T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4082 – Assignment #1 - Report

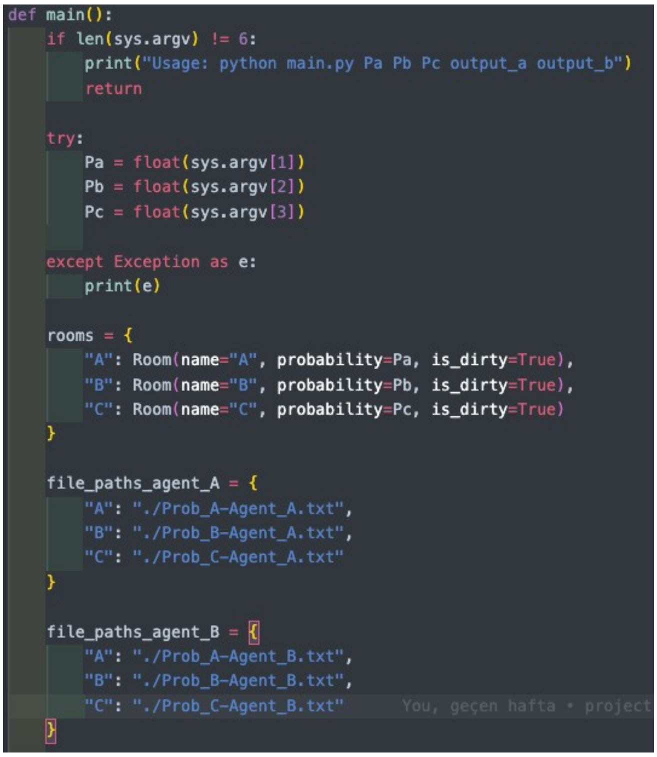
150119671 - Ömercan Göktaş

This project is about programming a smart vacuum cleaner robot that knows how to clean a house with three rooms. The house layout is simple: just three spaces named Room A, Room B, and Room C. The robot must decide on its own the best way to keep all these rooms clean over time.

1. **Room.py**: Class for room objects has implemented in this file. Each room has a room name, dirt probability, variable to identify if the room is dirty or not.
2. **Main.py**: Creating objects for rooms and robots, training robots, and simulating the cleaning processes have implemented in Main file.
3. **Robot.py**: Robot class has implemented in this file. All the actions are taken here, such as moving the agent, increasing, or decreasing point, cleaning the current room, and so on.

All the details about how to obtain dirt probabilities, moving the agent, and other actions are going to be explained below.

1. Main.py

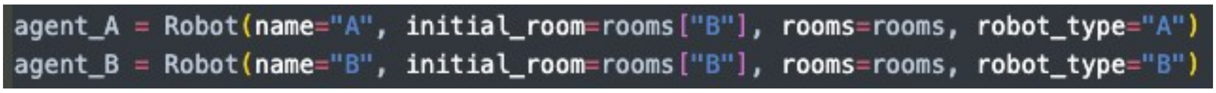


```
def main():
    if len(sys.argv) != 6:
        print("Usage: python main.py Pa Pb Pc output_a output_b")
        return

    try:
        Pa = float(sys.argv[1])
        Pb = float(sys.argv[2])
        Pc = float(sys.argv[3])

    except Exception as e:
        print(e)

    rooms = {
        "A": Room(name="A", probability=Pa, is_dirty=True),
        "B": Room(name="B", probability=Pb, is_dirty=True),
        "C": Room(name="C", probability=Pc, is_dirty=True)
    }

    file_paths_agent_A = {
        "A": "./Prob_A-Agent_A.txt",
        "B": "./Prob_B-Agent_A.txt",
        "C": "./Prob_C-Agent_A.txt"
    }

    file_paths_agent_B = {
        "A": "./Prob_A-Agent_B.txt",
        "B": "./Prob_B-Agent_B.txt",
        "C": "./Prob_C-Agent_B.txt"
    }
```

In this part of main function, arguments are being checked. If user does not provide necessity arguments such as rooms' probabilities and output files' names, the simulation is not going to run.

