POLITÉCNICO DE LEIRIA

COMPUTER ENGINEERING DEPARTMENT

COMPUTER ENGINEERING

ARTIFICIAL INTELLIGENCE

2020/2021 – 2nd semester

WORKSHEET 1 - REACTIVE AGENTS

Goals for this worksheet:

- To implement a reactive agent;
- Understanding the architecture and limitations of reactive agents.

Reactive agents are one of the simplest approaches to agent building. These agents are only able to react according to their most recent perceptions. The algorithm for this type of agents can be summarized as follows:

```
function reactive_agent(environment)
 repeat
   perception ← build_perception(environment)
   action ← decide(perception)
   excecute(action)
 while …
end

function decide(perception) : action
 if perception = state1 then return action1
 elseif perception = state2 then return action2
 …
End
```

You are expected to develop a reactive agent which is capable of navigating in an environment consisting of a grid of cells containing obstacles (walls). A cell may be empty, contain an agent and/or dirt, or be occupied by a wall – blocking the movement of the agent. The agent can perceive four positions and move in four directions (the main cardinal points). Whenever the agent visits a dirty cell, it cleans it.

In the course's Moodle page, you will find a scaffold project (ReactiveAgents.zip) containing the base code to be used in this sheet, which implements the base architecture of a reactive agent. You will be mostly working in the `ReactiveAgent` class, but, at some point, some changes should be done to environment cells (Cell class).

In the `ReactiveAgent` class, the `buildPerception()`, the `decide()`, and `execute()` methods are of particular relevance because they correspond to the implementation of the fundamental algorithm of a reactive agent. In method `decide()`, you should call the `decide_a()`, `decide_b()`, `decide_c()` or `decide_d()` methods, depending on the exercise you are trying to solve.

Start by carefully reviewing all the code provided in order to understand the proposed architecture. Then:

a) Implement the decision process of a basic version of the reactive agent, which should simply wander around the world avoiding hitting the walls. This agent doesn't care about dirt, yet (but it always cleans visited cells on this and the next exercises).

b) Implement a memory mechanism that allows the agent to prefer visiting cells that were visited the longest. Your goal is that the agent visits as many cells as possible. This agent doesn't care about dirt, also. You may need to change the `Cell` class…

c) The agent should now be able to perceive dirt in adjacent cells and use that information in its decision process. This agent prefers to visit dirty cells first. In this exercise, the agent should not use any memory mechanism.

d) Finally, implement a decision process in which the agent, besides preferring to visit dirty cells first, it prefers to visit cells that were visited the longest. It should consider the second criterium as a tiebreaker, when more than one adjacent cell is dirty or if there are no dirty adjacent cells.

e) Uncomment the `dirtUpdate()` method in the Environment class and call it in the `run()` method right before calling the agent's `call()` method. Run the application a few times. What do you think about the agent's performance? What modifications do you envisage to improve the agent's performance? Implement them.