CS1027

ASSIGNMENT 1

Computer Science Fundamentals II

This assignment is due on Tuesday, September 29, 2020 by 11:55pm. See the bottom for submission details.

Learning Outcomes

To gain experience with

- Creating classes with simple methods
- Using basic arrays and multi-dimensional arrays
- Algorithm design

Introduction

Battleship® is a classic two-player board game in which each of the players secretly place a number of different-sized ships onto a grid board and then take turns guessing where their opponents' ships are located. To make a guess, a player states a grid cell, like A4 or F10, and the opponent has to respond with "hit" or "miss" to indicate whether that guessed cell contains a piece of a ship or not. They would also indicate if the hit destroyed the ship, meaning that every piece of the ship has now been hit. Players alternate such guesses until one of them has successfully destroyed all their opponents' ships, and that player is declared the winner.

For this assignment, you are required to create a simplified one-person version of the Battleship game. Several classes are provided to help you get started, but the project is missing a few important classes so you will have to create and code those classes to complete the game program.

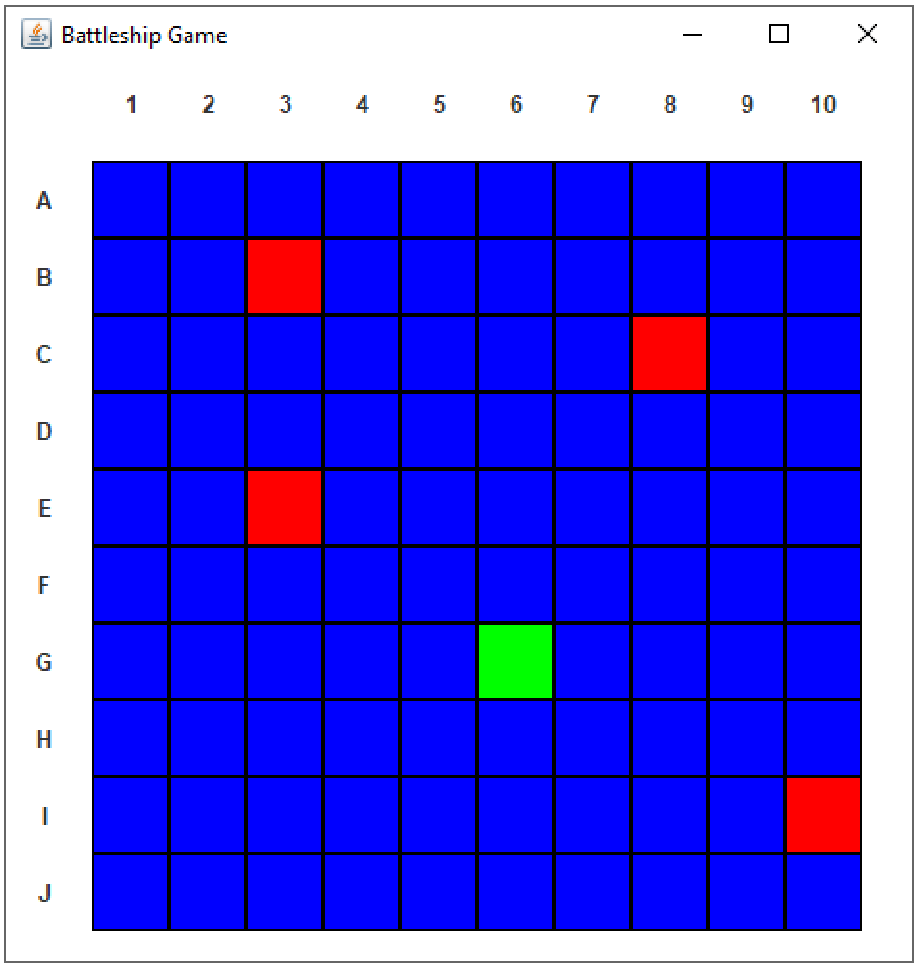


Figure 1. A screenshot of the finished game with several guesses to locate the hidden ships. Cells coloured red (B3, C8, E3, and I10) are "missed" guesses, the cell in green (G6) indicates a "hit" (a ship is located either horizontally or vertically through cell G6), and all the other cells coloured blue have not yet been guessed so they remain a neutral colour.

Provided files

The following is a list of files provided to you for this assignment. Please do not alter these files in any way.