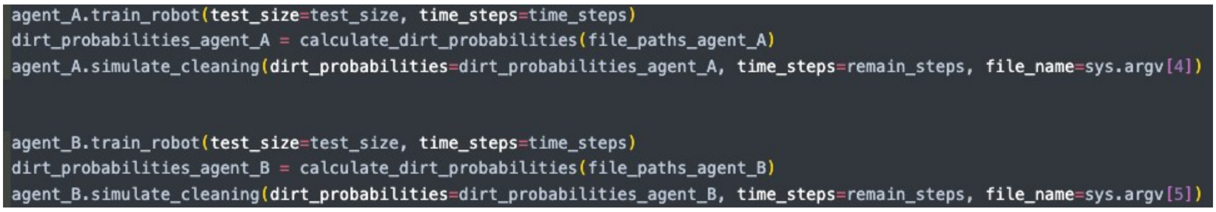
Rooms objects has implemented with dictionary for ease of use. Also, there are other variables as can be seen from the code given above which are file_paths_agent_A and file_paths_agent_B. These variables are going to be used to store obtained room states with time steps. Details will be explained in a specific way later in the **Robot.py** part.

And this is how the agents are initialized in main function:



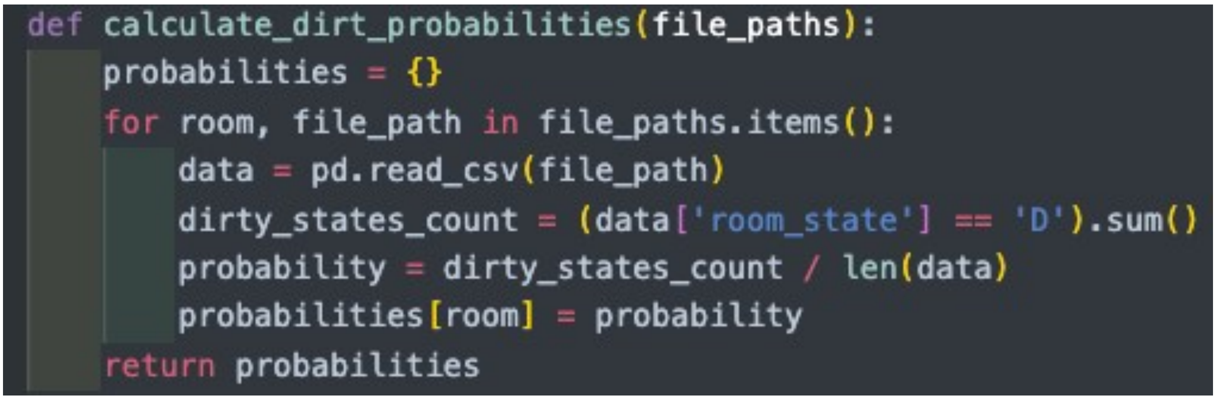
```
agent_A = Robot(name="A", initial_room=rooms["B"], rooms=rooms, robot_type="A")
agent_B = Robot(name="B", initial_room=rooms["B"], rooms=rooms, robot_type="B")
```

The way of simulating the cleaning process is given below:



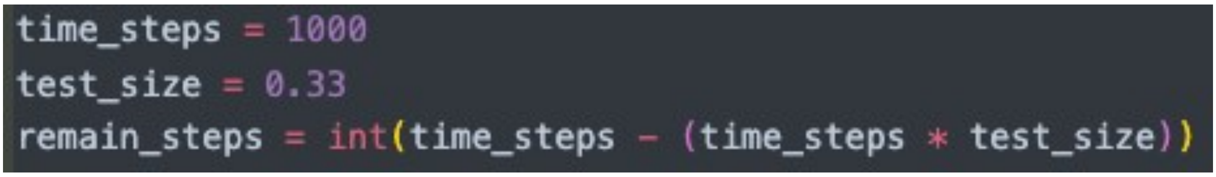
```
agent_A.train_robot(test_size=test_size, time_steps=time_steps)
dirt_probabilities_agent_A = calculate_dirt_probabilities(file_paths_agent_A)
agent_A.simulate_cleaning(dirt_probabilities=dirt_probabilities_agent_A, time_steps=remain_steps, file_name=sys.argv[4])


agent_B.train_robot(test_size=test_size, time_steps=time_steps)
dirt_probabilities_agent_B = calculate_dirt_probabilities(file_paths_agent_B)
agent_B.simulate_cleaning(dirt_probabilities=dirt_probabilities_agent_B, time_steps=remain_steps, file_name=sys.argv[5])
```

