CS 2110: Assignment 8

Joe, Matt, Liz, Jason, Ramya, Adam, and Lucas

April 26, 2017

Due Date will be posted on CMS
TL;DR: There is no TL;DR. Read the entire document. Any questions that have been answered in this document will not be answered on piazza.

1 Overview

We have received a distress call from a spaceship (the USS Python) that just crash-landed on an unknown planet somewhere in space (Planet X). Fleet Admiral Gries has dispatched you, as Captain of our fastest spaceship, to rendezvous with the ship and provide any necessary assistance. You will explore this uncharted region of space and return to Earth. Something suggests this may not be as trivial as it sounds ... The assignment has two phases, each of which involves writing a method in Java.

Figure 1: Space, the final frontier

2 Collaboration Policy and Academic Integrity

"Alone we can do so little;
together we can do so much."
—Helen Keller

You may complete this assignment with one other person. If you plan to work with a partner, login to CMS and form a group as soon as possible —at least a day before you submit the assignment. Both people must do something to form the group: one proposes and the other accepts.

If you complete this assignment with another person, you must actually work together. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should both take turns driving —using the keyboard and mouse to input code.

With the exception of your CMS-registered partner, you may not look at anyone else's code, in any form, or show your code to anyone else, in any form. You may not show or give your code to another person in the class. You can talk to others about the assignment at a high level, but your discussions should not include writing code or copying it down.

If you don't know where to start, if you are lost, etc., please see someone immediately: a course instructor, a TA or a consultant. Do not wait. A little in-person help can do wonders to get you unstuck.

3 The Rules and the TODOs

"Ipsa scientia potestas est."

—Sir Francis Bacon

Read the method specs provided for you in all the Java classes — we tell you later what classes to look at. A famous person once said, "Half of this assignment is writing the code. The other half (which should happen first) is understanding what is provided and using it."

Phase 1, the rescue phase, involves writing method `rescue()` in class `MySpaceShip`. Phase 2, the Return-to-Earth phase, involves writing method `returnToEarth()` in the same class.

We expect you to strictly adhere to a consistent style throughout the code you write.

3.1 Definition: Travel Time

The metric by which we will be measuring the optimality of your code is the travel time that your spaceship takes to complete both phases. Your spaceship has a speed (initially 1.0 light-year per year), and each edge between 2 planets has a length (in light years).[1]

3.2 Rescue Phase

Implement `rescue()` in `MySpaceShip.java`.

[1] fun fact: a light-year is a unit of distance. Your ship can travel faster than the speed of light.

On the way to Planet X, the layout of space is unknown. You know only the state of the Planet you are currently on, its neighbors, and perhaps anything that you may remember.

Your goal is to make it to Planet X, with the shortest travel time possible. Note: due to limitations with your spaceship, you can fly only from a planet to its neighbors.

Thankfully, the USS Python has a beacon that emits a continuous signal, which can be heard throughout the known universe. Due to budget cuts, this distress signal contains no information. However, the eggheads have determined that the strength of the signal is inversely proportional to the distance between your current location and Planet X. Look at function `getPing()` in `RescueStage.java` (implemented in `Board.java`). Feel free to use this information however you see fit.

3.3 Return-to-Earth Phase

Implement `returnToEarth()` in `MySpaceShip.java`.

Upon successful rescue of the USS Python, its Captain, who has perhaps 1 year of coding experience, tells you that some of the planets have been occupied by bad guys that don't like you. (Why? Perhaps they're envious of your 56 years of programming experience. We don't know.) Other planets have speed upgrades because they are good guys.

Fortunately, the USS Python has intel on every planet in the system, including whether they are hostile and/or whether they have upgrades for your ship. Your goal is to return to Earth in the **fastest travel time** possible and report to Gries of this disturbing development. Here are the rules governing travel time:

1. You start out at speed 1.0.
2. The travel time from one planet to a neighbor is the distance divided by your speed of travel.
3. If a ship is on a planet with a speed upgrade, it can pick up a speed upgrade, thereby increasing its speed by 0.2.
4. If a ship is on a hostile planet, its speed is decreased by 0.2.
5. If a ship lands on three separate hostile planets/nodes, it blows up and cannot reach Earth.
6. A planet could be **both** hostile and have a speed upgrade

4 What you can do

"The world is your oyster."
—William Shakespeare

This is your chance to do what you want. You can write helper methods (but specify them well). You can add fields (but have comments that say what they are for, what they mean). You can change the shortest-path method in class Paths to do something a little different. Whatever.

We suggest FIRST getting a solution that is simple and works. That gives you a minimum grade of about 85. Save this version so you have something to submit.

Then, begin looking for ways of optimizing —always making sure you have something that works that you can submit.

For debugging purposes, we provide two print wrappers `outPrintln` and `errPrintln`. These print only if `shouldPrint` is true.[2]

N.B.: in each stage, you are given an instance of a proxy class, which purely implements the interface (either RescueStage or ReturnStage). You CANNOT cast this class to the other interface —it will fail— and you CANNOT edit anything outside of the student package per the rules of this assignment.

5 Setting up Eclipse

"A journey of a thousand miles begins with a single step."

—Laozi

We have provided you enough instructions on the last few assignments. You are capable of setting up the package yourself this time. We believe in you :)

The only thing we will say is: When adding the release code to an Eclipse project, be sure to **copy the files** and not link them.

6 Running the Program

The program can be run in two ways, using classes in package `controllers`:

1. By running method main in `HeadlessDriver.java`. This runs the program without the GUI, instead using print statements for some of the more important state changes.

2. By running method main in `GUIDriver.java`. The GUI runs, and you should be able to see your spaceship move. This may be helpful for debugging.

By default, the program runs on a single map with a random seed.

