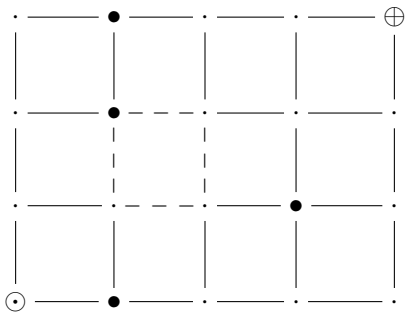
Object Oriented Programming Project 2017/18

MEEC – IST

1 Problem

Consider a grid $n \times m$. We want to find the best path, that is, the path with lowest cost between an initial point, with coordinates (x_i, y_i), and a final point, with coordinates (x_f, y_f). Each path has an associated cost that is determined by the number of edges traversed. In general, the cost of each edge is 1, but there may be special areas where the cost is higher. There are n_{obst} obstacles at some points on the grid through which one can not proceed.

For instance, consider the following figure which represents a grid 5×4:



In this figure, the initial point, marked with $\odot$, has coordinates $(1, 1)$, the end point, marked with $\oplus$, has coordinates $(5, 4)$, the obstacles are marked with $\bullet$, and the edges of the special cost areas, indicated by dashed, have cost 4. In this case, the best path is $(1, 1), (1, 2), (2, 2), (3, 2), (3, 1), (4, 1), (5, 1), (5, 2), (5, 3), (5, 4)$ with cost 12. There are shorter paths but whose cost is higher. For example, the path $(1, 1), (1, 2), (2, 2), (3, 2), (3, 3), (4, 3), (5, 3), (5, 4)$ is shorter but its cost is 13.

2 Approach

The goal of this project is to program in Java a solution to the problem presented above using evolutive programming modeled and implemented with objects.

The idea is to generate at instant zero a population of ν individuals, all placed at the initial point, and make it evolve until the final instant τ. Each individual z is associated with a comfort

$$\varphi(z) = \left(1 - \frac{cost(z) - length(z) + 2}{(cmax - 1) \times length(z) + 3}\right)^k \left(1 - \frac{dist(z, (x_f, y_f))}{n + m + 1}\right)^k$$

where:

- *cmax* is the maximum cost of an edge in the grid;
- *cost*(z) is the cost of the path of the individual z;
- *length*(z) is the length of that path – the number of edges traversed;
- *dist*$(z, (x_f, y_f))$ is the distance between the last point of the path and the final point – the fewer number of edges needed to be traversed until the final point is reached;
- k is an input parameter of comfort sensitivity to small variations.

Each individual z evolve according to its comfort, by the following random mechanisms:

- **Death**, exponential variable with mean value $(1 - \log(1 - \varphi(z)))\mu$.
- **Reproduction**, exponential variable with mean value $(1 - \log(\varphi(z)))\rho$ between events. From reproduction, a new individual is born with. The path of the new individual is a prefix of the parent path. The length of that prefix is determined by the parent's comfort, always having 90% of the parent's path and a fraction $\varphi(z)$ of the remaining 10%.
- **Move** (usually called mutation in evolutive programming), exponential variable with mean value $(1 - \log(\varphi(z)))\delta$ between events. The move can occur equiprobably to an adjacent position on the grid not occupied by an obstacle. If the individual moves to a position that is already on its path the corresponding cycle should be eliminated.

The population evolves in function of the individual evolution of its elements and also by the occurrence of **epidemics**. Whenever the number of individuals exceed a maximum ν_{max}, an epidemic occurs. To the epidemic will always survive the five individuals with greater comfort. For each of the remaining, the survival probability is $\varphi(z)$.

The evolution of the population is ruled by discrete stochastic simulation, that is, based on a pending event container.

3 Parameters

The program should receive the following data:

- the dimension n and m of the grid;
- the initial point (x_i, y_i) coordinates and final point (x_f, y_f) coordinates;
- a list of tuples (coordinate, coordinate, cost)

  $$\{((x_1, y_1), (x'_1, y'_1), c_1), \ldots, ((x_k, y_k), (x'_k, y'_k), c_k)\}$$

  to specify the rectangular special cost areas;
- the number of obstabcles n_{obst} and their coordinates (x_i, y_i), for $i = 1, \ldots, n_{obst}$;
- the final instant $\tau(> 0)$ of the evolution;
- the parameters $k, \nu, \nu_{max}, \mu, \delta, \rho$.

