3)	2
Intermediate (6×6)	2, 3, 4
Expert (9×9)	2, 3, 3, 4, 5

Use the table above to determine what kind of boats to place. **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. Think about what assumptions you can make to simplify it.

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 an integer. 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 2.0.

Turns

Each turn, the user will input a location x, y to attack on the board. **You can assume the input is always two 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 the 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 receives a penalty (for attacking the same cell twice or for attacking somewhere out of bounds), the user's next turn will be skipped. **The game ends when all of the boats have been sunk. The game should report how many turns it took to sink all of the boats**. Lower scores are better!

Example: Suppose the board is $3 * 3$ 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—and is penalized by losing turn 7.
Turn 7: Skipped.
Turn 8: The user selects $(0, 2)$ and “sunk” is printed. The game ends because the last boat has sunk.

Prior to each turn, a visual representation of the game board should be printed for the user to see. This representation can be formatted as a grid or simply as a list, but it should be easy to read and understand. The representation should display the cells that have been fired on so far and whether they are hits or misses. The cells which have not been fired on should be obscured to conceal the locations of the remaining boats. **The number of Power Points remaining should also be printed out at the beginning of each turn** (explained below).

Powers

There are three special powers you will implement in this project. **Each use of a power takes up one turn and costs one Power Point**. The number of Power Points you start off with (the number of times you can use a power throughout a single game) depends on the game mode selected.

- **Beginner**: 1 Power Point
- **Intermediate**: 3 Power Points
- **Expert**: 5 Power Points

Missile

The first power you will add is the missile attack. At any point in the game, once the board is initialized, 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. It will fire at a 3x3 square, with the chosen spot being the 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: 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). 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: The user should not be penalized if the cell to launch the missile has already been fired upon, or even if all the cells the missile will hit have already been fired upon. The user should also not be penalized if some of the cells the missile will hit are out of bounds as in the previous example. The only invalid missile use is selecting a center spot that is out of bounds, in which case it should continue to prompt the user for valid coordinates with no penalty.

Drone

The second power you will add is the drone attack. 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 “scan” a random row or column, and determine how many spots are there that contain a boat. It will then print that information out to the user. Boat spots that have already been hit and/or sunk will also be counted by the drone.

Example:
User: drone
Terminal: Drone has scanned 2 targets in row 2.
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 call the drone.

Submarine

The final power that will be added is the submarine attack. At any point in the game, once the board is initialized, the user can type in "submarine" (or whatever way you want them to signify to the program that they wish to use the submarine).

The program will then ask the user for a coordinate. The user will type in two numbers (Ex. 2 3). The program will then check if the coordinate (2,3) 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 execute a submarine attack on this location, regardless if the user has already guessed this location.

- If there is no boat at the location, print "miss".
- If there is a boat that has not been sunk at the location, then every location on that boat is hit. Afterwards, print "sunk".

Example:
User: Types in submarine
Terminal: Where would you like to attack with the submarine?
User: 2 3

In the above example, suppose that there is a `Boat` located on coordinates {(1,3), (2,3), (3,3), (4,3)}. Since the location (2,3) hits this boat, all of {(1,3), (2,3), (3,3), (4,3)} would be fired at.

If instead there was no boat on (2,3), then the submarine will have missed and "miss" would be printed to the screen.

Note: Like with the missile, the user should not be penalized if the coordinate they select has already been fired upon. If the input coordinate is out of bounds, the program should just continue to prompt the user for valid coordinates without penalty.

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 means printing the locations of boats as well as which sections have been hit. Provide some way to at least roughly identify which unhit cells belong to which boat (such as printing the boat length in the cell) so the debugging user can tell how close a boat is to sinking. You can format output as a grid or provide a list of information, but it should be easy to read and understand.

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

Getting Started

There is a lot to do in this project, but we recommend starting by making the `Board` class that contains a 2-dimensional array representing your game board. You should first set up the functionality to correctly size the array depending on the game mode the user selects. Then, determine how to randomly place the boats on the board without having them overlap. Implement the `display()` command to help you see the board and make sure that your boats are being correctly placed.

Board Class

- Contains a 2D Array of `Cell` objects
- Contains a 1D Array of `Boat` objects
- Contains `int` variables to keep tracks of turns and boats remaining.
- Constructor takes one `int` variable as an argument (depending on the game mode) 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 (This would be used for debugging purposes)
- Has `missile(int x, int y)` function to fire a missile on a specified coordinate
- Has `drone()` function to scan a random row or column
- Has `submarine(int x, int y)` function to launch a submarine attack at a specific location.

Cell Class

- Contains `int` variables for the row and column of the `Cell`
- Contains a `char` variable to keep track of the status of the `Cell` (whether it has been guessed and/or if there is a boat present)
- Getters and setters pertaining to the row and column of the `Cell`

- Getter and setter pertaining to the status of the `Cell`

Boat Class

- Contains an `int` size to indicate its length
- Contains a `boolean` orientation to indicate whether the boat is horizontal or vertical
- Contains a `Cell[]` array corresponding to where the boat is on the board
- Constructor initializing all of the necessary attributes
- Getters and setters for necessary attributes

Game Class

- The `main` function should be in here. Should create and initialize a `Board` object. Should use `Scanner` to take in user input until game end.
- Can also contain any helper functions that you may find helpful.

README

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

- Group member's names and x500s
- Contributions of each partner (if working with a partner)
- How to compile and run your program
- Any assumptions
- Additional features that you implemented (if applicable)
- Any known bugs or defects in the program
- Any outside sources (aside from course resources) consulted for ideas used in the project, in the format:
  - idea1: source
  - idea2: source
  - idea3: source
- Include the statement: **"I certify that the information contained in this README file is complete and accurate. I have both read and followed the course policies in the 'Academic Integrity - Course Policy' section of the course syllabus."** and type your name(s) underneath

There is a ten-point deduction for a missing or incomplete README.