Task: Building a Table-Driven Vacuum Agent

Step 1: Setting up the Environment

In this step, you will build the "World" where the vacuum lives.

1. Define a class named **VacuumEnvironment**.
2. Create the constructor **(__init__)** that accepts two parameters: rooms and **start_location**. Inside, set **self.rooms** to the rooms parameter and **self.location** to the start location.
3. Define a method called percept that takes only self. It should return a tuple containing the current location and the status of that location (Clean or Dirty) from the **self.rooms** dictionary.
4. Define a method called execute that takes action as a parameter.
    a. If the action is **'Suck'**, set the status of the current room in **self.rooms** to **'Clean'**.
    b. If the action is **'Right'**, change the **self.location** to **'B'**.
    c. If the action is **'Left'**, change the **self.location** to **'A'**.

Step 2: Building the Agent's Brain

Now, create the agent that remembers what it has seen and looks up what to do.

1. Define a class named **TableDrivenAgent**.
2. Create the constructor that accepts a table. Inside, save the table to **self.table** and initialize an empty list named **self.percept_sequence**.
3. Define a method named **agent_function** that takes percept as a parameter.
    a. First, append the new percept to the **self.percept_sequence**.
    b. Next, create a loop that iterates through the length of the percept sequence.
    c. Inside the loop, create a tuple of the sequence from the current index to the end.
    d. Check if that tuple exists in **self.table**. If it does, **return** the action associated with it in the table.
    e. If the loop finishes without finding anything, **return None**.

Step 3: Defining the Knowledge & Running the Task

Now, you will provide the "Rules" and run the simulation loop.

4. Create a dictionary named **sequence_table**. Use **tuples** of percepts as keys and strings as actions.

Hint: Map (('A', 'Dirty'),) to 'Suck', (('A', 'Clean'),) to 'Right', and sequences like (('A', 'Clean'), ('B', 'Clean')) to 'Exit'.

5. Initialize the environment by creating an object of **VacuumEnvironment**. Pass it a dictionary where **Room A** is **'Clean'** and **Room B** is **'Dirty'**.
6. Initialize the agent by creating an **object** of **TableDrivenAgent** and passing it your **sequence_table**.
7. Create an infinite loop (**while True**) to run the simulation:
    a. Call the percept method from the environment and store it.
    b. Pass that percept to the agent's **agent_function** and store the resulting action.
    c. **Print** the current percept and the chosen action so you can see the progress.
    d. Check if the action is **'Exit'**. If so, **print** a success message and break the loop.
    e. Otherwise, **call** the **execute method** on the environment using the chosen action.

Structure of the Code: Write this code in HTML <py-script> tag.

```
from pyscript import display
class VacuumEnvironment:
    def __init__(self, rooms, start_location='A'):
        /** Write you Code Here **/
    def percept(self):
        /** Write you Code Here **/
    def execute(self, action):
        /** Write you Code Here **/
class TableDrivenAgent:
    def __init__(self, table):
       /** Write you Code Here **/
    def agent_function(self, percept):
       /** Write you Code Here **/
    sequence_table = {
    (('A', 'Dirty'),): 'Suck',   (('A', 'Clean'),): 'Right',     (('B', 'Dirty'),): 'Suck',
    (('B', 'Clean'),): 'Left',   (('A', 'Clean'), ('B', 'Clean')): 'Exit',
    (('B', 'Clean'), ('A', 'Clean')): 'Exit'
}
    rooms = {'A': 'Clean', 'B': 'Dirty'}
    env = VacuumEnvironment(rooms)
    agent = TableDrivenAgent(sequence_table)
    /** Write loop Code Here **/
    env.execute(action)
```

OUTPUT:

Table-Driven Agent (Vacuum World – AIMA)

Table-Driven Agent Execution

Percept: ('A', 'Clean') -> Action: Right
Percept: ('B', 'Dirty') -> Action: Suck
Percept: ('B', 'Clean') -> Action: Left
Percept: ('A', 'Clean') -> Action: Exit
All rooms clean. Agent exits.