3-Practice

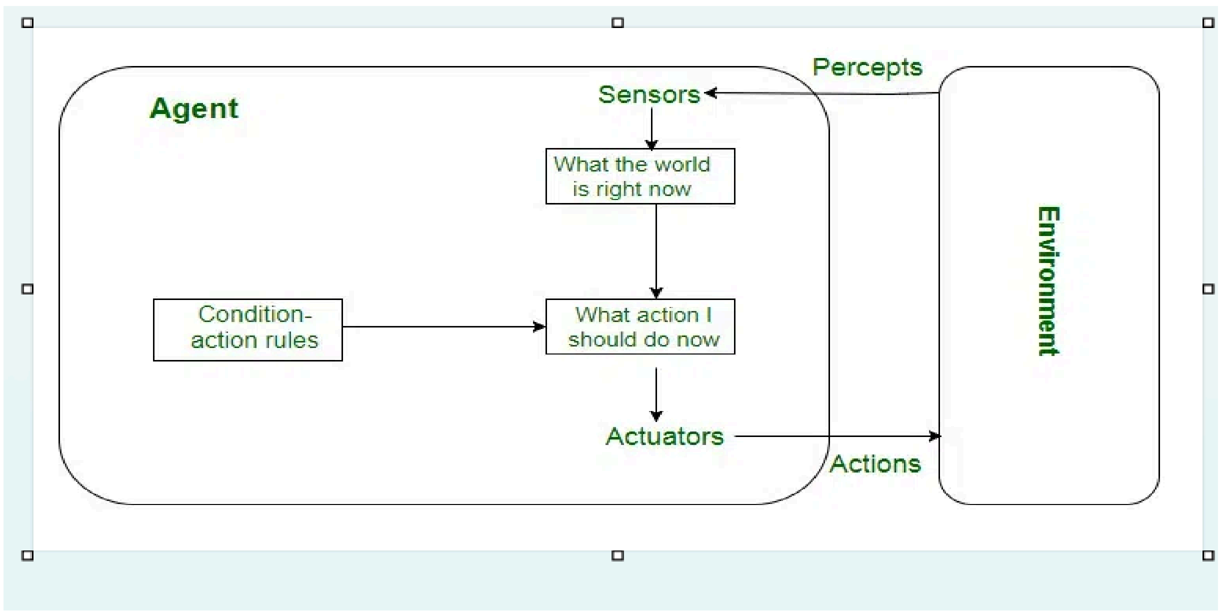
Theme: Agent Architecture. Designing a Reflexive Agent.

The objective of this work is to introduce students to the concept of agent architecture, with a particular focus on studying the operating principles of reflex agents from both theoretical and practical perspectives. Within the scope of this work, the simplest reflex agent and model are developed using the Mesa framework, and the agent's perception–action mechanism is demonstrated through practical examples.

Theorica session

A reflex agent is a type of agent that can independently make decisions and act based on the current state of its environment. Reflex agents are typically designed to perceive information from the environment and send control commands based on that information. The main characteristic of this type of agent is that it considers only the current state and does not rely on historical data.

The term reflex agent design is commonly used in artificial intelligence (AI) and robotics. A reflex agent is a simple type of agent that responds immediately to signals (inputs) from the environment without maintaining any internal state. It makes decisions solely based on information available at the present moment.



Key Characteristics of a Reflex Agent

1. **Acts Based on the Current State:** A reflex agent considers only the **current state** of the environment to make decisions and take actions.
2. **Simplicity:** Reflex agents are **simple**, as they operate solely based on the current state and do not maintain or use historical data.
3. **Autonomy:** Reflex agents are **highly autonomous**, because they make decisions independently and act without relying on external guidance.

The main idea of designing a reflex agent in Python is to ensure that the agent **acts based on the current state of its environment**. As a simple example, we will implement a **real-time reflex agent in Python** for a room-cleaning robot.

In this example:

- **Environment:** The room (either clean or dirty).
- **Sensors:** To detect the current state of the room.
- **Actuators:** To turn the cleaning unit on/off and move the robot.
- **Rule Set:** Defines what action the robot should take based on the current state.

```
import random

class Environment:
    def __init__(self):
        # Room condition: 'dirty' or 'clean'
        self.locations = ['A', 'B']
        self.state = {'A': 'dirty', 'B': 'dirty'}
        self.current_location = random.choice(self.locations)

      def get_percept(self):
        """Provides information about the environment to the
agent."""
        return (self.current_location,
self.state[self.current_location])

    def change_state(self, location):
        """The agent updates the state of the environment when it
moves and changes location."""
        if self.state[location] == 'dirty':
            self.state[location] = 'clean'
```

```
class ReflexVacuumAgent:
    def __init__(self):
        pass

    def action(self, percept):
        """Using Percept, the agent determines how it should act."""
        location, condition = percept

        if condition == 'dirty':
            print(f"Location {location} is dirty. Cleaning...")
            return 'suck'
        else:
            other_location = 'B' if location == 'A' else 'A'
            print(f"Location {location} is clean. Moving to
{other_location}...")
            return 'move'

            # Main programming
if __name__ == "__main__":
    env = Environment()  # Creating environment
    agent = ReflexVacuumAgent()  # Creating reflexive agents

    for step in range(5):  # We show agent activity in 5 cycles
        print(f"\n--- Step {step + 1} ---")
        percept = env.get_percept()
print(f"Percept: {percept}")

        action = agent.action(percept)
        if action == 'suck':
            env.change_state(percept[0])  # Turn on the cleaning
column and update the environment status.
        elif action == 'move':
            env.current_location = 'B' if env.current_location ==
'A' else 'A'  # Change location

        print(f"Updated Environment State: {env.state}")
        print(f"Agent's Current Location: {env.current_location}")
```

Result:

```
--- Step 1 ---
Percept: ('A', 'dirty')
Location A is dirty. Cleaning...
Updated Environment State: {'A': 'clean', 'B': 'dirty'}
Agent's Current Location: A

--- Step 2 ---
Percept: ('A', 'clean')
Location A is clean. Moving to B...
Updated Environment State: {'A': 'clean', 'B': 'dirty'}
Agent's Current Location: B
```

```
--- Step 3 ---
Percept: ('B', 'dirty')
Location B is dirty. Cleaning...
Updated Environment State: {'A': 'clean', 'B': 'clean'}
Agent's Current Location: B

--- Step 4 ---
Percept: ('B', 'clean')
Location B is clean. Moving to A...
Updated Environment State: {'A': 'clean', 'B': 'clean'}
Agent's Current Location: A

--- Step 5 ---
Percept: ('A', 'clean')
Location A is clean. Moving to B...
Updated Environment State: {'A': 'clean', 'B': 'clean'}
Agent's Current Location: B
```