- Config.java – provides some configuration settings and helpful methods
- GameGUI.java – provides the visual GUI for the program
- Play.java – provides a simple method to start playing a one-player game
- ShipRandomizer.java – places the ships randomly and secretly on the board
- TestGame.java – provides several tests to check that the program is working correctly

Classes to implement

For this assignment, you must implement 3 Java classes: *Ship*, *Board*, and *Game*. Follow the guidelines for each one below.

In all of these classes, you can implement more private (helper) methods, if you want to, but you may **not** implement more public methods. You may **not** add instance variables other than the ones specified below. Penalties will be applied if you implement additional instance variables.

Ship.java

This class represents one ship in the game. It contains information about the ship, like its size dimensions, and some simple methods. NOTE: length means the number of horizontal cells the ship takes up, while the width is the number of vertical cells the ship takes up. For example a ship that has length 3 and width 1 is placed horizontally taking up 3 adjacent cells in a row. If that ship was placed vertically instead, the length would be 1 and the width would be 3.

The class must have the following *private* variables:

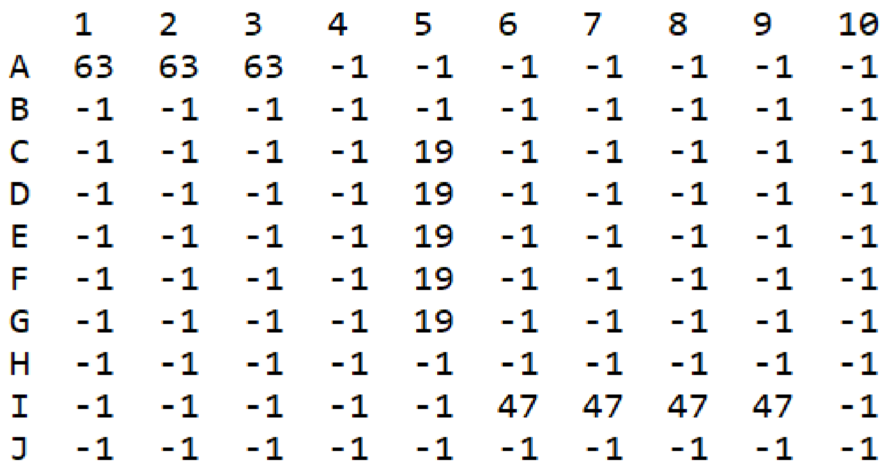
- ID (int)
- length (int)
- width (int)
- remainingCells (int) Note: this variable will start at the total number of cells taken up by the ship, and will decrease each time the ship is hit until it reaches 0

The class must have the following *public* methods:

- Ship (constructor) – takes in 3 parameters for ID, length, and width and assigns their values into the corresponding instance variables. The remainingCells variable must be set based on the total number of cells the ship takes up.
- getID – returns the ID
- getLength – returns the length
- getWidth – returns the width
- getRemainingCells – returns the number of remaining cells
- takeHit – decreases the remainingCells to indicate a hit on the ship

Board.java

This class represents the board for the game to keep track of all the ships. The grid variable is a 2-dimensional array (matrix) which will store the board's contents. By default, all the cells must have -1 to indicate they are empty. As each ship is added, all the cells taken up by that ship must be changed from -1 to the ID of that ship. See the following figure for an example.



```
    1   2   3   4   5   6   7   8   9   10
A   63  63  63  -1  -1  -1  -1  -1  -1  -1
B   -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
C   -1  -1  -1  -1  19  -1  -1  -1  -1  -1
D   -1  -1  -1  -1  19  -1  -1  -1  -1  -1
E   -1  -1  -1  -1  19  -1  -1  -1  -1  -1
F   -1  -1  -1  -1  19  -1  -1  -1  -1  -1
G   -1  -1  -1  -1  19  -1  -1  -1  -1  -1
H   -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
I   -1  -1  -1  -1  -1  47  47  47  47  -1
J   -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
```

Figure 2. A representation of the grid with 3 ships added to the board. Ship with ID #63 is added horizontally from A1 (0, 0). Ship with ID #19 is added vertically from C5 (4, 2). Ship with ID #47 is added horizontally from I6 (5, 8). All other cells retain their default value of -1.

