CS 2110 Assignment 8: Planet X

Your CS2110 TAs

Due date: See the CMS

1 Introduction

We have received a distress call from a spaceship (the USS Java) that just crash-landed on an unknown planet somewhere in space, on Planet X. Fleet Admirals Birrell, Gries, and Sampson have dispatched you, as Captain of our fastest spaceship, to rendezvous with the ship and provide 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 stages, each of which involves writing a method in Java. Watch the beginning of the lecture on spanning trees on VideoNote to see a short demo of A8.

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 Assignment Overview

"Ipsa scientia potestas est."

—Sir Francis Bacon

This assignment has two stages: the rescue stage and the return stage. We present a high-level view of the two stages. Low-level details of the implementation appear in the pinned Piazza note A8 FAQs.

3.1 Rescue Stage

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

The USS Java is on Planet X and is emitting a distress signal. You are on Earth, and you have been commanded to search space to rescue the USS Java. It's a rather random search, because you have no idea

where Planet X is. You just have to traverse space until you find it. From each planet you can travel only to certain other planets. (The planets are nodes of an undirected graph.)

Your ship must travel from planet to planet until it reaches Planet X, the site of the distressed ship. Your objective is to get to planet X in as short a distance as possible..

You can use the strength of the distress signal to help. The closer a planet is to Planet X, the stronger the signal. So, try flying first to neighboring planets where the signal is strongest. But that doesn't always help. The path the ship must fly to get to Planet X from that geographically closer planet may actually be longer.

3.2 Return Stage

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

Upon successful rescue of the USS Java, its captain, who has perhaps 1 year of coding experience, has discovered that some planets contain valuable SpaceGems™[1]. He wants *you* to plan a strategy to pick up as many gems as possible while still returning to Earth before running out of fuel. Here's your chance to show your mettle! There is no known best algorithm for that! How will you figure out a path around the planets to pick up as many gems as possible. Sure, you can get back home by traversing the shortest path to Earth. But how many gems will you pick up doing that?

4 The GUI

Fig. 1 on the next page is a picture of the GUI associated with the program. The left side is a 2D map of the planets, projected into whatever sized window the GUI is running in. The coordinates of the map are to scale: the actual distance between two planets is proportional to their apparent distance on the map. However, the *actual bounds of the graph of planets is square*, while your displayed map is most likely rectangular. This will result in a certain direction appearing to be more compressed than it actually is. You can switch to a new, randomly generated map by going to File→New Map and entering a number (seed) of your choice. If you leave the seed blank, the simulation will use a randomly chosen seed. To test your solution on a small map, try seed -2466017765573610414 (don't forget the negation sign). To see why you might want to optimize your rescue stage, try seed -1. To see a case where (most) optimizations don't work well, try running your algorithm on the graph produced with seed 16. We may put other interesting seeds on the A8 FAQs note as they are discovered.

The panel on the right-hand side of the GUI displays information of possible interest. This information changes slightly depending on which stage you are currently in. In Fig. 1, the rescue stage has completed; it took distance 5046 to reach Planet X. The return stage is now underway. The spaceship left Planet X and is on its way to planet Carletonmoore. The right-hand Panel gives the previous planet, Planet X, and the current score.

The ship has a limited amount of fuel, and the right panel shows that it has only enough fuel to go the remaining distance, in this case, 6475. If the ship tries to travel farther than that, it fails and never reaches the Earth. The score will be 0.

Click a planet and its name and the number of gems on it appear in the right-hand panel. The number of gems will update in real time. Planet Careltonmoore has 558 gems on it. In returning back to Earth, it's good to pick up as many gems as possible. So, the ship will want to visit as many planets as possible while still getting back to Earth before it runs out of fuel.

Slider "Simulation speed" can be used to make the simulation run faster. Use the slider to fast-forward past parts of the simulation in which you are not interested.

The "zoom" slider allows you to visually zoom in on the map, for situations where too many planets are in a small area, making them difficult to distinguish.

If you check box "Camera Follows Ship", the view of the map will keep the ship centered as the ship moves from planet to planet. If you do not check the box, you can manually move the view via the arrow keys, WASD, or even the vi-keys (hjkl) if you are so inclined.

