Assignment 1 (a)

Problem Statement

Simulate a simple two-room vacuum cleaner world where a vacuum cleaner agent is placed in a house with two rooms (Room A and Room B), and its goal is to clean the dirty rooms.

Tasks

1. Implement the **'SimpleVacuumEnvironment'** class that represents the vacuum cleaner world and includes methods to:
   - Check if a room is dirty.
   - Clean a room.
   - Move the agent to another room.
   - Display the current state of the environment.
2. Implement the **'SimpleReflexVacuumAgent'** class that represents the vacuum cleaner agent using a Simple Reflex approach. The agent should have methods to:
   - Perceive the current room and dirt status.
   - Decide actions based on the rules described above.
   - Perform the decided action.
3. Create an instance for each class and simulate the agent's actions for a few steps, displaying the environment's state after each action.

Code:

```
import random

class SimpleVacuumEnvironment:
    def __init__(self):
        # Initialize rooms,room_status,agent_location

        self.room_status={
            "A": "clean",
            "B": "clean"
        }
        self.agent_location="A"

    def is_dirty(self, room):
        # Returns if the room is dirty or not

        if(self.room_status[room]=="dirty"):
            return True
        else:
            return False


    def is_clean(self, room):
        # Returns if the room is clean or not
        if(self.room_status[room]=="clean"):
            return True
        else:
            return False

    def move_agent(self, room):
```

```
        # Assigns the agent_location with room
        self.agent_location=room

    def display(self):
        # Display the current state of the environment.
        print("Room A: ",self.room_status["A"])
        print("Room B: ",self.room_status["B"])


class SimpleReflexVacuumAgent:
    def __init__(self, environment):
        self.environment = environment

    def perceive(self):
        # The agent perceives the current room it is in and whether the room
is dirty or not.
        if(self.environment.is_clean(self.environment.agent_location)):
            return "clean"
        else:
            return "dirty"

    def decide_action(self, dirt_status):
        #  Based on the dirt status, the agent decides whether to clean or
move. If the room is dirty, it chooses to clean; otherwise, it chooses to
move.
        if(dirt_status=="clean"):
            return "move"
        else:
            return "clean"

    def act(self):
        # The agent performs the chosen action. If it decides to clean,
        # it cleans the room and updates the environment's status.
        # If it decides to move, it moves to the other room and updates its
location.
        perceived_status=self.perceive()
        decide_status=self.decide_action(perceived_status)

        if(decide_status=="move"):
            if(self.environment.agent_location=="A"):
                self.environment.move_agent("B")
            else:
                self.environment.move_agent("A")
        elif(decide_status=="clean"):
            self.environment.room_status[self.environment.agent_location]="cle
an"

# Example usage
env = SimpleVacuumEnvironment()
```

```
env.display()
print(" ")
agent = SimpleReflexVacuumAgent(env)
for _ in range(3):
    agent.act()
    env.display()
    print(agent.environment.agent_location)
    print(" ")
```

Output:

```
Room A:  clean
Room B:  clean

Room A:  clean
Room B:  clean
B

Room A:  clean
Room B:  clean
A

Room A:  clean
Room B:  clean
B
```

Assignment 1 (b)

Problem Statement

Simulate a simple two-room vacuum cleaner world where a vacuum cleaner agent is placed in a house with two rooms (Room A and Room B), and its goal is to clean the dirty rooms.

Tasks

1. Implement the **'SimpleVacuumEnvironment'** class that represents the vacuum cleaner world and includes methods to:
   - Check if a room is dirty.
   - Clean a room.
   - Move the agent to another room.
   - Display the current state of the environment.
2. Implement the **'ModelBasedReflexVacuumAgent'** class that represents the vacuum cleaner agent using a Model-Based Reflex approach. The agent should have methods to:
   - Perceive the current room and dirt status.
   - Maintain and update a model of the environment.
   - Decide actions based on the rules described above.
   - Perform the decided action.
3. Create an instance for each class and simulate the agent's actions for a few steps, displaying the environment's state after each action.

