Self-organization of robots in a hostile environment
2024-2025

Thanks to the concepts presented in the previous sessions, you now have all the keys in hand to model and simulate your own Agent-based model. So, you are going to model the mission of robots that have to collect dangerous waste, transform it and then transport it to a secure area[1] . The robots navigate in an environment broken down into several zones where the level of radioactivity varies from low radioactive to highly radioactive. Not all robots have access to all areas.
More precisely:

The environment is decomposed into three zones (from west to east):

1. z_1: area with *low radioactivity,* containing a random number of initial (green) waste;
2. z_2 : area *with medium* radioactivity ;
3. z_3: the last area with *high radioactivity*, where completely transformed wastes must be stored.

We have three different types of waste: green waste, yellow waste, and red waste.

We have three different types of robots:

- Green Robot:
  - walk to pick up 2 initial wastes (i.e. green),
  - if possession of 2 green wastes then transformation into 1 yellow waste,
  - if possession of 1 yellow waste, transport it further east,
  - green robot cannot exceed zone z_1.
- Yellow Robot:
  - walk to pick up 2 initial yellow wastes,
  - if possession of 2 yellow wastes then transformation into 1 red waste,
  - if possession of 1 red waste, transport it further east,
  - yellow robot can move in zones z_1 and z_2.
- Red Robot:
  - walk to pick up 1 red waste,

[1] This practical work is adapted and simplified from a real use case discussed with Safran.

- if possession of 1 red waste then transport it further east on the “waste disposal zone”, the waste is then “put away”,
- red robot can move in zones z_1, z_2 and z_3.

Prepare your implementation

You have to create a folder named ***robot_mission_numberofthegroup***, which will contain different files (each file will contain on the top the number of the group, the date of creation, and the names of the members of the group).

The principal elements are:

- ***agents.py*:** this allows you to define the **Robot** agents classes. We have one class per type of robot (greenAgent, yellowAgent and redAgent). Each class will implement the procedural loop of the agent: ***percepts, deliberate, do***. More precisely,
  - the “percepts” allows the agent to get information from the environment. For instance the presence of objects in the cell, the presence of other agents, etc.
  - the “deliberate” corresponds to the “reasoning” step of the agent. It takes as input the “knowledge” (current position, the target, …) that has the agent at each step and returns one or several actions (e.g. moving, collecting, etc.)
  - the “do” allows the agent to inform the environment of its actions (the results of the deliberation process) and, thus, the environment to apply the consequences of these actions. For instance, when an agent collects waste, the latter must no longer exist in the grid. Removing the waste is the responsibility of the environment.

  Thus, the agent step is given by the following method:

```
def step_agent (self):
        update(self.knowledge,  percepts)
        action = deliberate(self.knowledge)
        percepts = self.model.do(self, action)
```

- The attribute **self.knowledge** should represent the beliefs and knowledge of the agent. The representation of these is entirely up to you! (as a bare

minimum, you might consider storing the percepts and actions at each time step).

- The function **deliberate()** is not allowed to access any variable outside its argument.
- The variable **action** describes the action chosen by the agent, among a limited list, such as move to an adjacent tile, pick up, transform, put down, … Its implementation (e.g. as objects, strings, dictionaries, …) is left to you.
- The variable **percepts,** returned by the method Model.do, should contain information about the adjacent tiles and their content – use a dictionary!

- ***objects.py***: which allows you to define different types of agents that do not have behaviors. For instance:

    - the **Radioactivity** agents. This agent will have no behavior but two attributes: the zone to which it belongs and its level of radioactivity (low, medium and high). The level of radioactivity is calculated randomly according to the zone in which this radioactivated agent will be placed (between 0 and 0.33 for z1, between 0.33 and 0.66 for z2 and between 0.66 and 1 for z3). This radioactivity attribute will be used by Robot agents to know in which zone they are!

    - The **Waste Disposal zone** agent: this zone corresponds to a cell of the grid located as far to the east as possible. This cell can be chosen randomly among the eastern cells. The implementation of this zone can be done either by programming a new object agent (without behavior) or by using a radioactivated agent with a particular radioactivity value (this value will allow the robot agents to identify this cell as being the waste disposal zone).
    - the **Waste** agents. This agent will represent the object waste, and will have an attribute allowing to distinguish between green, yellow and red waste.

- ***model.py*****:** defines the RobotMission model itself, and uses the agents and the environment. The model allows the creation of a new RobotMission model with the given parameters. Moreover, do not forget to set up the grid and all the necessary for a visualization

  Moreover, the model is in charge of the execution of actions, with a method called **do**. This method should have as arguments the **agent** performing the action and t**he description of the action**. It should check whether the action is feasible (each action has requirements, and even if the agent believes its action

is feasible, it might be mistaken), then perform the changes entailed by the action.

```
def do (self, agent, action):
    # IF ACTION == MOVE
    ## CHECK IF THE ACTION IS FEASIBLE
    …
    ## IF FEASIBLE, PERFORM THE ACTION
    agent.pos = …
    percepts =      {pos1 : {contents of case at pos1},
                     pos2 : {contents of tile at pos2},
                     …}
    # IF ACTION = …
    …
    return percepts
```

(N.B. this is just an example of a simple implementation – you might consider using e.g. functional programming to represent the actions!)

- ***run.py***: handles the launch of the simulation.
- ***server.py*** contains all the necessary for running the visualisation.

The scenarios

You will try to gradually implement a multi-agent system with different behaviours and features to build the simulation corresponding to the problem of how different robots collaborate to clean the area from dangerous waste.

For each of the following steps:

- Design and implement the situation
- When everything is set up and running, try to extract some data and display these in a chart (for instance, number of wastes in the environment w.r.t the time,...)
- Finally, as we said during the previous sessions, one of the advantages of agent-based models is that we can watch them run step by step, potentially spotting unexpected patterns, behaviors or bugs, or developing new intuitions, hypotheses. So, set up a visualization to observe the implemented scenario.

Step 1: Agents without Communication capabilities

The first step consists in two tasks:

- from the model side, designing all the various components of the simulation: the robots, the wastes, the map, radioactivity, etc, as well as the actions available to agents.
- from the agent side, designing the deliberate function allowing each agent to reason about their environment so as to carry on their task (note that agents in different zones have different tasks). Your first implementation could consider a random agent without any memory, but you soon should equip the agents with a knowledge base storing the map and objects of interest, so that the deliberate function could return a relevant action.

Step 2: Agents with Communication capabilities

The objective here is to endow each Robot agent with communication capabilities. The idea is to enhance the waste collection process between the different agents (to set up a kind of collaboration strategy). We will discuss this after session 4.

Step 3 (you have to finish first steps 1 and 2): Agents with Communication and Uncertainties: TBA.