CS 2110: Object-Oriented Programming and Data Structures

Assignment 7. The New York Sewer System

November 5, 2018

1 Overview

Someone dropped a famous, expensive ring into a sewer in New York City. It drifted off somewhere and has to be found. It's a special, magical ring and can always be seen wearing special glasses,. Diver Max (it's Maxwell or Maxine, we can't tell which) is going into the sewer system with the special glasses to find the ring. And find it Max does! Max takes off the special glasses and heads to the exit. But now, with the special glasses off, Max sees coins all over the sewer system, and on the way to the exit, Max picks up as many coins as possible.

You are Max. You will write a method to find the ring and another method to get out within a certain number of steps, picking up as many coins as you can.

While this may seem like a rather flippant story, real people spend their time in sewers, keeping them in working order. It's dangerous, smelly, unhealthy, awful work, but someone has to do it. Watch these two videos about sewer divers in London and Mumbai to see what we mean:

London Sewage Worker https://www.youtube.com/watch?v=hIOVUA4Lqos
The Sewer Men of Mumbai https://www.youtube.com/watch?v=tWBCsn6pQWc

We have the utmost respect for these sewer workers, and we thank them profusely for the job they do.

Paris used to have tourist tours of their sewer system, and Prof Bracy actually took such a tour. Look at this website:

https://europeforvisitors.com/paris/articles/paris-sewers-museum.htm

2 Collaboration Policy and Academic Integrity

You may complete this assignment with one other person. If you plan to work with a partner, as soon as possible —at least by the day before you submit the assignment— login to CMS and form a group. 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, a consultant. Do not wait. A little in-person help can do wonders to get you unstuck.

3 Find-the-ring Phase

On the way to the ring (see Fig. 1 on the next page), the layout of the sewer system is unknown. Max knows only the status of the place on which Max is standing and the immediately neighboring ones (and perhaps others that Max remembers). The goal is to make it to the ring in as few steps as possible.

This does not have to be a completely blind search. Max can see the ring shining in the distance. The closer Max gets, the stronger the light from the ring. When standing on the ring, the distance is 0. Wherever Max is, Max will be able to see which neighbor is closer or farther from the ring.

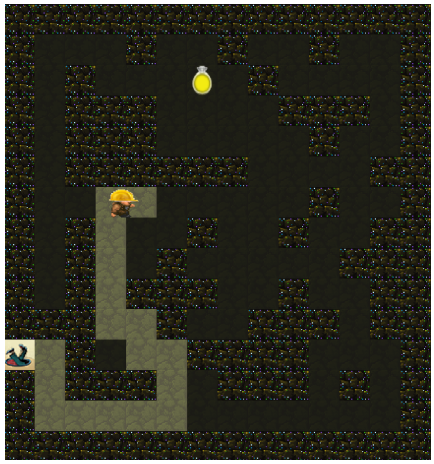


Figure 1: Find-the-ring phase

You will write the solution to this phase in method `findRing(...)` in class `DiverMax` within package `student`. We suggest a depth-first walk as a first implementation that will always find the ring but may not receive a large bonus multiplier. For this first depth-first walk, *study the video on a dfs walk found in link DFS/BFS in the JavaHyperText*! Here is is: http://www.cs.cornell.edu/courses/JavaAndDS/dfs/dfs01.html.

The first phase is discussed in Piazza note @821, which, of course, is reachable from pinned Piazza note @824, Assignment A7. More info also appears in the specification of method `findRing(...)`.

4 Get-out Phase

After picking up the ring and taking the glasses off, diver Max sees piles of coins all over the place. Luckily, underneath the ring is a map that reveals the full sewer system. Max rushes around, picking up as many coins as possible. But because the sewer system is a rather unhealthy environment, Max must get to the exit within a prescribed number of steps. See Fig. 2, which shows also the pane on the left of the GUI, giving information about the sewer system.

The goal of the get-out phase is to get to the exit within a prescribed number of steps. In Fig. 2, the exit/entrance is in the lower right-hand corner of the sewer system —you see the diver with his feet still showing. Max's score will be the product of these two quantities:

1. The value of the coins that Max picks up during the get-out phase.
2. The score multiplier from the find-the-ring phase.