Code:

```
import random

class SimpleVacuumEnvironment:
    def __init__(self):
        # Initialize rooms,room_status,agent_location

        self.room_status={
            "A": "dirty",
            "B": "dirty"
        }

        self.agent_location="A"



    def is_dirty(self, room):
        # Returns if the room is dirty or not

        if(self.room_status[room]=="dirty"):
            return True
        else:
            return False


    def is_clean(self, room):
        # Returns if the room is clean or not
        if(self.room_status[room]=="clean"):
            return True
        else:
            return False

    def move_agent(self, room):
        # Assigns the agent_location with room
        self.agent_location=room

    def display(self):
        # Display the current state of the environment.
        print("Room A: ",self.room_status["A"])
        print("Room B: ",self.room_status["B"])


class ModelBasedReflexVacuumAgent:
    def __init__(self, environment):
        # Initialize environment and model
        self.environment=environment
        self.model_room_status={
            "A": "dirty",
            "B": "dirty"
```

```
        }

    def perceive(self):
        # Perceive and return the current room and dirt status.
        if(self.environment.is_clean(self.environment.agent_location)):
            return "clean"
        else:
            return "dirty"

    def update_model(self, room):
        # The agent updates its model of the environment by marking
        # the current room as clean in its model.
        self.model_room_status[room]="clean"

    def decide_action(self, dirt_status):
        # the agent decides whether to clean or move based on its perception
and model.
        # If the room is dirty in the actual environment, it chooses to clean.
        # If the room is clean in the actual environment but marked as dirty
in the model,
        # it still chooses to clean based on the model. Otherwise, it chooses
to move.
        if(dirt_status=="dirty"):
            return "clean"
        elif(dirt_status=="clean"):
            if(self.model_room_status[self.environment.agent_location]=="dirty
"):
                return "clean"
            else:
                return "move"


    def act(self):
        # The agent performs the chosen action.
        # If it decides to clean, it cleans the room in both the actual
environment
        # and its model, and updates the model. If it decides to move, it
moves to
        # the other room and updates its location.
        # The agent performs the chosen action. If it decides to clean,
        # it cleans the room and updates the environment's status.
        # If it decides to move, it moves to the other room and updates its
location.
        perceived_status=self.perceive()
        decide_status=self.decide_action(perceived_status)

        if(decide_status=="move"):
            if(self.environment.agent_location=="A"):
                self.environment.move_agent("B")
```

```
            else:
                self.environment.move_agent("A")
        elif(decide_status=="clean"):
            self.environment.room_status[self.environment.agent_location]="cle
an"
            self.model_room_status[self.environment.agent_location]="clean"


# Example usage
env = SimpleVacuumEnvironment()
env.display()
print(" ")
agent = ModelBasedReflexVacuumAgent(env)
for _ in range(3):
    agent.act()
    env.display()
    print(agent.environment.agent_location)
    print(" ")
```

Output:

```
 Room A:  dirty
 Room B:  dirty

 Room A:  clean
 Room B:  dirty
 A

 Room A:  clean
 Room B:  dirty
 B

 Room A:  clean
 Room B:  clean
 B
```

Assignment 1 (c)

Problem Statement

Simulate a simple two-room vacuum cleaner world where a vacuum cleaner agent is placed in a house with two rooms (Room A and Room B), and its goal is to clean the dirty rooms.

Tasks

1. Implement the **'SimpleVacuumEnvironment'** class that represents the vacuum cleaner world and includes methods to:
   - Check if a room is dirty.
   - Clean a room.
   - Move the agent to another room.
   - Display the current state of the environment.
2. Implement the **'GoalBasedVacuumAgent'** class that represents the vacuum cleaner agent using a Goal-Based Agent approach. The agent should have methods to:
   - Set goals with actions and priorities.
   - Prioritize goals based on their priorities.
   - Perceive the current room and dirt status.
   - Decide actions based on the highest-priority goal.
   - Perform the decided action.
3. Create an instance for each class and simulate the agent's actions for a few steps, displaying the environment's state after each action.

Code:

```
import random

class SimpleVacuumEnvironment:
    def __init__(self):
        # Initialize rooms,room_status,agent_location

        self.room_status={
            "A": "dirty",
            "B": "dirty"
        }

        self.agent_location="A"
    def is_dirty(self, room):
        # Returns if the room is dirty or not

        if(self.room_status[room]=="dirty"):
            return True
        else:
            return False


    def is_clean(self, room):
        # Returns if the room is clean or not
        if(self.room_status[room]=="clean"):
            return True
        else:
            return False

    def move_agent(self, room):
        # Assigns the agent_location with room
```

```
        self.agent_location=room

    def display(self):
        # Display the current state of the environment.
        print("Room A: ",self.room_status["A"])
        print("Room B: ",self.room_status["B"])


class GoalBasedVacuumAgent:
    def __init__(self, environment):
        # Initialize environment and goals
        self.environment=environment
        self.priority_to_execute=1
        self.goals={}

    def set_goal(self, goal):
        # Set goals with actions and priorities
        self.goals[goal[1]]=goal[0]

    def prioritize_goals(self):
        # Sort goals based on priority criteria
        return

    def perceive(self):
        # Perceive and return the current room and dirt status.
        if(self.environment.is_clean(self.environment.agent_location)):
            return "clean"
        else:
            return "dirty"

    def decide_action(self, dirt_status):
        # The agent decides on the action to take based on the highest
        # priority goal. If there are no goals, the agent will have no action.
        return self.goals[self.priority_to_execute]

    def act(self):
        # The agent performs the decided action. If the action is to clean,
        # it cleans the room. If the action is to move, it moves to the target
room.
        self.priority_to_execute=1

        perceived_status=self.perceive()




        while(self.priority_to_execute<=len(self.goals)):
            decided_action=self.decide_action(perceived_status)
```

```
            if(decided_action=="move"):
                if(self.environment.agent_location=="A"):
                    self.environment.move_agent("B")
                else:
                    self.environment.move_agent("A")

                return
            elif(decided_action=="clean"):
                self.environment.room_status[self.environment.agent_location]=
"clean"

            self.priority_to_execute=self.priority_to_execute+1
# Example usage
env = SimpleVacuumEnvironment()
agent = GoalBasedVacuumAgent(env)
agent.set_goal(("clean", 2))  # Priority 1: Clean the room
agent.set_goal(("move", 1))  # Priority 2: Move to room B
agent.prioritize_goals()

env.display()
print(" ")

for _ in range(5):
    agent.act()
    env.display()
    print(agent.environment.agent_location)
    print(" ")
```