3.1 Input file format

The file that describes the input parameters of the simulation is an XML file. The simulation is an element `simulation` whose attributes indicate the final instant (`finalinst`), the initial population (`initpop`) and maximum population (`maxpop`) of the simulation, as well as the sensitivity of the comfort to variations (`comfortsens`). The element `simulation` contains six elements:

- The empty element `grid` whose attributes describe the grid dimension (`colsnb` and `rowsnb`).
- The empty element `initialpoint`, whose attributes describe the initial point coordinates (`xinitial` and `yinitial`).
- The empty element `finalpoint`, whose attributes describe the final point coordinates (`xfinal` and `yfinal`).
- The empty element `specialcostzones` which contains a list of elements `zone` whose attributes indicate its position in the grid (`xinitial`, `yinitial`, `xfinal`, `yfinal`) and whose content indicates the cost.
- The element `obstacles`, whose the only attribute describe the number of obstacles (`num`), which contains a list of empty elements `obstacle` whose attributes indicate its position in the grid (`xpos`, `ypos`).
- The element `events`, which contains three new empty elements (`death`, `reproduction` and `move`) whose attributes (`param`) describe the parameters related to the death, reproduction and move events.

3.2 Example

Consider the example described in Section 1:

- grid dimension: $n = 5$ and $m = 4$;
- initial point: $(x_i, y_i) = (1, 1)$;
- final point: $(x_f, y_f) = (5, 4)$;
- special cost zone: $(x_1, y_1) = (2, 2)$, $(x_1', y_1') = (3, 3)$ and $c_1 = 4$;
- obstacles: $n_{obst} = 4$ and coordinates $(2, 1), (2, 3), (2, 4), (4, 2)$;
- evolution final instant: $\tau = 100$;
- initial population: $\nu = 10$;
- maximum population: $\nu_{max} = 100$;
- comfort sensitivity: $k = 3$;
- parameters related to the events: $\mu = 10, \delta = 1, \rho = 1$.