[1] We do not endorse any product or institution that happens to carry this name

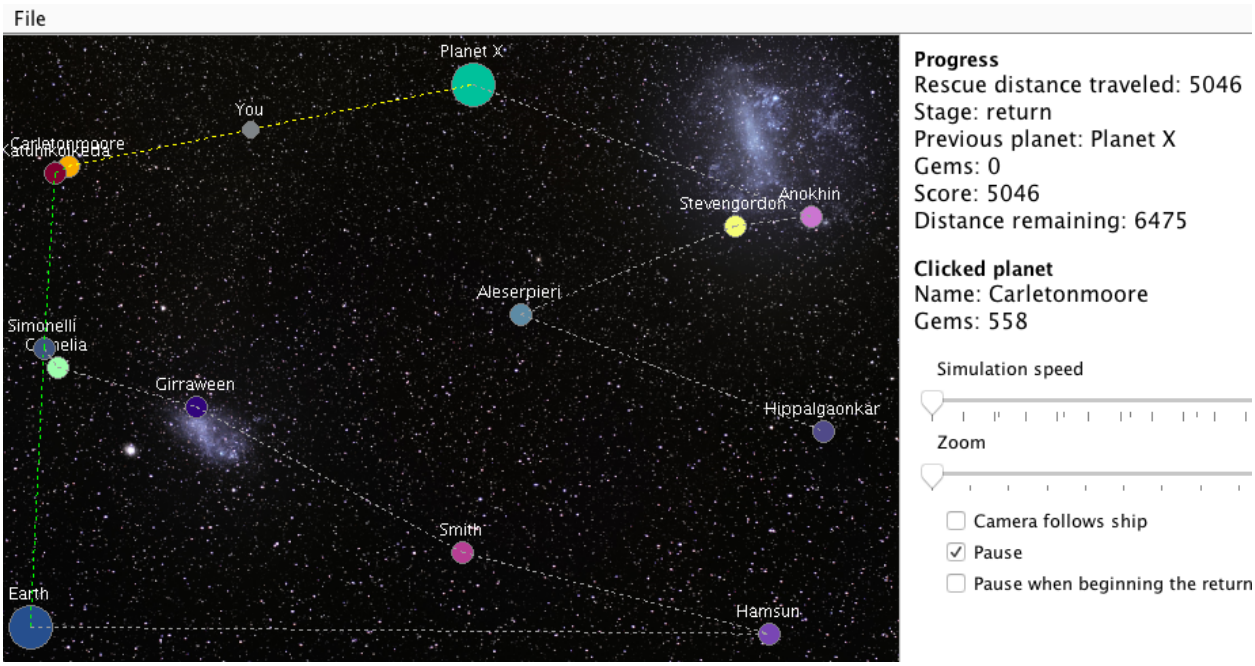


Figure 1: Example GUI screen

You can see that we Paused the simulation. Uncheck the box to have the simulation proceed. You can also have the simulation pause automatically when Planet X is reached at the end of the rescue stage.

Menu "File" in the top left has options for controlling the simulation. Choose "start" to start the simulation,"Reset" to reset the current map, and "New map" to generate a new map. A simulation can be run only once, starting from the paused state. To rerun it, use menu item File → Reset.

As the ship moves along, edges change color. Edges are initially gray; on the first traversal, they turn green; on the second traversal, yellow; and on subsequent traversals, red.

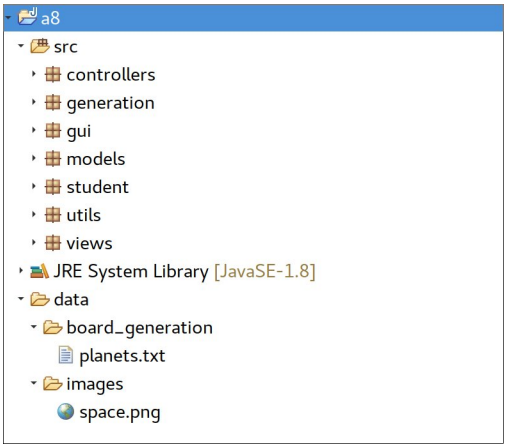
5 Setting up Eclipse

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

—Laozi

Download and unzip the release-code file. Start a new Eclipse project. To install the Java code, drag the directories in directory src over "src" in the Eclipse project. To install the data, drag directory data over the project itself. When you do this, when asked, tell it to copy rather than to link. You should end up with src and data on the same level within the project.

When adding the release code to an Eclipse project, be sure to **copy the files** and not link them. The end result should look like the image on the right. Choose whatever you want for project name itself (a8 in this example).



6 Running the Program

Method `main` needed to run the program is in class `PlanetX` of package `controllers`. The specification for `main` gives more details about possible arguments, though you may not need any of them. By default, the program runs on a single map with a random seed. It is possible to run in *headless mode*, in which you do not have a GUI. This allows you to estimate how much "real time" your simulation takes. We'll discuss this later in the Piazza A8 FAQs note.

7 Grading

> *"...programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil..."*
>
> —*Donald Knuth*

The vast majority of points on this assignment will come from a correct solution that always finds Planet X and returns to Earth safely (within the given distance limit), so your priority is 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, so think about ways to optimize your solution, to pick up as many gems as possible.

The use of Java Reflection mechanisms aside from method `getClass()` and field `class` in method `equals()` is **strictly forbidden** and **will result in significant penalties**.

8 What to Submit

Zip package student and submit it on CMS. **Be sure that you have not changed the interfaces to methods `rescue()` and `returnToEarth()`**, because doing so would make your solution fail to compile, resulting in a very poor score.

You may add helper methods or additional classes to package student, but specify everything well.

It is crucial that you submit a zip archive that contains package student: nothing else, and nothing less. As a check, unzipping the archive should result in a directory named `student`, which solely contains your `.java` files.

Finally, your submission should be well-written. Don't leave any traces of debugging code, such as print statements.

9 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 for *your* benefit, write precise and complete Javadoc specs for them.) 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 different.

We suggest *first* getting a solution that is simple and works. This will give 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.

10 Acknowledgements

"If I have seen further, it is by standing on the shoulders of giants."

—Sir Isaac Newton

We are grateful to past and previous members of the CS2110 course staff for their assistance and contributions, including but not limited to: Michael Patashnik, Alex Fusco, Eric Perdew, Nate Foster, Joe Antonakakis, Matthew Weidman, Elizabeth VanDenburgh, Jason Liu, Ramya Babu, Adam Gleisner, Lucas Png, and Devin Lehmacher.

Jason Liu has been the main guy in hammering this assignment into shape! Thanks, Jason!

Good luck and have some fun!