Programação Orientada a Objetos

Project 2014/15

Mobile robot Simulator

Description of the problem

The enterprise Guuugle intends to work on mobile robots, having hired various scientists for developing the brains of the robots. The goals are many, including the development of domestic robots, rescue robots (e.g., in earthquakes), or even for planetary exploration. In all those situations the robots will have to live in a dynamic environment, comprising objects of different categories, and learn/map the environment they inhabit (i.e., they have to know where things are and what are those things).

Before and simultaneous to field tests, the performance of the robots is usually tested in controlled simulated environments. Usually, a virtual world is created and populated with agents (virtual representations of the robots) and objects. The agents are provided with reasoning mechanisms (capability for autonomous decision) so that they can execute the same tasks as in a real environment. Several simplifications can be assumed, especially at perception level. In a virtual environment, the processing of visual stimuli can be ignored since the information about the objects can be directly given to the mind of the agents. This way, the agents don't need to do work at this level.

The aim of this project is to develop a simulator of this kind for Guuugle, considering the features presented as follows.

The environment (a 2D matrix) is populated with entities. All those entities have to have a unique identifier, a colour, a geometric shape (for the sake of simplicity, only the well known geometric forms such as triangular, pentagonal, etc., are assumed), and a pair (x,y) of coordinates in the matricial environment. The entities can be of two kinds: agents (represent the robots) or inanimate objects. Objects can be of different kinds (for instance, in the interior of buildings there are chairs, tables, waste bins, etc).

The agents move throughout the environment exploring it in order to acquire information about all the objects that populate it. To do that, agents have a memory container that allows them to store all the acquired information, i.e., a list of all the learned objects. Each agent possess also a visual field which means that an agent percepts only the objects with coordinates belonging to the space delimited by a circumference with radius equal to the visual field parameter and center in the coordinates of the agent.

The agents should move from object to object. At each stop in an object, the agent should obtain the list of objects present in its visual field. Given this list. The agent should chose the next object to visit by making use of one of the following strategies:

i. Random strategy: the next object to be visited is chosen randomly from the list of objects in the visual field.
ii. Strategy of the maximum difference to the objects stored in memory: this strategy allows the agent to select as the next point of view the object that is more different to the objects in memory. The algorithm involves the comparison of each object in the visual field with each object in memory. This difference may be computed using the Hamming distance between two objects. For instance, an object in the visual field characterized by the green colour, rectangular geometric shape, and which is a chair, is at the Hamming Distance of 2 from another object with blue colour, squared geometric shape and also from type chair. After computing all these differences, the minimum of them is taken as the difference between the object and the set of objects

in memory. After computing all these differences for all the objects in the visual field of the agent, the one with the maximum difference is selected to be visited.

iii. Strategy of the closest object: the next object to be visited is the one in the visual field that is geographically closer to the agent.

Each agent should use only one of these strategies which means that there will be 3 types of agents.

The program should allow the following operations:

1. Configure the environment, including the features of the various entities. This configuration should be stored in a file.
2. View and store in a file the memories of the agents along their life.
3. View and store in a file the perceptions (list of object in the visual field) of the agents along their life.
4. View and store in a file the sequence of steps of the life of the agents.
5. View and store in a file the final statistics about the performance of the different agents. Information to compute:
    - Distance traversed by the agents;
    - Number of objects learned;
    - Number of different objects learned.

Develop a GUI for your program that will allow interaction with the user to facilitate data entry and presentation of results.

Implementation

The program should be implemented in Java. As an object-oriented language, it should take into account the following aspects:

1. Each class should manage internally its data, so it must care for the protection of its variables and methods;

2. Each object should be responsible for a specific task or objective, you should not give it undue functions;

3. Use the **static** *keyword* only when necessary and not to circumvent compiler errors.

Make a diagram with your classes and objects (UML), before starting implementation, to predict your project structure. Please also consider the following points that will be important for evaluation:

1. Comment on the classes, methods and public variables in the Javadoc format. This will allow you to automatically generate an HTML file structure, descriptive of your code, which you must include in your report;

2. Comment the rest of the code when algorithms' reading is not obvious;

3. As suggested above, avoid the abusive use of **static** and of **public** variables and methods;

4. In choosing names for variables, classes and methods, follow the conventions adopted in the **Java** language.

5. Each Java class has to be placed in a file with the same name of the class. In each file, as a comment, you should put the name and the number of each student of the group.

Deadlines

The delivery of the work comprises three distinct deadlines:

Deadline 1 (1 point) – Project analysis and class diagram (till 9h00 of **1 December**);

Deadline 2 (1 point) – First version that allows the implementation of requisites 1 through 5 without the use of files (till 9h00 of **15 December**);

Deadline 3 (3 points) – Final version that already manages all the information based on files (till 9h00 of **14 January**).

Works will be compared, in order to detect fraud per copy (between discipline works and from Internet code). Where it is found that there was a copy of all or part of the work the groups involved will have their projects cancelled, failing the discipline.

Material to be delivered

Each group is required to deliver:

Deadline 1: UML Class Diagram

1. Upload the class diagram in InforEstudante as a pdf file.

2. Deliver a printed copy at the class defence.

Deadline 2: First version that allows the implementation of options 1 through 5 without the use of files.

1. Upload zipfile with all the code done so far in InforEstudante.

Deadline 3: Final version that already manages all the information based on files

1. Upload in InforEstudante (mandatory) zipfile with:
   - all .java classes
   - executable
   - test data files.

2. Written report in paper describing the program from a technical point of view and that should include:
   a. General structure of the program;
   b. Class diagrams: initial, interim and final;
   c. Description of the main data structures and files used;
   d. Small description of how the program is run;
   e. Javadoc;
   f. Attached print of the program (except GUI).

3. In **deadline 3** projects can be delivered one day later, with a penalty of 1 point, or two days later, with a penalty of 2 points. Projects are not accepted after these dates.

Project evaluation

For project evaluation two types of factors are considered:

- Black box (as perceived by the user):
  - the set of features implemented;
  - Robustness of the program;
  - Quality of the interface.
- White box (the way it is built):
  - Quality of the technical solutions found to the problems involved; Structuring of the code;
  - the quality of the comments.

The evaluation of each deadline is individually made, regardless of the groups' composition. To the 5 points assigned to the project is associated a minimum of 47.5%. Thus, students who have an evaluation result below 47.5% will not be allowed to any exam.

Note: We do not accept projects that have compilation errors at the defence and that are not properly structured from the perspective of Object Oriented Programming.

Groups' Composition

The work must be done in groups of two elements, and the elements of the group should belong to the same class. The project defence is done individually, as well as its assessment.

Project defence

The work must be defended by an individual classroom discussion. For this, each group must register for a schedule available for the defence at InforEstudante.