INTRODUCTION TO PROGRAMMING
DM550, DM857, DS801 (Fall 2020)

Exam project: Part III — Deadline: 23h59 on Friday, December 18th, 2020

Overview

In this part, your task is to develop the classes in the middle layer of the card game: classes `Colony` and `Graph`.

Class `Colony`

This class extends class `Node` with three new methods, which manage the colony's sugar stock. A colony does not store any transportable sugar (the sugar managed by the `Node` class). This class should provide the following methods:

- a constructor with no arguments, creating a new colony with no stock of sugar;
- a method `void topUp(int sugar)` that increases this colony's stock of sugar by the given amount;
- a method `void consume()` that decreases this colony's stock of sugar by one unit;
- a method `boolean hasStock()` that indicates whether the stock of sugar in this colony is non-empty.

Additionally, your implementation can override any methods from class `Node`, if necessary.

Class `Graph`

Instances of this class represent a graph on which the ants can move throughout the simulation. This class should provide the following methods:

- a constructor taking a width, a depth, an array of `Colony`s, the probability of a node initially containing sugar and the average amount of sugar in such a node, creating a new grid graph with the specified dimensions (see below);
- a constructor taking a filename, an array of `Colony`s, the probability of a node initially containing sugar and the average amount of sugar in such a node, creating a new graph with the nodes and edges provided in the file given (see below);
- a method `int pheromoneLevel(Node source, Node target)` that returns the pheromone level in the edge connecting `source` and `target` in this graph;
- a method `void raisePheromones(Node source, Node target, int amount)` that increases the pheromone level in the edge connecting `source` and `target` in this graph by `amount`;
- a method `Node[] adjacentTo(Node node)` that returns an array with all the `Node`s that are connected to the given node by an edge in this graph;
- a method `void tick()` that decreases the level of pheromones in this graph by a unit (simulating a tick), and possibly spawns sugar in a random node (see below).

Constructors. The first constructor creates a grid graph, where the nodes are spaced at regular intervals in a two-dimensional grid, and each node is connected to the nodes above, below, to the left of, and to the right of it (except for the nodes on the border and corners, which do not have nodes in some of those positions). The location of the colonies is random.

The second constructor creates a graph from a file containing (i) the total number of nodes in the first line, (ii) the locations of the colonies in the second line, separated by spaces, and (iii) edges in the remaining lines, one per line, represented as two integers, separated by spaces. The nodes are assumed to be numbered from 1 to the total number of nodes. If there is an error reading from the file, you should print an error message and terminate the program – this method should not throw any exception. If no errors occur, you can assume that the file is well-formed (in particular, that there are no duplicate edges).[4]

[4]A well-formed example file is available for download with the project.

The graph is meant to be undirected. This means that: if there is an edge between n_1 and n_2, then this is also an edge between n_2 and n_1. In particular, `pheromoneLevel(n1,n2)` and `pheromoneLevel(n2,n1)` should return the same result. Your implementation must satisfy this requirement.

After creating the graph (whether it is a grid or read from a file), you should go through all nodes and decide whether they should have sugar or not. For each node that is not a colony, call method `RandomUtils.coinFlip()` with the given probability to decide whether it gets sugar. In the affirmative case, method `RandomUtils.randomPoisson()` generates a random number with the expected value given as parameter.

Important. Do not use any other classes to generate random numbers, as this may cause your program to malfunction.

Method `tick()`. One of the effects of this method is possibly spawning a fresh stock of sugar at a random node. This happens if method `RandomUtils.coinFlip()` returns `true` when called with the probability given in the constructor. In the affirmative case, a random node that is not a colony should be chosen, and its sugar stock should be increased by the result of calling `RandomUtils.randomPoisson()` with the same parameter as in the constructor.

Expected results

You must hand in a zip file containing no subdirectories and the following files.

1. Java source files `Colony.java` and `Graph.java` implementing the contracts described above. These classes are either clients of or providers to the classes developed in the previous parts of the project. They should be compatible both with the provided implementations of them, and with your own implementations.

2. A pdf file `report.pdf` describing the design of your classes – your choice of attributes, whether you provide any additional getters and setters, which universal methods you override, and any other design choices that you think are important to document. One of the members of the group must be clearly identified on the title page as the coordinator for this part of the project; the coordinator cannot be the same as in the first or second part (unless the group does not have enough elements). The source code for the developed classes should be included as appendix.

If you do not follow these rules, your project may be rejected and not graded. The report will be the basis for the evaluation.