MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE
" IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE"

Volodymyr Shymkovych

Technologies of Artificial Intelligence

LABORATORY WORK #1

Intelligent agents

Kulubecioglu Mehmet

IM-14 FIOT

Kyiv
IHORY SIKORSKY KYIV POLYTECHNIC INSTITUTE
2024

Introduction

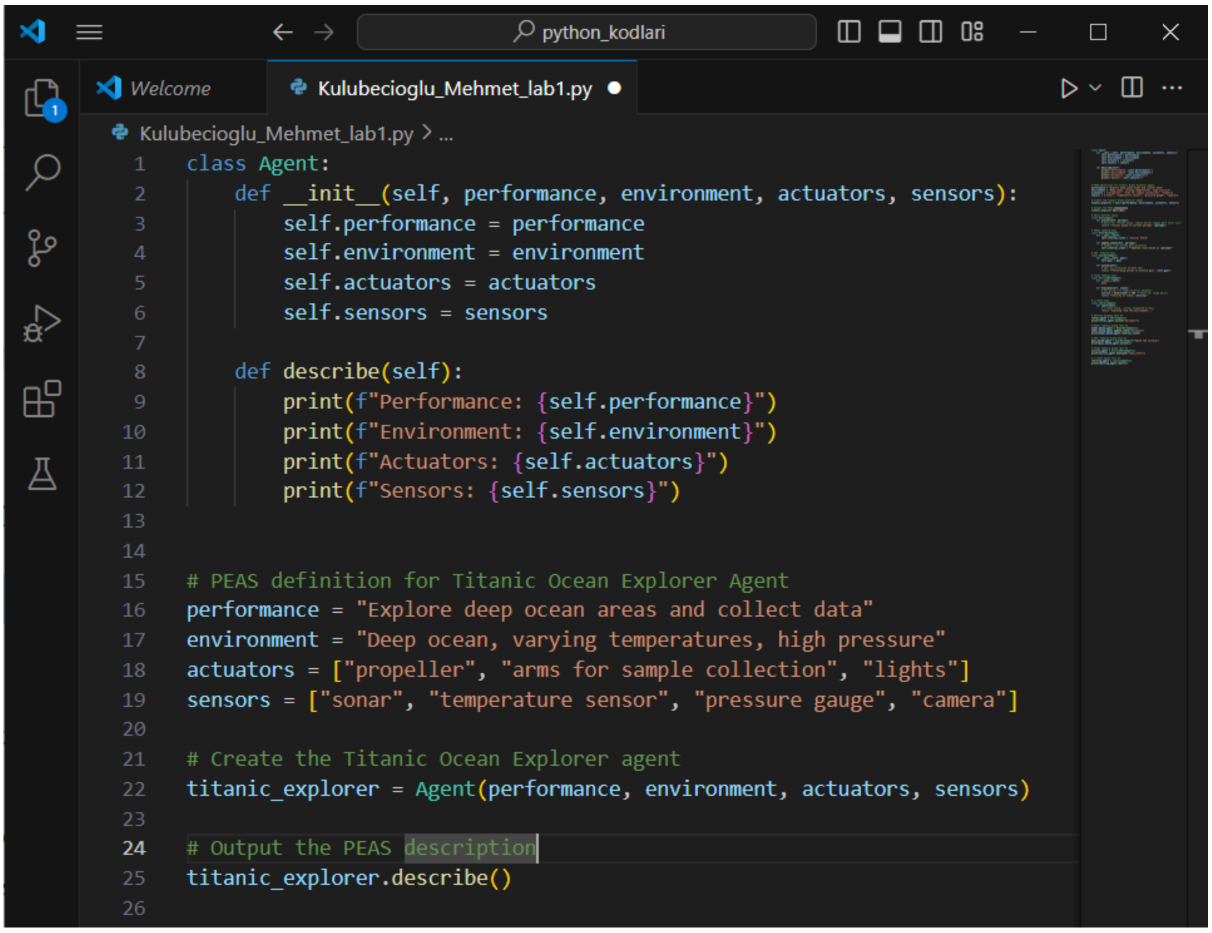
In this lab assignment, I am tasked with designing and testing various types of agents following the PEAS framework (Performance, Environment, Actuators, Sensors). The agents include reflex agents, model-based agents, goal-based agents, utility-based agents, and learning agents. I will walk through the steps of building each agent, explaining the code associated with each step, and the expected output.

Step 1: Define the PEAS Framework for a Titanic Ocean Explorer Agent

The first step in building an agent is to define the **PEAS** model, which stands for:

- **Performance**: What the agent is supposed to achieve (goal).
- **Environment**: Where the agent operates.
- **Actuators**: The mechanisms through which the agent can interact with the environment.
- **Sensors**: The tools the agent uses to perceive the environment.