Firstly, agents are being trained by the method which we have implemented. After training part, obtained results will be printed in the text files, which were explained before. Then obtained results are going to be analyzed and dirt probabilities for each room will be calculated by calculate_dirt_probabilities function. Lastly, the cleaning process is going to be started by using the obtained dirt probability values by the training part.



```
def calculate_dirt_probabilities(file_paths):
    probabilities = {}
    for room, file_path in file_paths.items():
        data = pd.read_csv(file_path)
        dirty_states_count = (data['room_state'] == 'D').sum()
        probability = dirty_states_count / len(data)
        probabilities[room] = probability
    return probabilities
```

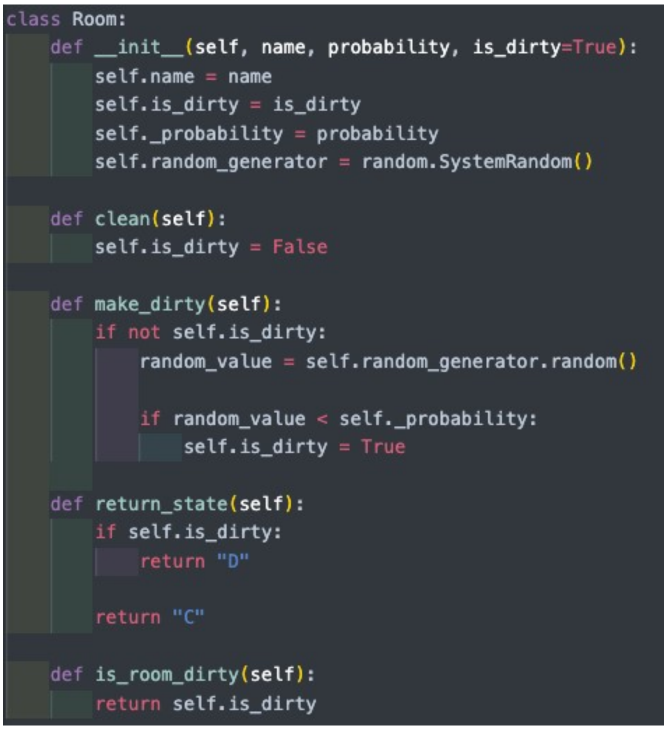
From the code above, obtained room states by the agents with each time step are being used to calculate dirt probabilities for each room.



```
time_steps = 1000
test_size = 0.33
remain_steps = int(time_steps - (time_steps * test_size))
```

Also, as can be seen the code above, each agent has 1000-time step for each action. The test size has been determined as 0.33 which we can say that 330-time step is being used for training each agent and 670-time step is being used to simulate remained cleaning process.

