Activity 1

November 14, 2022

Team "ERROR 404"

- **José Carlos Martínez Núñez** | A01639664
- **Tania Sayuri Guizado Hernández** | A01640092
- **Gerardo Cortés Leal** | A01639946
- **Óscar Jahir Valdés Caballero** | A01638923

For this problem, you must submit, a PDF report that studies the statistics of a reactive cleaning robot, as well as the link to the Github repository of the code developed for this activity. The code should conform to the style requested in the following document.

Given:

- Room of MxN spaces.
- Number of agents.
- Percentage of initially dirty cells.
- Maximum execution time.

Perform the following simulation:

- Initialize dirty cells (random locations).
- All agents start at cell [1,1].

At each time step: - If the cell is dirty, then it vacuums. - If the cell is clean, the agent chooses a random direction to move (one of the 8 neighboring cells) and chooses the move action (if it cannot move there, it will stay in the same cell). - The maximum set time is executed.

You must collect the following information during the execution:

- Time needed until all cells are clean (or the maximum time has been reached).
- Percentage of clean cells after the end of the simulation.
- Number of movements performed by all agents.

Analyze how the number of agents impacts the time spent, as well as the number of movements performed. Develop a report with the observations.

[ ]:
```
MODEL_PARAMETERS = {
    "ROOM_SIZE": (10, 10),  # H * W of the Room
    "AGENT_NUM": 10, # The number of cleaning robots inside the room
    "DIRTY_PERCENTAGE": 0.4, # Dirty Tiles in Room Percentage
    "MAX_EJECUTION_STEPS": 500, # The maximum number of steps in the simulation
}
```

[143]:
```
import agentpy as ap
import matplotlib.pyplot as plt
import IPython
from enum import IntEnum
from random import choice as select_randomly
```

[144]:
```
class ModelStatus(IntEnum):
    DIRTY = 1
    ROBOT = 2
```

[145]:
```
class DirtyRoom(ap.Model):
    def setup(self):
        # Parameters to Report
        self.cleaned_tiles = 0
        self.agent_moves = 0

        # Dirty Tile Agent Creation
        dirty_tiles = ap.AgentList(self, int(self.p.ROOM_SIZE[0] * self.p.
 ↪ROOM_SIZE[1] * self.p.DIRTY_PERCENTAGE))
        dirty_tiles.type = ModelStatus.DIRTY

        # Robot Creation
        robots = ap.AgentList(self, self.p.AGENT_NUM)
        robots.type = ModelStatus.ROBOT

        # Room Creation
        self.room = ap.Grid(self, self.p.ROOM_SIZE, track_empty=True)
        self.room.add_agents(robots, [(0, 0)] * len(robots), empty=False)
        self.room.add_agents(dirty_tiles, random=True, empty=True)


    def step(self):
        # Looping through each grid position
        for pos in self.room.all:
            robot = None
            dirt = None
            # For each agent located in that position
            for agent in self.room.grid[pos][0]:
                if agent.type == ModelStatus.ROBOT:
                    robot = agent
                elif agent.type == ModelStatus.DIRTY:
                    dirt = agent
            # If we found both a dirty tile and a robot in that position
            # The robot should clean that position
            if dirt and robot:
                self.room.remove_agents(dirt)
                self.cleaned_tiles+=1
```

```
                break

            else:
                # If there was no dirty tile in that position then the robot
↪should move to one
                # of it's neighbors
                if robot:
                    self.room.move_by(robot, select_randomly(
                        [(0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1),
↪(-1, 0), (-1, 1)]))
                    if pos != self.room.positions[robot]:
                        self.agent_moves += 1
                    break
        # Getting List of Currently Active Agents
        active_agents = self.room.agents.to_list()
        # Currently Active Dirt Piles
        dirty_tiles = active_agents.select(active_agents.type == ModelStatus.
↪DIRTY)

        # If we have cleaned the whole grid or if we reached the maximum steps
        # allowed we stop the execution of the simulation
        if len(dirty_tiles) == 0 or self.t == self.p.MAX_EJECUTION_STEPS:
            self.end()
            self.stop()

    def end(self):
        # Reporting data
        self.report('Tiles Cleaned by Robots Percentage',
        round((self.cleaned_tiles / self.p.ROOM_SIZE[0] * self.p.ROOM_SIZE[1])/
↪self.p.DIRTY_PERCENTAGE,1))
        self.report("Time Taken (Steps)", self.t)
        self.report("Moves made by Robots",self.agent_moves)
```

[146]:
```
def animation_plot(model, ax):
    attr_grid = model.room.attr_grid('type')
    color_dict = {ModelStatus.DIRTY: '#92623a', ModelStatus.ROBOT: '#0000FF',
↪None: "#ffffff"}
    ax.set_title(f"Robot Cleaning Simulation", loc="left", fontdict={'family':
↪'Futura', 'color': 'black', 'size': 15})
    ax.set_xlabel(f"Step: {model.t}")
    ap.gridplot(attr_grid, ax=ax, color_dict=color_dict, convert=True)
```

[147]:
```
model = DirtyRoom(MODEL_PARAMETERS)
#fig, ax = plt.subplots()
model.run()  # Modified for PDF Generation
#animation = ap.animate(model, fig, ax, animation_plot)
```

```
#IPython.display.HTML(animation.to_jshtml(fps=15))
```

```
Completed: 500 steps
Run time: 0:00:00.056145
Simulation finished
```

[147]:
```
DataDict {
'info': Dictionary with 9 keys
'parameters':
    'constants': Dictionary with 4 keys
'reporters': DataFrame with 4 variables and 1 row
}
```

[148]:
```
print(model.reporters)
```

```
{'seed': 158401884456030357074192608106337267696, 'Tiles Cleaned by Robots
Percentage': 35.0, 'Time Taken (Steps)': 500, 'Moves made by Robots': 460}
```

After doing some testing of the model it's evident that with a higher number of agents the program takes a lot longer to run and the moves made by all the agents also increases significantly. It is important to note that the time complexity of each step taken is $O(N \times M \times A_n)$, which means the bigger the room and the more agents inside the room the longer the program takes to run. It is worth noting that all the time measurements in the program are made using `steps` as a unit, this is because using time units like seconds or milliseconds will impact the results according to the hardware used to run the program.

The code can also be found in the following Github Repository.