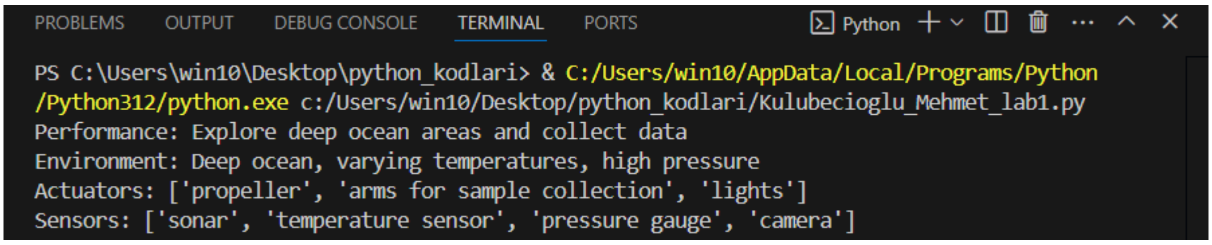
I start by creating a Python class called `Agent` that will represent the Titanic Ocean Explorer agent. This agent will explore the deep ocean, collect data, and navigate using various sensors and actuators.



python_kodlari
Welcome
Kulubecioglu_Mehmet_lab1.py
Kulubecioglu_Mehmet_lab1.py > ...

```python
class Agent:
    def __init__(self, performance, environment, actuators, sensors):
        self.performance = performance
        self.environment = environment
        self.actuators = actuators
        self.sensors = sensors

    def describe(self):
        print(f"Performance: {self.performance}")
        print(f"Environment: {self.environment}")
        print(f"Actuators: {self.actuators}")
        print(f"Sensors: {self.sensors}")


# PEAS definition for Titanic Ocean Explorer Agent
performance = "Explore deep ocean areas and collect data"
environment = "Deep ocean, varying temperatures, high pressure"
actuators = ["propeller", "arms for sample collection", "lights"]
sensors = ["sonar", "temperature sensor", "pressure gauge", "camera"]

# Create the Titanic Ocean Explorer agent
titanic_explorer = Agent(performance, environment, actuators, sensors)

# Output the PEAS description
titanic_explorer.describe()

```

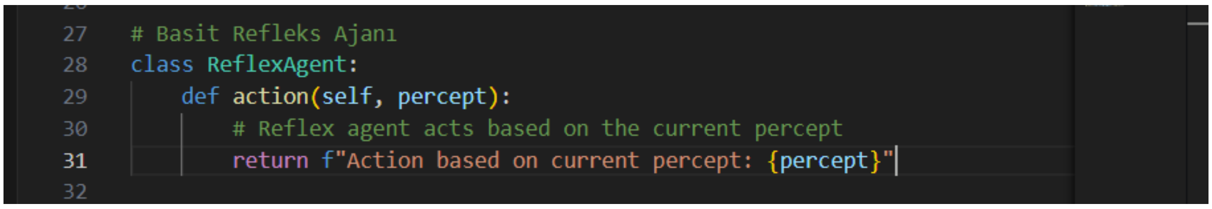
Expected Output:



```
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python
/Python312/python.exe c:/Users/win10/Desktop/python_kodlari/Kulubecioglu_Mehmet_lab1.py
Performance: Explore deep ocean areas and collect data
Environment: Deep ocean, varying temperatures, high pressure
Actuators: ['propeller', 'arms for sample collection', 'lights']
Sensors: ['sonar', 'temperature sensor', 'pressure gauge', 'camera']
```

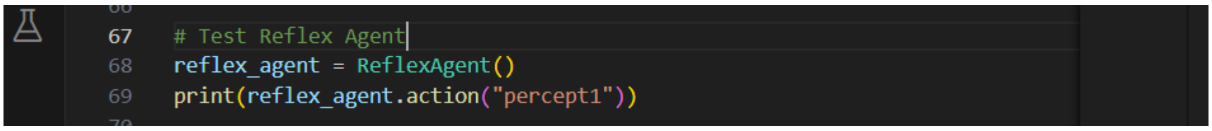
Step 2: Implement the Reflex Agent

A **Reflex Agent** makes decisions based solely on the current percept (input) without considering previous experiences. I implement this by creating the `ReflexAgent` class, which has an `action()` method that responds directly to a percept.