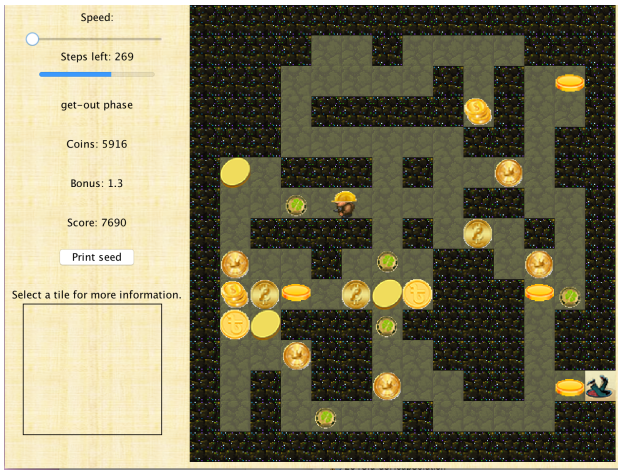


Figure 2: Collecting coins during the get-out phase

You write your solution to this part in function `getOut()` in class `DiverMax` in package `student`.

A good starting point is to write an implementation that takes the shortest path to the exit, which is guaranteed to succeed. After that, consider how to traverse the sewer system to pick up more coins to optimize your score using more advanced techniques. However, the most important part is always that your solution successfully gets to the exit of the sewer system in the prescribed number of steps. If you improve on your solution, make sure that it never fails to get out in time.

The get-out phase is discussed in Piazza note @825, which, of course, is reachable from pinned Piazza note @824, Assignment A7. More info also appears in the specification of method `getOut(...)`.

5 What you can do

This is your chance to figure out for yourself a good way to gather coins. No algorithm works best on all sewer systems (graphs). 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.

6 Creating the Project in Eclipse

Download the zip file from the course website. The zip file is an Eclipse project, which you can most easily import into Eclipse using menu item Open Projects From File System. Here are the steps:

1. Use menu item File $--> $ Open Project From File System

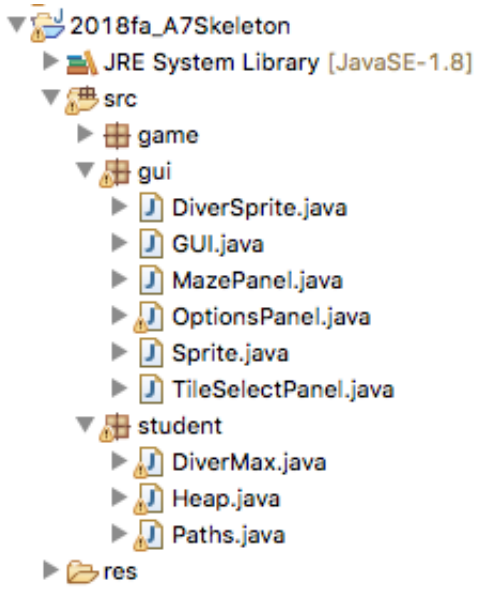
2. In the window that opens: select ”archive”, navigate to the downloaded zip file, select it, and click Open.

3. If you get a choice of Folders to select, select only the zip file with the ”Import as” option set to ”Eclipse Project”.

4. Want to change the project name? Use the refactoring tool.

You can, if you want, copy parts individually into a project. But that will require putting JUnit5 on the build path, etc. The project should look like the diagram on the right (your JRE System Libraries may be different).

The release code contains our solution to `Heap.java`. You can get our solution to A6, `Paths.java`, from the pinned Piazza post on A6 after the deadline for submission, which is 10 November. You may use your own implementations of these two classes —but only if you know they are correct.



7 Running the program

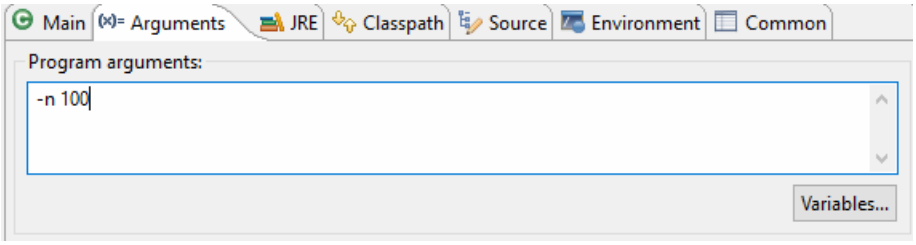
The program can be run from two classes. Running from `GameState.java` runs the program in headless mode (without a GUI); running it from `GUI.java` runs it with an accompanying display, which may be helpful for debugging. By default, each of these runs a single map on a random seed. If you run the program before any solution code is written, you will see diver Max stand still and an error message pop up telling you that your solution to find returned at the wrong location.