2. Room.py

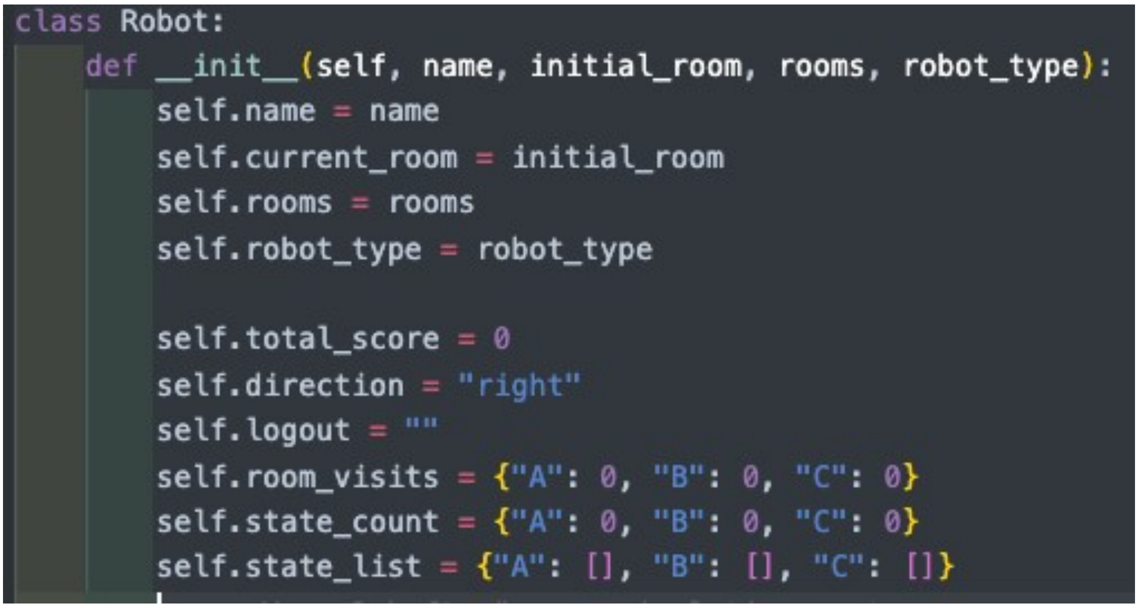


```
class Room:
    def __init__(self, name, probability, is_dirty=True):
        self.name = name
        self.is_dirty = is_dirty
        self._probability = probability
        self.random_generator = random.SystemRandom()

    def clean(self):
        self.is_dirty = False

    def make_dirty(self):
        if not self.is_dirty:
            random_value = self.random_generator.random()

            if random_value < self._probability:
                self.is_dirty = True

    def return_state(self):
        if self.is_dirty:
            return "D"

        return "C"

    def is_room_dirty(self):
        return self.is_dirty
```

2.1. Room Class: This class represents a room and manages its state.

- **clean:** This method is used to mark the room as cleaned by setting the **is_dirty** property to **False**, which means that the current room has cleaned, and its state becomes dirty.
- **make_dirty:** This method calculates the probability of the room becoming dirty and can make the room dirty according to the value calculated with random number generator.
- **return_state:** This method returns a string representing the state of the room "**D**" for dirty or "**C**" for clean.
- **is_room_dirty:** This method checks if the room is dirty or not and returns True or False based on the **is_dirty** property.

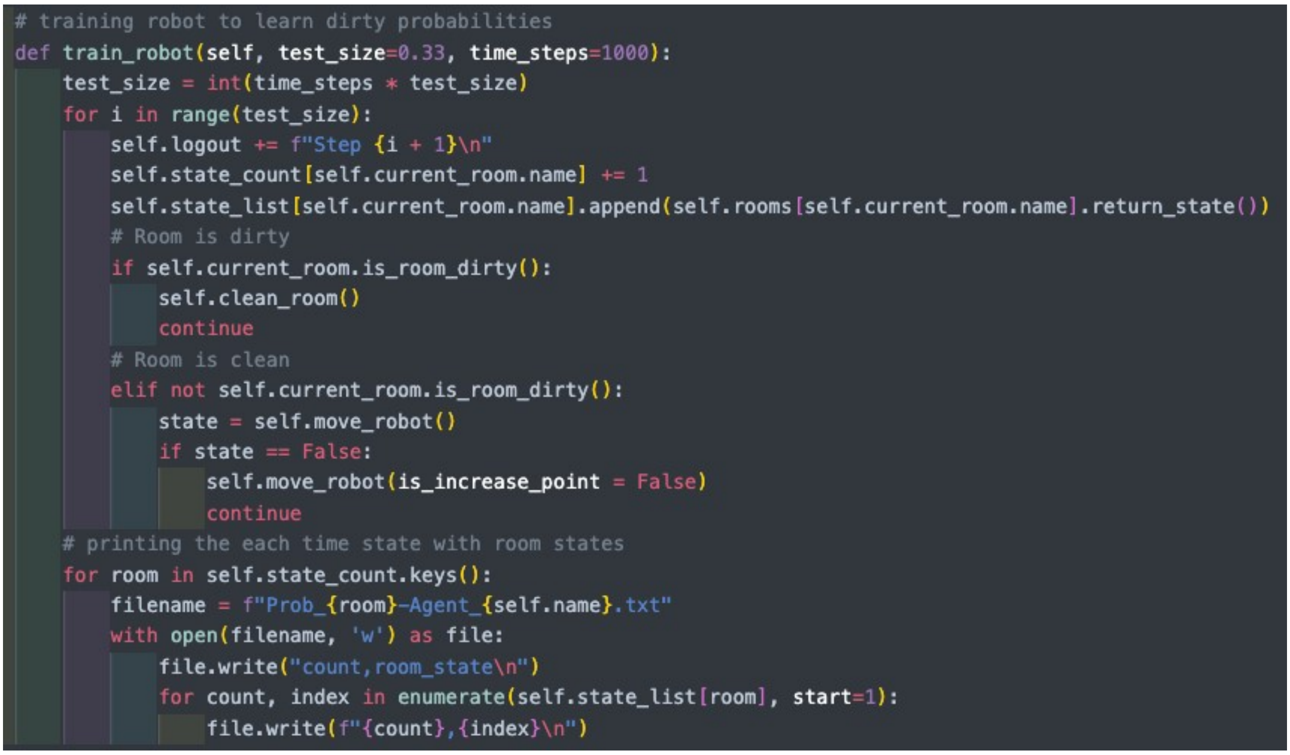
3. Robot.py



```
class Robot:
    def __init__(self, name, initial_room, rooms, robot_type):
        self.name = name
        self.current_room = initial_room
        self.rooms = rooms
        self.robot_type = robot_type

        self.total_score = 0
        self.direction = "right"
        self.logout = ""
        self.room_visits = {"A": 0, "B": 0, "C": 0}
        self.state_count = {"A": 0, "B": 0, "C": 0}
        self.state_list = {"A": [], "B": [], "C": []}
```

