Instituto Superior Técnico

Object Oriented Programming

Evolutive Path Simulator

Author:
Group 16
Miguel Pinho 80826
Miguel Malaca 81702
João Mak Duarte 81660

Professor:
Alexandra Carvalho

May 11, 2018

1 Introduction

The project presented was to develop a specific type of simulator in a previously determined setting. In this case it was a stochastic simulation that consisted in determining the best path of a specimen from a starting point to the final one, on a static map with different cost edges and obstacles. As in any kind of simulation of this type, a population is submitted to events that in some way shape their path, in this case they were deaths, reproductions and mutations.

The way the several problems were tackled and decisions were made relatively to the implementation of this simulator is described in the following chapters, were there's also a section of overall review of the work done with examples.

2 Problem Approach

2.1 Choice of packages and general organization

The mission statement when developing this project was that it could be as generalized as possible, meaning that it could be reused not only for this type of simulation, but also in any other stochastic simulation.

With that guiding line, the package simulation was created. Besides that, all the classes that can be generalized belong to a different package than the ones that inherit from those and are specific to this problem. Another perk of this organization is that the generalized packages can be used independently, depending on the problem at hand.

2.2 The class Path and its relation to the grid

In this problem of evolutive programming the simulating objects are the paths. For these specific elements it was created the class Path. In this sense, the structure in which the paths act upon is the map. To connect the Path with the Map the original idea was to create a static field in the Path. After further analysis, a conclusion was reached that this made the program less extensible.So, in order to improve, a reference was put in the Path to the Map. This reference is needed only when the first paths are created, because the map is the same for every specimen in the same simulation.

The map has a representation of the grid that contains all the edges of every point, implemented in a class Edge. In this way, the path also keeps a list of edges that represent the route that individual has made. In this case the edges that the Path and the Map have are the same. This happens for every edge except for the initial point of the Path that has to have a cost 0 and thus, doesn't belong to the grid. In this way, just need one instance of the edges was needed to represent both structures.

2.3 The Events and their Pending Event Container

The Event is implemented as an abstract class that later is inherited by the specific events of this simulation, so it can be later reused. As for the Pending Event Container (PEC), it was chosen a priority queue to implement it. Originally a tree set was used, derived from the necessity of having a sorted list of events by their own trigger time but due to it's limitations, such as it's inability to handle duplicate events, although it was unlikely to happen, it was discarded, even though the problem could be solved by offering an alternative comparison than the event's trigger time. A comparable was implemented to override the queue's original order. Since that for the major part of the simulation the PEC's main operations are insertions and removals in a sorted list, the priority queue is the better fit for it's structure type.

On the other hand, for the situation in which a epidemic occurs, the queue offers a method for transposing it to an array, which allows to more easily remove any invalid events (events that occur to individuals that died as a result of the epidemic) which in any case it's a process independent to the sorting, validating even further this choice for structure type.

With all this said, it's possible to imagine that these two classes are extensible to other types of simulation, as a deterministic one.

2.3.1 Event and Simulation times

The Simulation class (and the PopulationSimulation by inheritance) provide an API setting the total simulation time and for running the simulation step by step. In each step interval, the events are triggered in temporal order only up to the end of that interval, the rest being kept in wait. However, care is also taken to not create events that would happen after the simulation, so that the PEC is empty in the end of the simulation.

For the 20 requested observations, the simulation is run with $\frac{\tau}{20}$ interval steps (for a total τ time), with the output being generated in between runs.

It can happen that all the specimens die before the end of the simulation, so the PEC will be empty. In this case, no more steps (and observations) of the simulation are executed, and the final instant is the end of that step.

2.4 The Specimen and the Population

2.4.1 The Classes

For the population of specimens a linked list was used since there was no need for permanent sorting and it had to be a structure with simple access to it's components. The class Population is, in this context, the state of the evolution at a given moment. It contains the specimens, it's best organism so far and it acts upon the initialization, epidemics and dead specimen removal. On the other hand, the class Specimen holds the state and behavior of a single individual. It is a node of the population list.

Both classes were defined such that they would be generic so that they could be made to work with any form of class that complies with certain specifications, more concretely, the interface Organism. In this project, they are used for the class Path.

Their relation isn't of aggregation, since the specimen changes the state of the population (epidemics, reproduction and death).

2.4.2 Epidemic

The epidemic function of the simulation was implemented by the use of an exception. When in a reproduction event the current population exceeds the given maximum, it throws an exception that triggers the epidemic event.

