Advance Topics In Programming – Exercise

TAU-Robot

Introduction

You are an algorithm developer in an autonomous robotic vacuum cleaner company. You and your colleagues have developed several algorithms for the robots. Your manager asked you to develop a simulator to test the different algorithms for the robotic cleaner, compare between them and decide on the best algorithm. He also wants you to develop a visual (GUI) simulation to present visually how the algorithms actually run and to see their decisions in a simulation.

In this course we will have 4 exercises, for solving and comparing between algorithms for autonomous robotic vacuum cleaners.

The main goal is to write the simulator, which will receive as input:

1. Descriptions of houses (e.g. size of the house, description of the walls) and the spread of the dust in the house.
2. Several algorithms for the cleaning robots.

The simulator will then run all algorithms on all houses, and will compute a score for each of the algorithms according to its performance. The results of all algorithms will be compared and presented to the user. In the last exercise we would also want to run a visual simulation, i.e., sketch the house and present a visualization of the motion of robot(s) during the cleaning process.

Description of a House:

A house is a rectangle of size R x C, with R rows and C columns. We consider the leftmost column being "west" side, the rightmost column begin "east" side, the upper row being "north" side, and the bottom row being "south" side.

There are walls inside the house, partitioning it into rooms. We assume there is a wall surrounding the entire house (that is, the upper row, bottom row, rightmost column and leftmost column are all walls, if this is not the case, the simulation shall turn them into walls).

We assume walls are the only obstacles in the house. We denote walls with the letter W.

Other than walls we have information about the dust levels in the house. For each square unit of the house, we rank the dust level of it using a number in the range 0-9, where 0 means "no dust", and 9 is the maximum dust. Though the digit 0 is legitimate in the house representation, usually it will be denoted as space, representing the exact same meaning: a square with no dust.

There is also a square unit where the docking station is located. We denote the docking station with the letter D.

Here is an example of a house:

North

	W	W	W	W	W	W	W	W	W	W	
	W	2	2			D	W	5	9	W	
West	W			W		1	1	1	9	W	East
	W		W	W	W	3	W	W		W	
	W	6				3	W			W	
	W	7	8	W			W			W	
	W	9	9	W			W			W	
	W	W	W	W	W	W	W	W	W	W	

South

W – wall, D – Docking Station (start position and battery recharge), number 1 to 9 – dust level, empty square – no dust. In this example the house has 8 rows (so R=8), and 10 columns (so C=10).

A house description is stored as a file, as following:
Line 1: House Short Name
Line 2: House Long Description
Line 3: R (number of rows in the house, in our example 8)
Line 4: C (number of columns in the house, in our example 10)
Lines 5 to R+4: a matrix representing the house, where: W-represents wall, D- the docking station, 1-9 the dust level, space/0 – represents 'no dust'.

Our example house is therefore represented as the following text file:

```
Simple1
2 Bedrooms + Kitchen Isle
8
10
WWWWWWWWWW
W22  DW59W
W  W 1119W
W WWW3WW W
W6   3W  W
W78W  W  W
W99W  W  W
WWWWWWWWWW
```

The algorithm:

The purpose of the algorithm is to clean the maximum amount of dust during a limited amount of time. How can we make a sophisticated algorithm?

First of all, there is a problem that the robotic cleaner doesn't know the description of the house, it doesn't know in advance where all the dust and walls are. Instead, it has a sensor. Using the sensor, it may sense its immediate neighborhood as described below. The algorithm may store this information as it is revealed during its route, and use it in order to navigate wisely (or it may not, behaving naively to some extent).

The sensor:

A sensor is attached to the robot and may be used by the algorithm. The sensor can supply the following information:

1. Is there dust in the spot where the robot is currently positioned? And if there is dust, what is the level of the dust on that spot?
2. Is there a wall touching the robot in one of the possible four directions: East, West, North, or South?

The algorithm decision at each step:

Using the sensor, the algorithm for the robotic cleaner should guide it through the house and the cleaning process. It should decide to which direction to advance, it may also decide to stay several time units on the same spot if the amount of dust at that spot is high. So, the possible steps for the algorithm to decide upon are:

1. Stay on the same spot (e.g., a spot which is very dirty may require several steps until it is completely cleaned).
2. Advance to one square to one of the directions: East, West, North, or South (only one of the directions may be chosen in a single step).

The Sensor and The Algorithm – Defining the interface

The sensor implements the following interface:

```
enum class Direction {East, West, South, North, Stay};

struct SensorInformation {
   int dirtLevel;
   bool isWall[4];
};

class AbstractSensor {
public:
   // returns the sensor's information of the current location of the robot
   virtual SensorInformation sense() const = 0;
};
```

Using the sense() method we get information about the walls and dust at the position where the robot is currently located. The algorithm, given that information, should decide what to do during the current time unit (move East/West/South/North or Stay).). Sensor is managed by the simulator and should be updated by the simulator.