Output:

```
Room A:  dirty
Room B:  dirty

Room A:  dirty
Room B:  dirty
B

Room A:  dirty
Room B:  dirty
A

Room A:  dirty
Room B:  dirty
B

Room A:  dirty
Room B:  dirty
A

Room A:  dirty
Room B:  dirty
B
```

Assignment 1 (d)

Problem Statement

Simulate a simple two-room vacuum cleaner world where a vacuum cleaner agent is placed in a house with two rooms (Room A and Room B), and its goal is to clean the dirty rooms.

Tasks

1. Implement the **'SimpleVacuumEnvironment'** class that represents the vacuum cleaner world and includes methods to:
   - Check if a room is dirty.
   - Clean a room.
   - Move the agent to another room.
   - Display the current state of the environment.
2. Implement the **'UtilityBasedVacuumAgent'** class that represents the vacuum cleaner agent using a Utility-Based Agent approach. The agent should have methods to:
   - Calculate utility values for each room based on cleanliness.
   - Decide actions based on utility values.
   - Perform the decided action.
3. Create an instance for each class and simulate the agent's actions for a few steps, displaying the environment's state after each action.

Code:

```
import random

class SimpleVacuumEnvironment:
    def __init__(self):
      self.room_status = {
          'A': random.choice(['Clean', 'Dirty']),
          'B': random.choice(['Clean', 'Dirty'])
      }
      self.agent_location = random.choice(['A', 'B'])

    def is_dirty(self, room):
        return self.room_status[room] == 'Dirty'

    def clean(self, room):
        self.room_status[room] = 'Clean'

    def move_agent(self, room):
        self.agent_location = room

    def display(self):
        print("Room A:", self.room_status['A'])
        print("Room B:", self.room_status['B'])
        print("Agent Location:", self.agent_location)
        print()

class UtilityBasedVacuumAgent:
    def __init__(self, environment):
        self.environment = environment
        self.utilities = {'A': 0, 'B': 0}

    def calculate_utilities(self):
```

```
        for room in self.utilities:
            if self.environment.is_dirty(room):
                self.utilities[room] = -1 # Negative utility for dirty rooms
            else:
                 self.utilities[room] = 1 # Positive utility for clean rooms
    def decide_action(self):
        current_room = self.environment.agent_location
        other_room = 'B' if current_room == 'A' else 'A'

        if self.utilities[current_room] < self.utilities[other_room]:
            return "Clean" if self.environment.is_dirty(current_room) else
"Move"
        else:
             return "Move"
    def act(self):
        current_room, dirt_status = self.environment.agent_location, self.envi
        action = self.decide_action()

        if action == "Clean":
            self.environment.clean(current_room)
            print("Agent cleans", current_room)
        elif action == "Move":
            target_room = 'B' if current_room == 'A' else 'A'
            self.environment.move_agent(target_room)
            print("Agent moves to", target_room)
    self.environment.display()
# Example usage
env = SimpleVacuumEnvironment()
agent = UtilityBasedVacuumAgent(env)

for _ in range(5):
    agent.calculate_utilities()
    print(f"Utility Values: {agent.utilities}")
    agent.act()
    print()
```

Output:

```
Utility Values: {'A': 1, 'B': 1}
Agent moves to A
Room A: Clean
Room B: Clean
Agent Location: A


Utility Values: {'A': 1, 'B': 1}
Agent moves to B
Room A: Clean
Room B: Clean
Agent Location: B


Utility Values: {'A': 1, 'B': 1}
Agent moves to A
Room A: Clean
Room B: Clean
Agent Location: A


Utility Values: {'A': 1, 'B': 1}
Agent moves to B
Room A: Clean
Room B: Clean
Agent Location: B
```