An optional argument flag can be used to run the program slightly differently.[3] `-s <seed>` runs the program with a predefined seed. This allows you to test your solutions on particular maps that can be challenging or that you might be failing on and thus is very helpful for debugging.

[2] Refer to What to Submit for why you should use them.

[3] We showed you how to use these in one of the early recitations.

6.1 How to read the code

There are a lot of files, don't panic! Read them in the following order:

1. `models.Spaceship.java` and `student.MySpaceship.java`
2. `models.RescueStage.java` and `models.ReturnStage.java`
3. the public methods in `models.GameState.java`

This should give you a good overview of: what we want you to implement, and the methods that may help you.

7 The GUI

The GUI was created to make this assignment more visually appealing. The Interface itself is made up of two main components, a screen and a side panel. Out of the box, running method main in class `GUIDriver` will generate a GUI, which will fail since you have not implemented the two deliverables.
For debugging purposes, a slider allows you to change the speed at which the spaceship travels across the GUI. The higher the speed, the faster the spaceship travels.[4]
The side panel contains many useful statistics that you can analyze. Additionally, during **only** the return phase, the borders of certain planets may change to reflect the presence of hostility or speed upgrades. The left border of a planet will turn green if it holds a speed upgrade, while the right border of a planet will turn red if it is hostile.

7.1 How to Run the GUI

1. Open `GUIDriver.java` and hit run.
2. A GUI will pop up. In the top left hand corner, click `File` – `Start`.
3. The Spaceship (represented by a circle) will start moving along an edge away from Earth if everything goes well.
4. If `rescueStage` has been implemented, it will look for Planet X (listed as Planet X on the GUI). Note that both Planet X and Earth are twice as large as the rest of the planets.
5. Once it finds Planet X, if `returnStage` has been implemented, it will return to Earth.
6. ALWAYS CLICK **reset** or **new random seed** before clicking **start**. The behavior of clicking **start** multiple times in a row without **reset** or **new random seed** is undefined.

[4] Relevant meme https://i.imgur.com/E1YRsXQ.jpg

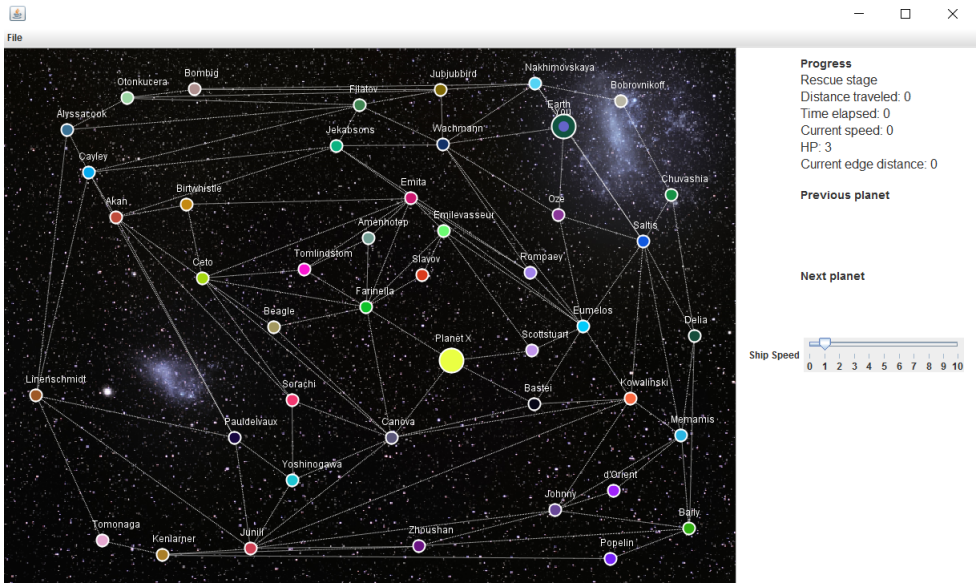

Figure 2: Example GUI screen

8 Grading

The vast majority of points on this assignment will come from a correct solution that always finds Planet X and returns to Earth safely (without timing out), so your priority should be to make sure that your code always does this successfully.

However, in order to receive full credit for the assignment, your solution must also get a reasonably high score (minimizing total time travelled in Phase 1 and Phase 2), so spend some time thinking about ways to optimize your solution.

The use of Java Reflection mechanisms in any way is strictly forbidden and will result in significant penalties.

Note that if you run your solution with `headlessDriver.java`, your Phase 1 solution should complete in 3 seconds, and your Phase 2 solution should complete in 7 seconds. In `headlessDriver.java`, the solution will be strictly limited to 5 seconds for Phase 1 and 10 seconds for Phase 2. If your solution does not complete in time, you may want to investigate further, since this is how we will be grading your submission. (We will be running your submissions on a high performance processor: if it can finish in 10 secs total on your computer, it will on ours.)

9 What to submit

Zip package student and submit it on CMS. **Be sure that you have not changed the interface to methods `rescue()` and `returnToEarth()`**, because those changes would not be registered in your CMS submission, would break the autograder, and would result in a very low score.
You may add helper methods or additional classes to package student, but make sure you specify everything well.
It is crucial that you submit a zip that contains package student: nothing else, and nothing less.
Also ensure that all print statements that you use for debugging purposes use `outPrintln` and `errPrintln`. These are only printed if `shouldPrint` is true. When we grade, we set `shouldPrint` to false. Any other print statements that you have will result in point deductions.

Figure 3: Neil Armstrong and Apollo 11 (Somewhere in Space, 1969, colorized 2017)

10 References and Acknowledgements

We are grateful to previous members of the CS2110 course staff for their assistance, including but not limited to Michael Patashnik, Alex Fusco, Eric Perdew and Nate Foster.

Good luck and have some fun!