In our simulation there is no need to deal with the Robot itself (you may create if you want a Robot class, but the simulation doesn't assume such a class). The simulation is concerned about the algorithm decisions, given the information that the sensor provides. Thus, the simulation should pass a "sensor" to the algorithm, and ask the algorithm for its move decisions.

The algorithm implements the following abstract class:

```
class AbstractAlgorithm {
public:
   // setSensor is called once when the Algorithm is initialized
   virtual void setSensor(const AbstractSensor& sensor) = 0;
   // setConfiguration is called once when the Algorithm is initialized - see below
   virtual void setConfiguration(map<string, int> config) = 0;
   // step is called by the simulation for each time unit
   virtual Direction step() = 0;
   // this method is called by the simulation either when there is a winner or
   // when steps == MaxSteps - MaxStepsAfterWinner
   // parameter stepsTillFinishing == MaxStepsAfterWinner
   virtual void aboutToFinish(int stepsTillFinishing) = 0;
};
```

Time-Unit:

We assume each of the algorithm's step above is done in a single time unit. That is, it takes a single time unit, also called "step", for the robot to move in either direction (or it can decide to stay in the same square). Each time unit the robot sits at a specific square, including the first visit of this square, it cleans one level of dust at this position.

Dust-Levels:

We rank the amount of dust at a spot using 10 dust-levels, numbered 0-9, where 0 represents no dust, and 9 represents the maximal amount of dust. Each time unit a robot is on a specific position, it cleans one level of dust at this spot. For example, assume the robot is on position (1,1), moves to (1,2), moves to (1,3), stays in its position for additional time unit, and then moves to position (1,4). In this journey, the robot cleaned 1 level of dust from positions (1,1), (1,2), (1,4) and two levels of dust from position (1,3).

Battery Constraints and the Docking Station:

To make things more realistic, we also add a battery considerations to the robot, as well as a docking station were the robot is positioned at the beginning. The docking station is also the spot where the robot's battery is charged. When we design the algorithms for the cleaning robots we should make sure that the robot never gets stuck with empty battery – it must always make sure that it has enough battery to go back to its docking station for recharge. If the robot finished cleaning the house, it must go back to its docking station before it is considered that it terminated successfully.

Configuration File:

You should also create and manage a configuration file for the simulation. All the parameters below are assumed to be integer values.

The parameters for the configuration file should be:

- MaxSteps: An execution of an algorithm on a house will run for at most MaxSteps time units. If the algorithm hasn't terminated during this amount of time, we stop running it.

- MaxStepsAfterWinner: If we run a simulation of several algorithms on a specific house, when the first algorithm finishes, we let all other algorithms continue for additional MaxStepsAfterWinner time units. After that time we stop running all algorithms which haven't finished yet. Anyhow, we don't run any algorithm more than MaxSteps time units.
- BatteryCapacity: assuming the battery is fully changed, this is the number of time units a fully charged battery may hold.
- BatteryConsumptionRate: this is the rate at which the battery is consumed during each time unit the robot is not in its docking station. The current battery load is an integer between zero (empty battery) and BatteryCapacity (full-charged). Each time unit the robot is not in its docking station, the current battery load is reduced by this number, per each time unit the robot is not at the docking station.
- BatteryRachargeRate: this is the rate at which the battery is charged. Each time unit the robot is at the docking station, the current battery load increases by the amount RechargeRate.

<u>The Simulator:</u>

The simulator has full information of all components above, and it should simulate the execution of the algorithms on the houses. More precisely:

- The simulator should read all houses descriptions from text files (including dust-levels).
- The simulator should dynamically load libraries for all the algorithms.
- The simulator should execute every algorithm on every house. It should initiate a Sensor and pass it to the Algorithm on start.
- The simulator should track the movement of the robot along the house, understand when the algorithm has finished cleaning, update the dust-levels as the cleaning process goes on and update the sensor on the exact location of the robot, so the sensor may answer its queries - what's the dust level at the current position of

the robot, and is there a wall touching the robot to its east/west/north/south direction?

- For every house, the simulator should run all algorithms in parallel, in a "round-robin" fashion. That is, every time unit we iterate all algorithms and run a single step of the algorithm. We stop either when each algorithm has run for MaxSteps time unite, or MaxStepsAfterWinner time units after the first robot has finished cleaning the house and returned to its docking station (the minimum between the two options).
- In Exercise 3, the run on the different houses will be parallelized using threads.
- If an algorithm makes an invalid step, e.g., walk into a wall, the simulator should identify it; stop the simulation of this specific algorithm, and output that the algorithm made an invalid step.
- If an algorithm runs out of battery, the simulator should identify it and stop the simulation of this specific algorithm.
- In case all algorithms stopped due to bad behavior or exhausted battery or any other reason, simulation on this house shall stop.
- At the end, the simulator should aggregate the performance of each algorithm, and output the score of each algorithm on each house as a comparison matrix. It should also declare on the winning algorithm.
- The score formula is defined as followed:
    - Score = f (
        - position_in_copmetition
        - winner_num_steps
        - this_num_steps
        - dirt_collected
        - sum_dirt_in_house
        - is_back_in_docking