The class must have the following *private* variables:

- width (int)
- height (int)
- shipNum (int)
- ships (Ship[])
- grid (int[][])

The class must have the following *public* methods:

- Board (constructor) – takes in 2 int parameters for width and height and assign them to the corresponding instance variables. Set shipNum to 0 to begin and initialize the ships array with 10 slots (we'll assume there will never be more than 10 ships in a single game – the typical amount is 5). Initialize the grid multi-dimensional array using the width and height, and then loop through all the grid cells to initialize them all to -1.
- addShip – take in 3 input parameters: a Ship object, and int positions sx and sy to indicate where in the board we are attempting to add the ship. This method will return a boolean to indicate the status from the attempted ship insertion. If we have already added the full capacity of ships to the array (i.e. 10), then immediately return false. If the sx or sy are out of bounds, immediately return false. Using the ship's dimensions, determine if the ship would extend out of bounds from the given sx,sy location. If even one cell of the ship would be out of bounds, return false. If any of the new ship's cells would overlap with an existing ship on the board, return false immediately. If none of the

above conditions were met, then add the ship by changing all the -1 values in the grid to the ship's ID, add the ship to the array, increment shipNum, and return true to indicate the successful insertion.
- getGrid – returns the grid
- getCell – takes in 2 int parameters indicating the x and y (or row and column) indices of a cell in the grid and returns that cell. You can assume these indices are within the bounds of the grid.
- checkDestroyedShip – take in 1 int parameter representing a ship's ID. Find the ship with the given ID (note that these IDs are not regular array indices) and then call the takeHit method on that ship. Check if the number of remaining cells of that ship has reached 0. If so, return true to indicate that the ship is destroyed, otherwise return false because the ship was hit but not completely destroyed.

Game.java

This class, as its name suggests, will be the main heart of the program. It will be the entry point of the program, read in the file of cities and create objects for each of them, contain the array of those cities, as well as perform tasks that interact with the GUI.

The class must have the following *private* variables:

- board (Board)
- gui (GameGUI)
- isTesting (boolean)
- activeShipCount (int)

The class must have the following methods:

- Game (constructor) – take in a boolean input parameter to indicate whether the game is to be run in testing mode or regular play mode (true = testing, false = play mode). Assign this value to the corresponding instance variable. Initialize the board using size the sizes given in Config, and set the activeShipCount to 0. If we are in play mode (using the testing boolean mentioned above), then call the placeShips method in ShipRandomizer to make the computer automatically place the ships. Nothing further is needed in testing mode.
- initializeGUI – instantiate the gui variable. Look at the GameGUI constructor to determine what needs to be sent in here.
- addShip – take in 3 input parameters: a Ship object, and int positions sx and sy to indicate where in the board we are attempting to add the ship. Call the corresponding addShip method in the Board class from here. If the ship is successfully added, increment the activeShipCount. Return a boolean to indicate whether or not the ship was added successfully.

- shootTarget – takes in a String input parameter "target" which will be represent the cell guesses made in the game, i.e. "F7". Use the provided methods from Config and your checkDestroyedShip method in Board to help you determine if the given target is valid, and then to shoot at that cell if it is valid. This method must return an int and the value to return must indicate the status of the attempted shot, given the following rules:
    - -1 if the given target is not valid (i.e. out of bounds or not the correct format)
    - 0 if the shot hit open water (miss)
    - 1 if the shot hit a piece of a ship
    - 2 if the shot hit the final piece of a ship, rendering it destroyed
    - 3 if the shot hit the final piece of the last remaining ship of the fleet

Marking notes

Marking categories

- Functional specifications
    - Does the program behave according to specifications?
    - Does it produce the correct output and pass all tests?
    - Are the class implemented properly?
    - Are you using appropriate data structures?
- Non-functional specifications
    - Are there Javadocs comments and other comments throughout the code?
    - Are the variables and methods given appropriate, meaningful names?
    - Is the code clean and readable with proper indenting and white-space?
    - Is the code consistent regarding formatting and naming conventions?
- Penalties
    - Lateness: 10% per day
    - Submission error (i.e. missing files, too many files, etc.): 5%
    - "package" line at the top of a file: 5%

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through cheating-detection software.

Submission (due Tuesday, September 29, 2020 at 11:55pm ET)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the assignment.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- Ship.java
- Board.java
- Game.java