Implementing Agent Based Model with Mesa library

Self-organization of robots in a hostile environment

AI Training
CentraleSupélec - SAFRAN
27-29.09.2021

Thanks to the concepts presented in the previous section, you now have all the keys in hand to model and simulate your own ABM. 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 (white) 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: white waste, yellow waste, and red waste.

We have three different types of robots:

- White Robot:
  - walk randomly to pick up 2 initial wastes (i.e. white),
  - if possession of 2 white wastes then transformation into 1 yellow waste,
  - if possession of 1 yellow waste, transport it further east,
  - white robot cannot exceed zone z_1.
- Yellow Robot:
  - walk randomly 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 randomly to pick up 1 red waste,

[1] This practical work is adapted from this exercise: http://emmanuel.adam.free.fr/site/spip.php?article80

- 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.

Notes:

- You may have several wastes on the same cell;
- Avoid to have several robots per cell;
- A robot has only a local perception of the environment and his actions/perceptions are limited to the 8 surrounding cells;
- Define different parameters to ease the control of the simulation : initial number of wastes (white, yellow, red), initial number of robots (white, yellow and red).

A. Situation 1: Random Walk...

Question 1 : Implement this ABM.

To help you, in what follows some indications. Of course, you are free to have your own implementation choices.

⚠ Warning: since it is not possible to represent objects with Mesa, you can consider, for instance, that waste, radioactivity (which will help to define the radioactivity areas) are agents who have no behaviours.

You have to create a folder named *robot_mission*, which will contain different files, for instance:

- *robot_mission/random_walk.py*: this defines the **RandomWalker** agent, which implements the behavior of moving randomly across a grid, one cell at a time.
- *robot_mission/agents.py*: this allows you to define the **Robot** agents classes. We have one class per type of robot (whiteAgent, yellowAgent and redAgent). This agent then will have an attribute corresponding to the robot type. Moreover, each class will have an attribute allowing it to check how much waste the robot has (i.e. if it is less than one thus the robot should start looking for the waste).
- *robot_mission/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 z_1, between 0.33 and 0.66 for z_2 and between 0.66 and 1 for z_3). 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 white, yellow and red waste.

- *robot_mission/schedule.py*: defines a custom variant on the **RandomActivation** scheduler, where each type of agent is activated once per step, in random order.
- *robot_mission/model.py*: defines the RobotMission model itself, and uses the agents, the scheduler 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.

  Note: it is suitable to create first the radioactivity agents, the waste Disposal zone, then the waste agents and finally the robot agents.

- *run.py*: handles the launch of the simulation.

Question 2 : 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 session, 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.

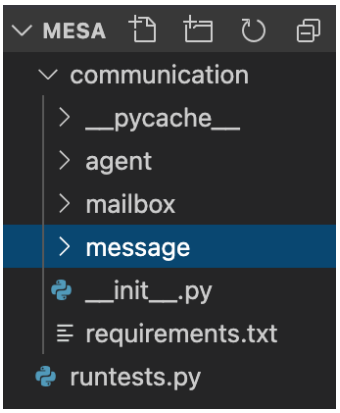
Question 3: Modify your implementation to include a visualization !

You need to add the following file:

- robot_mission/server.py: sets up the interactive visualization server

B. Situation 2: Random walk and Communication

The objective here is to endow each Robot agent with communication capabilities. To do this, we will use the communication layer integrated into Mesa. The idea is to allow each agent type to inform other agents of the availability of a type of waste in a given cell.



For reminder,

- communication: the root folder of the communication layer;
- agent: the folder which will contain the implementation of the communicating agent class;
- mailbox: the folder which will contain the implementation of the mailbox class;
- message: the folder which will contain the implementation of the message and performative class.

For our problem, we will try to include communication as follows:

- White Robot:
  - walk randomly to pick up 2 initial wastes (i.e. white),
  - if possession of 2 white wastes then transformation into 1 yellow waste,
  - if possession of 1 yellow waste, transport it further east,
  - inform the yellow agent of the availability of a yellow waste (position)
  - white robot cannot exceed zone z_1.
- Yellow Robot:
  - Check the mailbox:
    - for all messages, walk to the corresponding position to pick up the yellow wastes.
    - else, walk randomly 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,
    - inform the red agent of the availability of a red waste (position)
    - yellow robot can move in zones z_1 and z_2.
- Red Robot:
    - Check the mailbox:
        - for all messages, walk to the corresponding position to pick up the red waste.
        - else, walk randomly to pick up 1 red waste,
    - 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.

Question 4: Adapt your code, to take the communication aspect. More precisely, you will need to adapt your code to take the following into account:

- the Robot agent are "CommunicatingAgent" (see mesa/communication/agent),
- Agents can move randomly (random_move()) or towards a given position (move_to(position)), depending if they have messages or not.
- In addition to the previous behaviors (question 1), the agents are able to send messages.

Warning: in the model (robot_mission), you need to add the following:

- in the _init_:

```
self.__messages_service = MessageService(self.schedule)
MessageService.get_instance().set_instant_delivery(False)
```

- in the step function: self.__messages_service.dispatch_messages()

C. Situation 3: Planning

Py2PDDL pour écrire et générer du PDDL

- https://github.com/remykarem/py2pddl

Solver

- https://towardsdatascience.com/how-i-implemented-algorithm-in-python-planning-graph-f3ac98351add
- https://pypi.org/project/pyperplan/