Two optional flags can be used to run the program in different ways.

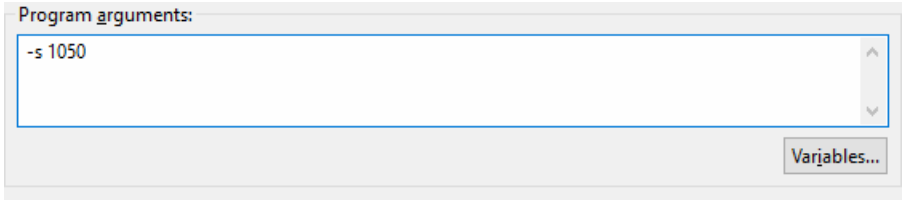
1. `-n <count>` runs the program *count* times. This option is available only in headless mode; it is ignored if run with the GUI. Output will still be written to the console for each map so you know how well you did, and an average score will be provided at the end. This is helpful for running your solution many times and comparing different solutions on a large number of maps.

2. `-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. It is helpful for debugging. This can be used both with the GUI and in headless mode.

To set these arguments, in eclipse click Run → Configurations, click on tab Arguments, and enter the arguments under Program Arguments. For instance, in order to run the program 100 times in headless mode, write:



To run the program once with a seed of 1050, write:



When running in headless mode, you may also combine these flags. If you specify both a seed and a number of times to run, the first run will use the provided seed and subsequent runs will use additional seeds generated by the one provided. This may be useful if you want to compare two solutions on the same sequence of random maps.

8 The GUI

When running your program (except in headless mode), you can watch diver Max making moves in a GUI. When the GUI is running, each call to `moveTo()` blocks until the corresponding move completes on the GUI —that is, a call to `moveTo()` will not return and consequently your code will not continue running until the corresponding animation on the GUI has completed. For that reason, running in headless mode will generally finish in less time than running it with the GUI.

You can use the slider on the left side of the GUI to increase or decrease Max's speed. Increasing the speed makes the animation finish faster. Decreasing the speed might be useful for debugging purposes and to get a better understanding of what exactly your solution is doing. Also, the number of steps remaining during the get-out phase (both as a number and a percentage) is displayed. A Print Seed button prints the seed to the console to easily copy and paste into the program arguments in order to retry your solution on a particularly difficult sewer system.

You also see the bonus multiplier and the number of coins collected, followed by the final score computed as the product of these. The multiplier begins at 1.3 and slowly decreases as Max takes more and more steps during the find-ring phase (after which it is fixed), while the number of coins increases as they are collected during the get-out phase.

Finally, click any square in the map to see more detailed information about it on the left, including its row and column, the type of tile, the tile's unique ID number, and the value of the coins on that square.

9 Grading

85 points on this assignment come from a correct solution that always finds the ring and always gets out within the prescribed number of steps, so your priority should be to make sure that your code always does this successfully. A submission that is always successful will receive a minimum of 85/100. To receive a higher grade, your solution must also get a reasonably high score (achieved by optimizing the bonus multiplier in the explore phase and collecting as many coins as possible in the get-out phase), so spend

some time thinking about ways to optimize your solution. There may also be a few bonus points available for exceptional solutions.

The amount of time your code takes to decide its next move does not factor into the number of steps taken and consequently does not effect your score. However, we cannot wait for your code forever, so we must impose a timeout when grading. When run in headless mode, your code should take no longer than roughly 10-12 seconds to complete on any single sewer system (map). Solutions that take significantly longer on a map will be treated as if they did not successfully complete and will receive a low grade on that map.

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

10 Instructions for submission

Zip package student and submit it on the CMS. Before submitting, make sure that you have not changed the signatures of methods `findRing(...)` or `getOut(...)` in class `DiverMax` and that these methods work as intended. You may add helper methods and additional classes to package student, but make sure to specify everything well. In the end, make sure that if we replace our student package in our solution with your student package, your solution will still work as intended.

It is important that the zip file you submit contains exactly package student and nothing else. To do this, select directory student and do what you have to do to zip it.

Five (5) points will be deducted if your submission contains println statements.