)

- In Exercise 1, the actual score formula would be:
    - Score = Max(0,
        - 2000
        - Minus (position_in_copmetition - 1) * 50
        - Minus (winner_num_steps - this_num_steps) * 10
        - Minus (sum_dirt_in_house - dirt_collected) * 3
        - Plus (is_back_in_docking? 50 : -200)

)
    - All Algorithms which finished cleaning the house will get a position_in_copmetition according to the minimum value of 4 and their actual position in the competition (i.e. this value will not exceed 4 for algorithms which finished cleaning)
    - All Algorithms which didn't finish cleaning the house will get the value 10 for position_in_copmetition
    - In case the simulation stopped due to reaching the MaxSteps limit, winner_num_steps would be = MaxSteps
    - In case Algorithm stopped for bad behavior (trying to get into a Wall), its score would be zero.
- In Exercise 2 and on, the actual score formula can be loaded as a shared library, according to a command-line argument to be detailed later.

Roadmap:

In this course we will have 4 exercises:

Exercise 1 (full details follow below):
Write a simulator which runs one algorithm on several houses:

- Write the simulator program.
- Read houses from files
- Implement a very simple algorithm for the robotic cleaner.
- Using the simulator, run a simulation of the robot's algorithm on all the houses and output the score of its performance.

Exercise 2:

Extend the simulator to support multiple algorithms. The simulator should read multiple houses descriptions, should dynamically load shared object files of several algorithms, and run each algorithm and stop several time units after the first algorithm finishes. Consider this as running a competition between the different algorithms on different houses. The results of the competition between the algorithms should be presented to the user. Write this code without using multithreaded programming.

Command line argument may be used to ask the simulation to load the score formula as a shared library.

Exercise 3:

A. Write the best algorithm you can think of for the robotic cleaner.
B. Make the simulator more efficient by using multithreaded programming. Run in parallel the simulation per each house, using min(X, numHouses) threads, if number of houses > X, reuse threads for more than single house.

Exercise 4:

GUI – using QT, present on screen:
A. "real-time" simulation of a given algorithm on a given house. Draw the house, the walls, and the robot's position and movement.

B. "real-time" simulation of several algorithms together.

We now continue with further details on exercise 1.

Exercise 1 Additional Details

In this exercise your task is to design and implement the simulator for the simple case of a single "hard-coded" algorithm running on multiple houses.

Your task is to:

1. Write a design of your solution. Explain it in text. Include class diagram and sequence-time diagram (30% of the points)
2. Write code to implement your design (70% of the points).

For the algorithm, consider the algorithm which simply chooses each of its steps at random among all valid steps (go east, got north, go south, go west, stay), i.e. among all steps which don't make it run it into a wall.

For the configuration file use this exact information:

<u>config.ini</u>

MaxSteps=1200

MaxStepsAfterWinner=200

BatteryCapacity=400

BatteryConsumptionRate=1

BatteryRachargeRate=20

Your program should generate a single binary: simulator

1. Get the following command-line arguments:
    a. –config <config_file_location >
    b. –house_path <houses_path_location>
2. Both arguments can be set with absolute or relative path
3. Both arguments can be missing, in which case application will look for the files in the working directory.
4. Files to look for:
    a. Config file, named config.ini
    b. House files, files with suffix: .house
5. Implement a single algorithm for a cleaning robot - a very simple one which simply moves at random.
6. Read all house files (files with suffix: .house) from the house_path
7. Run a simulation of the algorithm on the house for at most MaxSteps steps. The simulation stops when either:

- MaxSteps time units have passed, at each time unit we let the robot do a single step (this may be good or bad depending on how much dirt the robot has cleaned). However, we may stop before MaxSteps, if one of the following happens.
- If the house is completely cleaned and the robot has returned to its docking station (good).
- Or if the robot got stuck with an empty battery somewhere in the house and has not made it to the docking station (bad…)
- Another possible reason for stopping the simulation is if the algorithm is deciding to make a decision which is impossible, such as trying to walk through a wall or trying to walk outside the boundaries of the house (also bad).

8. Finally, print the score table of the algorithm for all houses in the format:

   [<House Short Name>]\t<Score>\n
   …
   [<House Short Name>]\t<Score>\n

   Each house name will be surrounded with rectangle brackets, then a tab, then the score, then a new line. A line per each house.

<u>Submission guidelines</u>
You are required to submit your source files. The files must include a <u>comment on top specifying your username and ID</u>. Notice that the file names have to be exactly as indicated above.

You are required to <u>submit the files electronically</u> by placing them in Moodle.

1. Submit a zip file named : EX1_<your id>.zip with all your files.
2. You MUST TEST YOUR CODE on Linux environment before submission
3. Make sure that your output is exactly the same as we requested! We may check your code by automatic scripts, which expect accurate output