Although the specimens needed not to be sorted in a general case, for the epidemic process, a comparator was implemented to use the sort method of the list. This sorted the list by comfort of each individual, so that the process of identifying the five best individuals (highest comfort) was made simpler. After this selection, a random generator creates a number between zero and one. If it's larger than the specimen's comfort, it is marked dead, otherwise, carries on. After the determination of the victims, their correspondent events in the PEC are deleted and then the dead specimens are removed from the population.

2.5 Population Simulation

The only package that is harder to extend to another use other than the one it was designed for is the Population Simulation. It's extensible until a certain point, granted that is a simulation based on a population. It's methods are broad enough that an event with the same name (mutation or reproduction) can mean different things with a different context, for instance a simulation of the development of a bee colony, in which the events have different impacts on it's population, but the mold is the same.

This class has a complicated but necessary relation with the rest. Given the nature of the problem the best way to implement this class was to link it with the Events and with the Population. Both classes are of generic nature as an consequence of the Population Simulation being generic. This connection happens because the Population (state) is interfered with by the Events, being the contrary also true. In this way, the Population Simulation serves as a bypass for this indirect relation. Also, to complete the link, the class specimen, as said before, interferes with the events and with the population, taking a generic nature as well. The Population Simulation's role in this scheme of processes is one of higher level, controlling triggers and simulation times.

2.6 Population Event

This is one of the classes that contributes to the general polymorphism of the project. Having inherited from the class Event, it is able to take the form of any of the necessary events in the simulation, defining a condition of validity for the event. It can become any of the three events determined but has enough freedom that it can take any other form, allowing it's use for a different kind of simulation over a population.

3 Execution examples

3.1 Map 1

The first map in which the program was tested was the one given by the teacher. In this one the obstacles force the particle to choose a path with edges with bigger cost. There are two possibilities that it can choose. Or it goes through another edge with bigger cost and do the shortest way or it goes through a normal edge but the path is not the shortest. Since the objective of the problem is getting the path with minor cost the second option should be the one that is more frequent. And so it happens. Although there are sometimes when the final result is the shortest way - random properties of the solution - in the majority of times it gets the correct solution.
In terms of the number of events it is seen that it is normally around the 40000 or around 26000. The last case is spotted when it can't get the "cheapest" path. This happens due to the random properties of the simulation and how the initial population behaves influences a lot what will happen (lower comforts). With less events it is normal that it is more difficult to find the correct solution.

3.2 Map 2

With this map is possible to see how the program performs with a big grid and with the population getting really big. In terms of the final result it is hard to tell whether it is really the best one. Anyway the objective of this test was to show the differences in execution when everything is far bigger. It worked well in the beginning but as the number of specimens, the events and the length of the paths increased it started to become slower. This is what is expected since the population starts to grow over the maximum number and with this come the epidemics which will have to act upon lists with thousands of events or specimens.
It is also possible to notice that the time elapsed until the final point is hit is bigger than in smaller maps. This is an effect of the larger distance between the initial point and the final point.

3.3 Map 3

In this example the idea was to show what happened when the final point is unaccessible. This means that there is no connection available with him because all the adjacent positions are objectives. It works as it was expected. Even when the time of simulation is really big it never hits the final point. The best result will be the point with the best comfort and not the best cost.

3.4 Map 4

This test file is responsible for creating a map in which there are only one possible way to the final point. It has also the characteristic of having really small comfort values and one of the most comfortable points is the initial. This way it is hard to arrive to the final point happening just some times. We can see also that there are no paths with cycles which would be really common with this grid.

3.5 Map 5

In this test file the event parameters were changed. The death parameter is now lower and the reproduction one is bigger. They are responsible for changing the mean value of the random distribution. This way the death time will tend to be minor than the one of reproduction which means that a lot of reproductions will not happen and the population won't grow, in the other side, it will be extinguish faster. This was verified, it was hard to reach the full time of simulation.

4 Conclusion

In summary, the project respected the main focus of its design, extensibility through java tools, as much as possible. All the decisions made were with that focus in mind. If something needed to be noticed it's the over necessity of inter-package dependencies in the simpopulation package, since most classes possessed a one to one relation with another class. This factor made it less prone to be extensible but, one solution found to this problem was the use of generic classes that, through the organization and design of the project, was able to be implemented. With that said, it was a problem inertly associated with the nature of the problem.

The remaining classes were made in such a way that they could be used in other contexts as a whole and separately, not possessing any strong inter dependencies between them.

Besides the aforementioned, functionality wise, the project runs as expected, as it's possible to observe in the notes in chapter three.