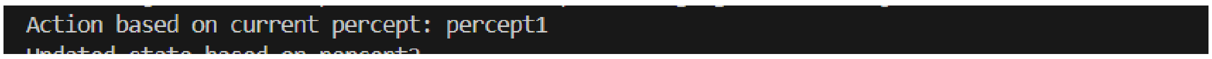
```python
# Basit Refleks Ajanı
class ReflexAgent:
    def action(self, percept):
        # Reflex agent acts based on the current percept
        return f"Action based on current percept: {percept}"
```

Test Reflex Agent:



```python
# Test Reflex Agent
reflex_agent = ReflexAgent()
print(reflex_agent.action("percept1"))
```

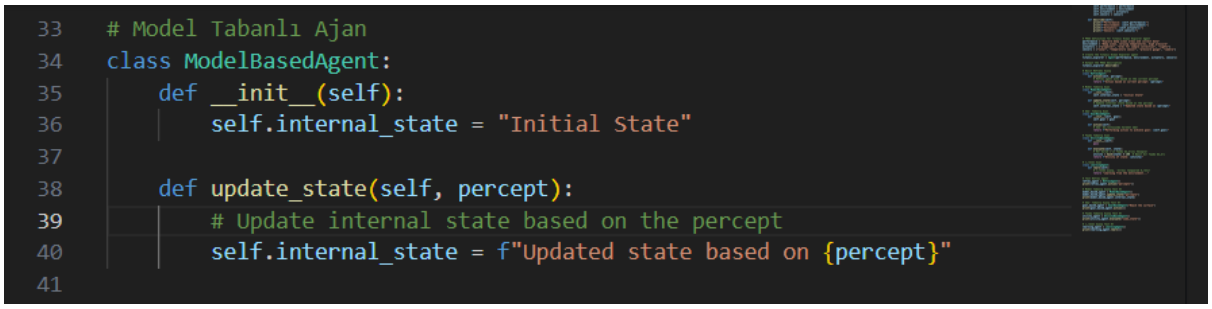
Expected Output:



```
Action based on current percept: percept1
```

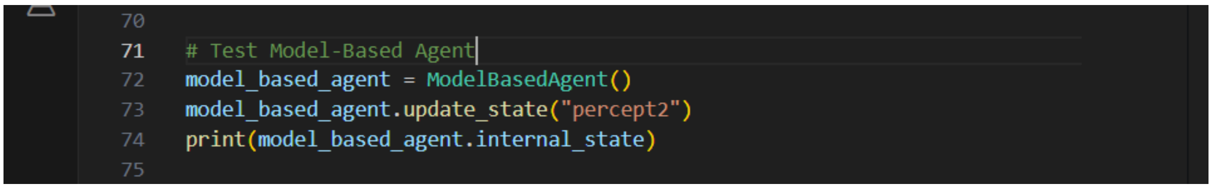
Step 3: Implement the Model-Based Agent

A **Model-Based Agent** maintains an internal state that represents the world. The agent updates this internal state based on percepts received over time. I create a `ModelBasedAgent` class with an `update_state()` method that modifies the internal state based on the percept.



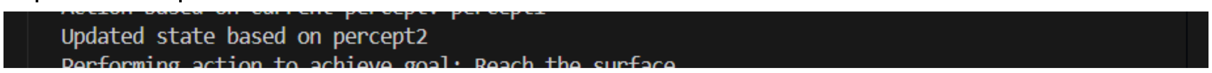
```python
# Model Tabanlı Ajan
class ModelBasedAgent:
    def __init__(self):
        self.internal_state = "Initial State"

    def update_state(self, percept):
        # Update internal state based on the percept
        self.internal_state = f"Updated state based on {percept}"

```

Test Model-Based Agent:



```python

# Test Model-Based Agent
model_based_agent = ModelBasedAgent()
model_based_agent.update_state("percept2")
print(model_based_agent.internal_state)

```

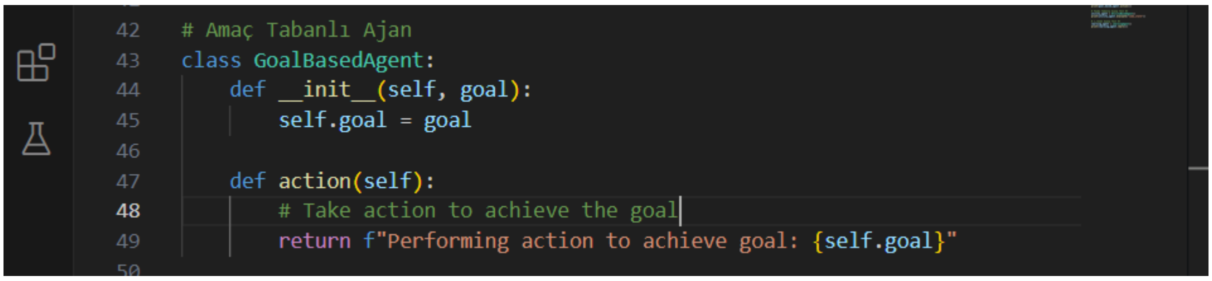
Expected Output:



```
Action based on current percept: percept1
Updated state based on percept2
Performing action to achieve goal: Reach the surface
```

