Lab 1 - Vacuum Cleaning Agent – Artificial Intelligence

Teacher: Stephan Schiffel

January 8, 2019

Note: For this lab, you can work together in teams of up to 3 students, although ideal is probably 2. You can use Piazza to find team mates and discuss problems.

You will need a Java Development Kit (JDK) and a text editor or Java IDE.

1 Time Estimate

Assuming you went to the lab class, I assume you will need to spend an additional 3h (each student) to finish this assignment. This number may vary a lot depending on your prior experience with programming, Java, reactive controllers and team organization.

2 Problem Description

Your task is to implement the control program for a vacuum cleaner robot. We abstract from some of the complexity of the real world by assuming the environment to be a rectangular grid of cells without obstacles. Some cells may contain dirt. The agent is located somewhere in this grid and facing in one of the four cardinal directions: north, south, east or west. The agent can execute the following actions:

`TURN_ON`: This action initialises the robot and has to be executed first.

`TURN_RIGHT`,`TURN_LEFT`: rotates the robot 90° clockwise/counter-clockwise.

`GO`: lets the agent attempt to move to the next cell in the direction it is currently facing. In case there is an obstacle in that direction, the agent stays at its current location.

`SUCK`: cleans the current cell, has no effect if the current cell is not dirty.

`TURN_OFF`: turns the robot off. Once turned off, it can only be turned on again after emptying the dust-container manually.

The robot is equipped with a dust sensor and a touch sensor. If there is dirt at current location of the robot, the agent will sense `DIRT`. If the robot is bumping into an obstacle or wall, the agent will sense `BUMP`. The goal is to clean all cells and return to the initial location before turning the robot off. Note, a full charge of the battery of the robot will only last for a limited number of actions, but this should be sufficient to clean the whole room.

To make your life easier, the environment does not contain any obstacles except for the walls surrounding it. There are four straight walls, one on each side, that all meet at right angles. That is, the strategy “Go until you bump into a wall then turn right and repeat” will make the agent

walk straight to a wall and then around the room along the wall. The rooms the robot is supposed to clean are all fairly small and you should be able to visit every cell, clean all dirty cells and return to the starting location with executing no more than 100-200 actions.

3 Tasks

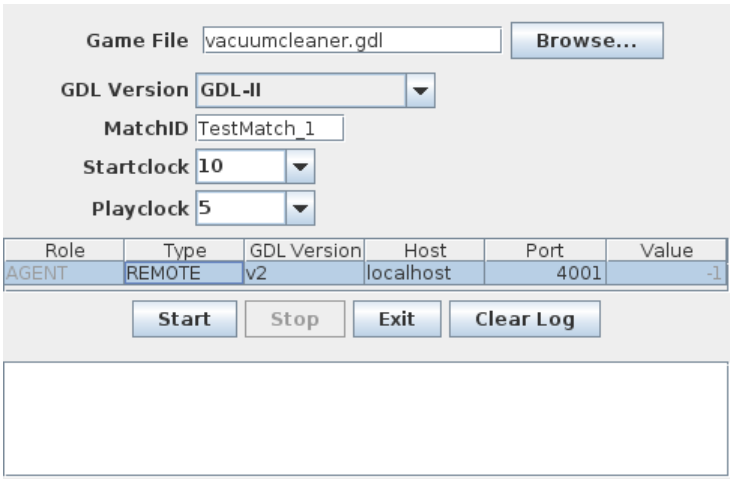
1. (12 points) Characterise the environment (is it static or dynamic, deterministic or stochastic, ...) according to all 6 properties mentioned on slide 13 (Agents) or section 2.3.2 in the book.
2. (10 points) Develop a strategy for the agent such that it fullfills the goal and describe this strategy in a few sentences.
3. (60 points) Implement the missing parts of the vacuum cleaner Java program (see below) such that it encodes your agent function (strategy).
4. (10 points) Test your program with all three provided environments. Test with environments with random in their name can give slightly different results each time, so they should be repeated several times. Record the number of steps it takes to finish each environment.
5. **(5 bonus points)** Optimize the agent to reduce the number of steps it takes to clean an environment. Think about which actions are undesirable in which situation and how to avoid them.
6. (8 points) Is your agent rational? Justify your answer.

4 Material

The Java project for the lab can be found on Canvas inside the archive for this lab: `lab1.zip`. The ZIP archive contains code for implementing an agent in the `src` directory. The agent is actually a server process which listens on some port and waits for the real robot or a simulator to send a message. It will then reply with the next action the robot is supposed to execute.

The zip file also contains the description of three example environments (vacuumcleaner*.gdl) and a simulator (gamecontroller-gui.jar). To test your agent:

- Start the simulator (execute gamecontroller-gui.jar with either double-click or using the command `java -jar gamecontroller-gui.jar` on the command line).
- Setup the simulator as shown in this picture:



- If there is a second role called `RANDOM` in the game, leave its type as `RANDOM`.
- Run the Main class in the project. If you added your own agent class, make sure that it is used in the main method of `Main.java`, or you will just do random moves. You can also execute `ant run` on the command line, if you have Ant installed. The output of the agent should say "NanoHTTPD is listening on port 4001", which indicates that your agent is ready and waiting for messages to arrive on the specified port.
- Now push the "Start" button in the simulator and your agent should get some messages and reply with the actions it wants to execute. At the end, the output of the simulator tells you how many points your agent got: "Game over! results: 0". In the given environment you will only get non-zero points if you manage to clean everything, return to the initial location, and turn off the robot within some reasonable number of steps. If the output of the simulator contains any line starting with "SEVERE", something is wrong. The two most common problems are the network connection (e.g., due to a firewall) between the simulator and the agent or the agent sending illegal moves.
- The output of the simulator shows the true state of the environment (which your agent can not see). Use that for debugging, e.g., to see if you managed to return to the home position.

You can see here, what the example environment looks like. Of course, you should not assume any fixed size, initial location or locations of the dirt in your implementation. This is just an example environment.

5 Hints

For implementing your agent:

- Add a new class that implements the Agent interface. Look at RandomAgent.java to see how this is done.
- You have to implement the method "nextAction" which gets a collection of percepts as input and has to return the next action the agent is supposed to execute.

- Before you start programming a complicated strategy, think about it. The things your agent has to do are: execute `TURN_ON`, visit every cell and suck up any dirt it finds on the way, return to the initial location and `TURN_OFF`. For this, your agent needs some internal model of the world. Figure out what you need to remember about the current state of the world and the agent.
- Implementing this agent is much less error-prone if you think of a state machine encoding the agents internal state where states correspond to the intentions of the agent and state transitions correspond to actions the agent is doing and percepts the agent is perceiving.

As a general hint "Days of programming can save you minutes of thinking.". Think of a strategy, the rules to implement it and the information you need to decide on the actions **before** you start implementing it.

6 Handing In

Hand in a PDF file with the answers to the questions and a zip file with your code on Canvas. The contents of the zip file should have the following structure:

```
lab1
lab1/build.xml
lab1/src
lab1/src/Agent.java
lab1/src/GamePlayer.java
lab1/src/Main.java
lab1/src/NanoHTTPD.java
lab1/src/RandomAgent.java
```

Additional files (like your agent class) should be added in the appropriate place.

Following this structure helps grading the assignment faster. **We will deduct some points for submissions that do not follow this structure and need to manually changed for our tests.**