As it is obvious the given code above, it is constructor method for each robot agent. When the robot agent object is created, the agent must be trained to obtain dirt probabilities for each room.

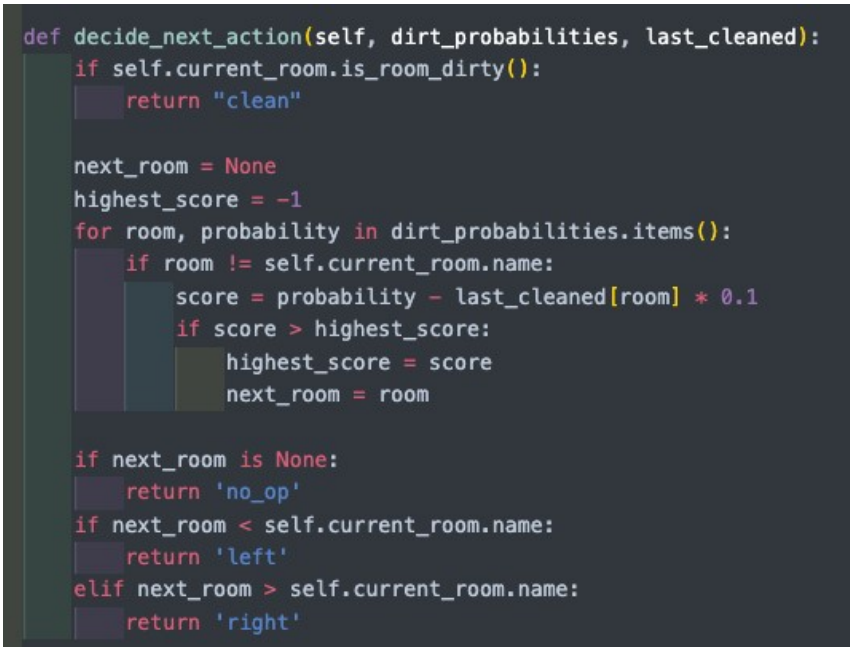
3.1. Training the agent



```
# training robot to learn dirty probabilities
def train_robot(self, test_size=0.33, time_steps=1000):
    test_size = int(time_steps * test_size)
    for i in range(test_size):
        self.logout += f"Step {i + 1}\n"
        self.state_count[self.current_room.name] += 1
        self.state_list[self.current_room.name].append(self.rooms[self.current_room.name].return_state())
        # Room is dirty
        if self.current_room.is_room_dirty():
            self.clean_room()
            continue
        # Room is clean
        elif not self.current_room.is_room_dirty():
            state = self.move_robot()
            if state == False:
                self.move_robot(is_increase_point = False)
                continue
    # printing the each time state with room states
    for room in self.state_count.keys():
        filename = f"Prob_{room}-Agent_{self.name}.txt"
        with open(filename, 'w') as file:
            file.write("count,room_state\n")
            for count, index in enumerate(self.state_list[room], start=1):
                file.write(f"{count},{index}\n")
```

What is the algorithm for training robot?

