COMPSCI4039 – Programming IT

AE2: Battleship

There is a FAQ at the end of this document, please read it.

Introduction

Battleship is a popular board game involving 2 players. The aim of the game is for the players to place their ships on a grid and then guess the positions of their opponents' ships. Full rules can be found here. https://www.thesprucecrafts.com/the-basic-rules-of-battleship-411069

In this assessment you are being asked to program a simpler version of this game. Initially, the battleships will be placed on a 10x10 grid randomly, players will then proceed to guess where the ships are on the board. If they guess correctly, they 'hit' the battleship, otherwise they 'miss'. The objective of the game for both players is to find and sink all the battleships on the board. The player who hits more ships than the other, wins after all ships have been destroyed.

In this assessment you will create classes and logic necessary to run a game of battleship.

This assessment is split into multiple sections. Partial marks will be awarded in this assessment, so you do not need to complete the project in full to be awarded marks. A maximum number of marks that can be awarded for each section is shown to help you manage your time with this assessment.

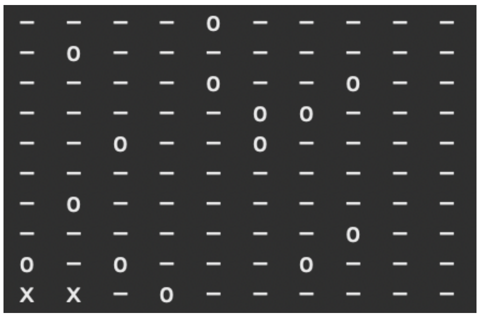
The marking scheme is below. Marks will be awarded for:

- Functional correctness
- Sound OO design
- Clean, tidy and commented code

The marks are awarded out of 22. Remember that you do not need to complete the project to be awarded marks. You do not need to complete a section or task to completion to be awarded marks. Partial marks will be awarded in the event you can demonstrate an attempt that was made to address the task.

Part 1: The Players and the Square classes

This following image provides an idea on what the board should look like:



```
- - - - o - - - - -
- o - - - - - - - -
- - - - o - - o - -
- - - - - o o - - -
- - o - - o - - - -
- - - - - - - - - -
- o - - - - - - - -
- - - - - - - o - -
o - o - - - o - - -
x x - o - - - - - -
```

There are 10 rows and 10 columns. Each cell in the board is a `Square` object. Each square object must consist of:

- An integer denoting the row position, inclusive of 0.
- An integer denoting the column position, inclusive of 0.
- An attribute to indicate whether the square has a ship in it.
- A reference to a battleship if one is currently on the Square.
- An attribute to indicate whether the player has fired a shot at the square.

You will also require appropriate getters and setters to utilise this class within the assessment.

Each player class should have a reference to a board object they will play the game with. Their name, and scores. Make these attributes private. The constructor to the Player class should take in the name of the player, along with a reference to the board that will be associated with the player object.

Each player will have a method to take turns. This method will be called from the main controlling logic of the game.

[5 marks]

Part 2 – The board and Battleship classes

You are to specify a Battleship class. This class should specify the following attributes:

- Whether it is sunk.
- The remaining health (how many more hits it can take before sinking)
- The size of the ship.

Each Battleship object will be checked against whenever a player fires into the board on a particular square. In the event the player hits the ship, it will be necessary for you to decrease the health of the ship and of course update its state in the event the ship is destroyed. A battleship should be specified as being 2 units in size, therefore it is to take up 2 squares when it is placed on the board. The size of the ship will directly correspond to the amount of health the ship has.

The board constructor should take in two values representing the number of rows and columns, again you can assume that these values will be 10 when testing your program. The board class itself should have two methods, one to populate the board data structure with `Square` objects. The other will be responsible for generating battleships.

When generating battleships, the placement of the ships should follow certain restrictions. Of course, you should ensure in your code that the ship length does not mean it will exist out of bounds of the board. In addition, you are to ensure that ships do not overlap when they are placed on the board. Placement of the ships **must be randomised**. You can use the following code to generate a random generator.

```
Random r = new Random();
r.nextInt(n);
```

In addition to this, you are to also have some ships placed vertically on the board, whilst others are horizontal. Again, this should be randomised, you can use the following 'coin flip' code to achieve this:

```
Random r = new Random();
Return r.nextBoolean();
```

You are to place 5 battleships on the board at random, in random orientations.

[5 marks]

Part 3 – Playing the Game

When creating `Player` objects, the users should be prompted for their names and have this be used as input to the constructors. Players have a `takeTurn` method, which should be called from a main game loop within the main class. This method should prompt the user to input their guess from the console. You can assume that the user will input their guess as 'x y' where x represents the row and y represents the column that is to be targeted. The traditional game uses letter + number combinations, you are not expected to use this when requesting player input. Column and row numbers will be fine.

