

UNIVERSITY OF MICHIGAN
EECS 282

Assignment 2 – Spaceship Battle Game

Due Date: 1st Stage Due: Monday, January 19th at 11:00 PM

Overview

In the next few assignment, you will be working towards designing, testing, and implementing a game. This game is similar to Battleships that you may have played before. It is also similar to “Kill a Dot Com” in Head First Java. Feel free to read the book if you have it (Chapter 5) and take advantage of the stuff in the book. *Don’t worry – you will build the game in stages, starting out with a very simple version that you can complete within a week, and then gradually extending it till it turns into a game that you will actually enjoy playing.*

We will give you 0th version of the game as a starting point. You will use some aspects of a development paradigm called Extreme Programming (or Agile Development), which sounds scary, but is actually quite easy to do. Three key principles are that you (1) strive to always have something that works (even if it does very little) and add functionality gradually; (2) you write the test cases first before you code; and (3) refactor or improve your code design whenever you see an opportunity. In practice, Extreme Programming has been found to be very effective in producing better code. It may also help to form some friendships in the class to bounce off ideas and discuss your design/code, though you should all write and submit your own version (acknowledge help from others and the extent of collaboration or help).

The end goal of this assignment and next two stages of this assignment

Klingons with cloaking devices have invaded our galaxy (assume to be a 2-dimensional grid for simplicity). They have placed spaceships of different shapes and sizes in this 2-D grid. Your goal is to design and implement a game in which the user guesses where the ships are and attempts to shoot them down in the minimum number of shots. To kill a ship completely, you have to ship all the grids that it occupies. Coordinates start from (0,0). We will develop the program in stages, but first we give an overview of where we want to end up with. **The game ends when all the ships have been destroyed.**

In the full-blown version of the program, the user is told how many spaceships there are and of what types. You will support multiple types of ships, e.g., ships of 4 consecutive squares, ships of 3 consecutive squares, and a ship of 2x2 square shape.

The main program shall initialize the grid, place one or more ships on the grid (randomly). We will allow ships to overlap (imagine a ship behind another). Shooting on a square that contains multiple ships will count as a hit on all those ships (imagine you have very powerful missiles that go through ships easily). **But, a ship shall be contained entirely within the grid. Attempt to place a ship in any other location should be ignored.**

Your program should print out how many ships there are and their shapes (e.g., XXXX denotes a ship that occupies 4 squares in a row). It then should go into a loop that asks the user for a guess. It should print out whether the guess is a hit or a miss, as well as the grid, with X for squares that have not yet been shot at, a number for the square that had a ship in it and has been hit (the number indicates the number of hits at that square in case the ships overlap), and 0 for a square that did not have a ship, but has been hit. When all the ships have been destroyed, it should also print out the number of hits and the number of misses.

1st Stage of the Assignment: Due Monday Jan. 19th at 11 PM. (20 points).

To start out, your goal should be to first write a simplified version of the game in which the grid is 1-dimensional of length 10 (positions 0 to 9) and there is only one type of ship that occupies 4 squares. Before you write any code for the game though, you should think about the classes you need and the methods that you will need. Then, next write the test cases. **A program to test a class must be written before you write the code for a class -- this is a good practice to get into.** It is always a good idea to write the test programs before you write the code. In Eclipse, there is a huge advantage of writing a test case first. You can put your mouse over the functions you use in your test program and it will automatically offer to create stubs of those functions for you.

I will post some code that you can start out with and we will go over its design in the lecture or the lab. This code does not output the grid. But, it attempts to track the number of hits and misses. It uses a ship that occupies 3 squares and the program stops when 3 hits have occurred. The code also contains some test cases. You should add additional test cases so that one can be confident that the program will meet the specs in a 1-D case. If you start out with the sample code, you should work on improving its design as you go (that is called *refactoring*) so that future changes are easier.

The basic ideas in the design of the sample code are the following:

1. You need some way to know where the ship is. The sample code simply notes the starting position of the ship (on a 1-dimensional grid). It assumes that the ship occupies 3 slots (which you need to change to 4).
2. To test the game, we first need to create a game, and then initialize a ship on it by supplying its position. Then, the program should go into a loop that receives user's guess, checks it, and prints out the hits/misses and the grid's current status. `SpaceshipTestDrive` contains some of our tests. You should modify or add tests so that the program will be thoroughly tested. I have indicated in **bold** in Overview some of the aspects that you may want to test for more thoroughly.

Here is a sample output from a program that you are given as a starting point:

```
There is one ship of shape:
xxx
Please enter your guess: 3
Sorry, you missed!
```

```
Please enter your guess: 6
Sorry, you missed!
Please enter your guess: 5
Sorry, you missed!
Please enter your guess: 8
Sorry, you missed!
Please enter your guess: 1
You got a hit!
Please enter your guess: 2
You got a hit!
Please enter your guess: 0
You got a hit!
Game is complete
# of hits: 3
# of misses: 4
```

You should also add in a line in the printout after every guess that shows the status of the grid. For example,

```
Grid: x110x00x0x
```

should be printed out after the hits on 1 and 2 in the example above.

What to submit:

1. A test case class, `SpaceshipGameTestDrive`, with a main program that calls all the tests, along with sufficient comments or an additional writeup so that it is pretty clear what the test case does and what behavior it is expecting from the classes. The tests should use assert statements, wherever possible, to detect failure. This means that you will not be able to test whether the program is printing out the right output, only whether the class methods are working correctly. That is fine. You should define sufficient class methods so that the testing using asserts is possible.
2. Code for the `SpaceshipGame` class and any additional class you create. The `SpaceshipGame` class should have a main program that interacts with the user.
3. Screen snapshots of a run of the test and a run of the game.
4. Acknowledgements. Who did you give help to? Who did you receive help from? Any code that you shared with others (explain) or copied from external sources? (If you do not acknowledge, it may be considered cheating if we find strong similarities). We encourage working in pairs to bounce off ideas, discuss testing, and class design, though each of you should write code and comments in your own style.

2nd Stage of the Assignment: Due the following week. (more details to be provided later).

If you are done with Part 1, try to get started on this part.

Then, subsequent to that, you will extend the assignment to support a 2-dimensional 10x10 grid.

Next, you will extend the assignment to support multiple ships of the same class.

First, you should add/modify the test cases. That will help clarify the design as well. Then, you should add/modify the classes for the solution.

Third Stage of the Assignment: Due the following week (more details to be provided later).

Finally, you will support at least three types of ships: ships of 4 consecutive squares, ships of 3 consecutive squares, and a ship of 2x2 square shape. Call these classes Ship4, Ship3, and ShipSquare. They should be inherited from a common class Ship.

For simplicity, assume that the ships are only laid out at 0 degree angle. For example, a Ship4 or Ship3 will lie horizontally. Thus, we just need to know the starting coordinate to know where the ship is.

Also, it is OK for ships to overlap. In that case, if the user hits a square that belongs to two ships, both ships will be hit. (This should actually simplify the design since you can place the ships anywhere on the grid without worrying about overlaps).