- The method takes two optional parameters: test_size (default value is 0.33) and time_steps (default value is 1000). These parameters determine the number of training steps and the size of the testing portion.
- The method enters a loop that iterates for test_size times. Within each iteration:
    - It updates the state_count dictionary to keep track of the number of times the robot has visited each room.
    - It appends the current room's state as dirty or clean to the state_list for that room.
- If the current room is dirty self.current_room.is_room_dirty() returns True, the robot cleans the room using the self.clean_room() method and continues to the next step.
- If the current room is clean self.current_room.is_room_dirty() returns False, the robot decides whether to move to another room using the self.move_robot() method.
- It writes the recorded data to a text file named based on the room and agent's name (e.g., "Prob_A-Agent_A.txt") for analyzing the dirt probabilities for each room. **3.2. Simulating the process**



```
def decide_next_action(self, dirt_probabilities, last_cleaned):
    if self.current_room.is_room_dirty():
        return "clean"

    next_room = None
    highest_score = -1
    for room, probability in dirt_probabilities.items():
        if room != self.current_room.name:
            score = probability - last_cleaned[room] * 0.1
            if score > highest_score:
                highest_score = score
                next_room = room

    if next_room is None:
        return 'no_op'
    if next_room < self.current_room.name:
        return 'left'
    elif next_room > self.current_room.name:
        return 'right'
```

- It initializes a dictionary called last_cleaned, which keeps track of the number of steps since each room was last cleaned. By default, it assumes that all rooms were cleaned in the first step except for the current room.
- The simulation enters a loop that iterates for time_steps times. Within each iteration:

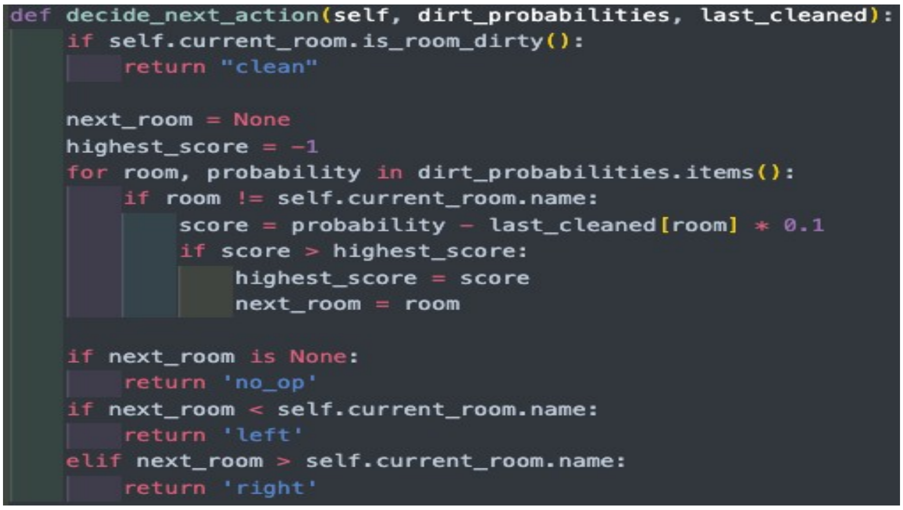
  - It calls the decide_next_action method to determine the robot's next action based on the provided dirt_probabilities and last_cleaned information.

- Depending on the action decided by decide_next_action:

  - If the action is "clean," the robot cleans the current room using the self.clean_room() method, and the number of steps since the room was last cleaned (last_cleaned) is reset to 0 for that room. ○ If the action is "no_op" (no operation), the robot stays in the current room without cleaning. The self.no_op() method is called to simulate this action, and the last_cleaned count for all rooms is increased by 1. ○ If the action is "right" or "left," the robot moves to an adjacent room in the specified direction. The self.direction attribute is set accordingly, and the self.move_robot() method is called. The last_cleaned count for all rooms is increased by 1, and the count for the current room is reset to 0.