Within the main class, add logic to allow players to take turns by implementing the `takeTurn` method. You do not need to check for improper input and can assume that users will enter correct information that is within bounds. The `takeTurn` method should return a boolean, if true then the player has hit and sunk the final battleship and has ended the game. Otherwise, the game continues, and the method returns false.

The player can only damage the ship in the event they land a fresh hit on it (i.e. they cannot hit the ship in the same square twice and have it count. Your code must check for this and prevent players from damaging the ship multiple times in the same square. Likewise, your code should also check for areas that have already been fired at. In the event the player enters coordinates that have been checked prior, they will lose their turn.

After each turn, the game should announce an update to the player (prompting the player for their guess, and indicating whether the guess either hit, missed, is invalid or has ended the game. If a player sinks a ship, they score a point. The game ends when a player sinks the last ship, so you will want to track how many ships have been sunk, and how many are left in the game. The winner is the player with the most points. If the points are equal, then it is a draw.

[4 marks]

Part 4 – Visualising the Game

The board class should have a method, `toString` that will allow the current state of the board to be displayed to the console. The minimum information necessary is presented in the figure in this document. It is not necessary to have extra information to score full marks.

The `toString` method should invoke the `toString` representation of each `Square` object within the data structure. The `Squares` class should have a `toString` method that will return a representation of the square. If the player has not interacted with a square then the representation should be '-', if the player has fired and missed, then the square representation should be 'o' and finally 'x' if they have hit a ship. Each square should be printed with `String.format` to exactly 3 spaces in length.

On each turn, the grid is to be updated to reflect the changes the players make to the board. So if they fire at a square and miss, then this should immediately be announced and the board updated before the next player takes their turn. Likewise, if they hit a ship then the board should be updated to reflect this and a hit should be announced. A `Square` object can have a battleship associated with it and this information should be used to inform the `toString` method in the `Square` class.

[4 marks]

Part 5 – Expanding on the Battleships

Expand on what you have implemented with the Battleship class to specify multiple sub-classes. A `SmallBattleship`, `MediumBattleship` and `LargeBattleship` should all inherit from the Battleship superclass.

Each sub-class should specify to the super class constructor specific information that will be hard coded and therefore specific to that sub class. In addition, you are to specify a static attribute for each sub class. This attribute should specify the total number of permissible ships of that class on the board. The following information should be associated with each class.

- `SmallBattleship` should have a size of 1, and there must be 3 on the board.
- `MediumBattleship` should have a size of 2, and there must be 2 on the board.
- `LargeBattleship` should have a size of 3, and there must be 1 on the board.

Update your placement logic to take into consideration the different battleship types that may exist on the board. Therefore, you should no longer have 5 ships that are of length 2, but you should now have 3 ships that are a single square in size, 2 that are 2 squares in size, and 1 ship that is 3 squares in size. You should use the static attributes for this. Again, ensure that these are randomly placed, and that there are no overlaps or placements that are out of bounds. Each battleship object is still worth 1 point when sunk.

[4 marks]

FAQ

1. *Is it important to make flashy graphics for my board?* No. Functionality is what we're after here.
2. *Should I comment my code?* Yes. Your code is the only thing you will submit. We reserve the right to penalise because we cannot understand, even if the code works. Provide clear comments.
3. *Should I provide unit tests?* You can get 100% of the marks without them. However, writing tests will help you to get things working quicker. If you do write them, feel free to submit them. However, you are not expected to write unit tests.
4. *If I can't get something to work, what should I do?* We can sometimes award marks for you telling us (in comments) how you might get something to work so if something isn't working, tell us why you think that is.
5. *How long should I spend on this assessed exercise?* Hard to say. However, we find every year that students spend **way** too long on AEs.
6. *I want to extend my code more as programming practice, any suggestions?* Great! It's a good way to improve your programming. One thing you could do is to consider creating additional ship classes to better reflect the original game of Battleship. You could also update the scoring logic to say, have different ships award different points. You could also if you really wanted to, work on a GUI that could be used here. However please note that if you want to work on this extra content **don't submit these extra classes – we need a clean version that stays within scope that we can mark precisely! (you will not be awarded extra marks for doing this.)**
7. *Will you post an example solution?* Yes. Once we have all the submissions and have completed the marking. This might be a while after the deadline (people get extensions for being ill etc.) so please be patient! We will devote an examples class to going over our model solution.