In the following we have the same parameters in XML where `simulation.dtd` must be provided and placed in the same folder as the `.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE simulation SYSTEM "simulation.dtd">
<simulation finalinst="100" initpop="10" maxpop="500" comfortsens="3">
  <grid colsnb="5" rowsnb="4"/>
  <initialpoint xinitial="1" yinitial="1"/>
  <finalpoint xfinal="5" yfinal="4"/>
  <specialcostzones num="1">
    <zone xinitial="2" yinitial="2" xfinal="3" yfinal="3" >4</zone>
  </specialcostzones>
  <obstacles num="4">
    <obstacle xpos="2" ypos="1"/>
    <obstacle xpos="2" ypos="3"/>
    <obstacle xpos="2" ypos="4"/>
    <obstacle xpos="4" ypos="2"/>
  </obstacles>
  <events>
    <death param="10"/>
    <reproduction param="1"/>
    <move param="1"/>
  </events>
</simulation>
```

4 Results

The program should print to the terminal at the end of the simulation the path of the *best fit individual* over all the simulation. By the best fit individual we mean:

- if there is an individual which reached the final point, the individual z with lower cost, independently from the individual z being or not alive at the end of the simulation;
- if none of the individuals reached the final point, the path of the individual z with greater comfort, independently from the individual z being or not alive at the end of the simulation.

The printed output should be in the following format:

$$\texttt{Path of the best fit individual = } \{(x_1, y_1), \ldots, (x_j, y_j)\}$$

During the simulation the program should also print to the terminal the result of observations of the population, realized from $\tau/20$ by $\tau/20$ time units. Each observation should include the present instant (*instant*), the number of events already realized (*events*), the population size (*size*), the path of the best individual and its cost/comfort until the actual instant (*cost* if the final point has been hit and *comfort* otherwise), according to the following format:

`Observation` *`number`*`:`

`Present instant:`	*`instant`*
`Number of realized events:`	*`events`*
`Population size:`	*`size`*
`Final point has been hit:`	`yes/no`
`Path of the best fit individual:`	$\{(x_1, y_1), \ldots, (x_j, y_j)\}$
`Cost/Comfort:`	*`cost/comfort`*

Any other printing to the terminal, or a print of this content out of this format, incurs in a penalty in the project grade.

5 Simulation

The simulator should execute the following steps:

1. Read the file that describes the input parameters of the simulation, and save/create the needed values/objects. The file with the input data is an XML file, whose format is described in Section 3.1, and it must be validated by an appropriate DTD.
2. Run the simulation cycle until: (i) the evolution final instant is reached, or (ii) there are no more events to simulate. During the simulation must be printed to the terminal the population observations described in Section 4.
3. At the end of the simulation must be printed to the terminal the information requested in Section 4.

6 Evaluation

The assessment will be based on the following 7-point scale:

1. **(1.5 point)**: UML. The UML will be evaluated, in a 1-point scale as: 0–very bad, 0.4–bad, 0.8–average, 1.2–good and 1.5–excellent.
2. **(5.5 points)**: A solution that provides an extensible and reusable framework. The implementation of the requested features in Java are also an important evaluation criteria and the following discounts, on a 5.5-point scale, are pre-established:
   (a) **(-2 points):** OOP ingredients are not used or they are used incorrectly; this includes polymorphism, open-close principle, etc.
   (b) **(-1 points):** Java features are handled incorrectly; this includes incorrect manipulation of methods from `Object`, `Collection`, etc.
   (c) **(-0.35 points):** Prints outside the format requested in Section 4.
   (d) **(-0.35 points):** A non-executable jar file, or a jar file without sources or with sources out of date. Problems in extracting/building a jar file, as well as compiling/running the executable in Java, both from the command line. Problems with Java versions; the Java executable should run properly in the laboratory PCs.

Finally, in a 7-point scale:

1. Files submitted outside of the required format will have a penalty of 5% over the respective grade.

2. Projects submitted after the established date will have the following penalty: for each day of delay there will be a penalty of 2^{n-1} points of the grade, where n is the number of days in delay. That is, reports submitted up to 1 day late will be penalized in $2^0 = 1$ points, incurring in a penalty of 0.35 points of the final grade; reports submitted up to 2 days late will be penalized in $2^1 = 2$ points, incurring in a penalty of 0.7 points of the final grade; and so on. Per day of delay we mean cycles of 24h from the day and hour specified for submission.

7 Deadlines and material for submission

The **deadline for submitting the project is May 11, before 18:00**. The submission is done via fenix, so ensure that you are registered in a project group.

The following files must be submitted:

1. An UML specification including classes and packages (as detailed as possible), in `.pdf` or `.jpg` format. Place the UML files inside a folder named UML.

2. An executable `.jar` (with the respective source files `.java`, compiled classes `.class`, and `MANIFEST.MF` correctly organized into directories).

3. Five examples of input file parameters (`test_i`, with `i`$= 1, \ldots, 5$) used to test the program. Place these examples inside a folder named TESTS.

4. Documentation (generated by the javadoc tool) of the application. Place the documentation inside a folder named JDOC.

5. A final report (up to 10 pages, in .pdf format) containing information that complements the documentation generated by the javadoc tool. Place the final report inside a folder named DOCS.

6. A self assessment form (in `.pdf` format) that will be made available in due time in the course webpage. Place the self assessment form inside the folder named DOCS (the same folder as the final report).

The UML folder, executable (the `.jar` file with the source files, besides the compiled files and `MANIFEST.MF`), the JDOC folder and the DOCS folder, should be **submitted via fenix in a single `.zip` file**.

The final discussion will be held from May 21 to June 1. The distribution of the groups for final discussion will be available in due time. All group members must be present during the discussion. **The final grade of the project will depend on this discussion, and it will be not necessarily the same for all group members.**