- After completing the specified number of time steps, the method saves the simulation results to a file named file_name.

3.3. Deciding next action



```
def decide_next_action(self, dirt_probabilities, last_cleaned):
    if self.current_room.is_room_dirty():
        return "clean"

    next_room = None
    highest_score = -1
    for room, probability in dirt_probabilities.items():
        if room != self.current_room.name:
            score = probability - last_cleaned[room] * 0.1
            if score > highest_score:
                highest_score = score
                next_room = room

    if next_room is None:
        return 'no_op'
    if next_room < self.current_room.name:
        return 'left'
    elif next_room > self.current_room.name:
        return 'right'
```

- It begins by checking if the current room (self.current_room) is dirty using self.current_room.is_room_dirty(). If the current room is dirty, it returns "clean," indicating that the robot should clean the current room.

- If the current room is clean, the method evaluates the dirt probabilities for other rooms to decide the next action.
- It initializes next_room and highest_score variables to None and -1, respectively, to keep track of the room with the highest score.
- For each room, it calculates a score based on the following formula:
    - score = probability - last_cleaned[room] * 0.1
    - It compares the calculated score to the highest_score obtained so far. If the new score is higher, it updates highest_score and sets next_room to the current room being evaluated.

- After evaluating all rooms, if next_room remains None, it means that all rooms have been recently cleaned, and there are no dirty rooms to clean. In this case, it returns 'no_op', indicating that the robot should remain in the current room without cleaning.
- If next_room is not None, it compares the room names to determine whether the robot should move to the left or right. If next_room is True, it returns 'left,’.

4. Mean Value and Standard Deviation of Result

P_A P_B P_C

0.3 0.3 0.3

	Run_1	Run_2	Run_3	Run_4	Run_5	Run_6	Run_7	Run_8	Run_9	Run_10	Mean	SD
Agent_A1	1635	1667	1686	1602	1620	1714	1624	1628	1593	1641	1641	37.69
Agent_B1	1392	1414	1425.5	1398.5	1393	1374	1417	1402.5	1373	1408.5	1393,8333	17.47

P_A P_B P_C

0.5 0.2 0.1

	Run_1	Run_2	Run_3	Run_4	Run_5	Run_6	Run_7	Run_8	Run_9	Run_10	Mean	SD
Agent_A2	1733	1664	1663	1711	1706	1698	1791	1717	1705	1677	1706,5	37.44
Agent_B2	1495.5	1506	1464	1496.5	1501	1476	1450	1465.5	1491.5	1469.5	1479,4	21.51

PA	PB	PC
0.2	0.4	0.2

	Run_1	Run_2	Run_3	Run_4	Run_5	Run_6	Run_7	Run_8	Run_9	Run_10	Mean	SD
Agent_A3	1761	1719	1702	1722	1750	1689	1707	1787	1684	1657	1717,8	39.08
Agent_B3	1467	1452.5	1436	1454.5	1460	1441	1449	1465.5	1455	1459.5	1451,3333	9.93

PA	PB	PC
0.5	0.1	0.3

	Run_1	Run_2	Run_3	Run_4	Run_5	Run_6	Run_7	Run_8	Run_9	Run_10	Mean	SD
Agent_A4	1753	1785	1786	1738	1740	1733	1806	1773	1735	1754	1760,3	25.26
Agent_B4	1522.5	1537.5	1495.5	1522.5	1498	1510.5	1448	1479	1486.5	1489.5	1475	25.7

PA	PB	PC
0.5	0.3	0.8

	Run_1	Run_2	Run_3	Run_4	Run_5	Run_6	Run_7	Run_8	Run_9	Run_10	Mean	SD
Agent_A5	1286	1276	1291	1264	1238	1312	1318	1318	1271	1303	1287,7	26.8
Agent_B5	1099	1151.5	1128.5	1160	1147.5	1157.5	1130	1158	1147.5	1169.5	1136,75	20.59