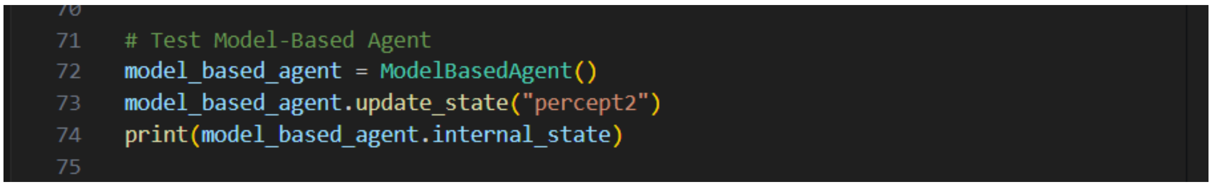
Step 4: Implement the Goal-Based Agent

A **Goal-Based Agent** not only uses percepts but also operates with a specific goal in mind. The agent chooses actions that will help achieve its goal. I implement this by creating the `GoalBasedAgent` class with an `action()` method that returns an action aligned with the agent's goal.



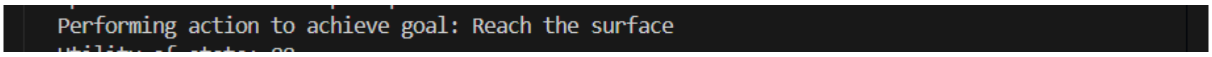
```python
# Amaç Tabanlı Ajan
class GoalBasedAgent:
    def __init__(self, goal):
        self.goal = goal

    def action(self):
        # Take action to achieve the goal
        return f"Performing action to achieve goal: {self.goal}"

```

Test Goal-Based Agent:



```python
# Test Model-Based Agent
model_based_agent = ModelBasedAgent()
model_based_agent.update_state("percept2")
print(model_based_agent.internal_state)
```

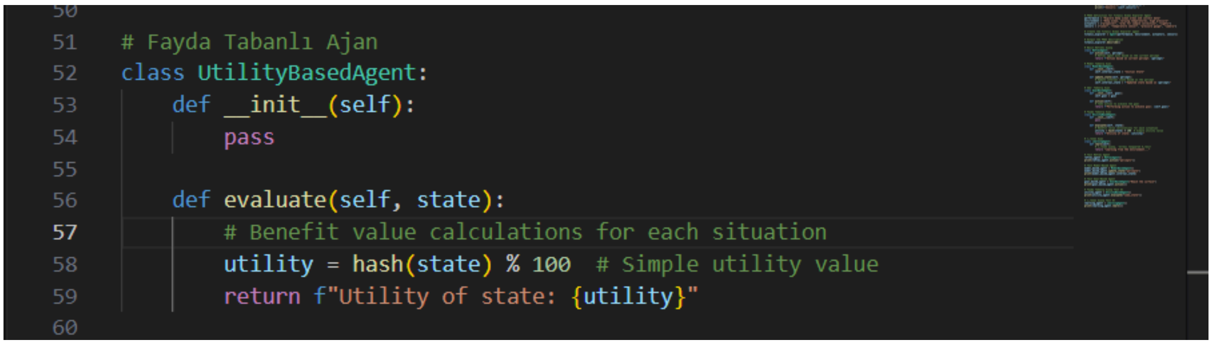
Expected Output:



```
Performing action to achieve goal: Reach the surface
```

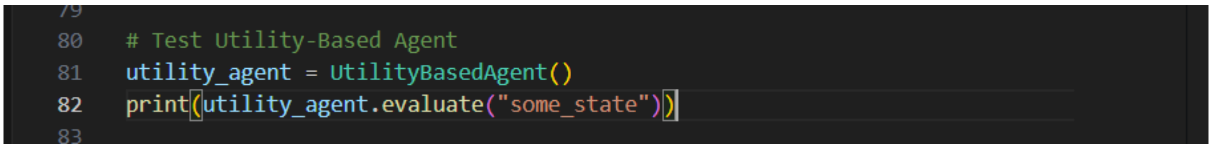
Step 5: Implement the Utility-Based Agent

A **Utility-Based Agent** evaluates different states of the environment and selects actions that maximize its utility. I create a `UtilityBasedAgent` class with an `evaluate()` method that calculates a utility score for a given state.



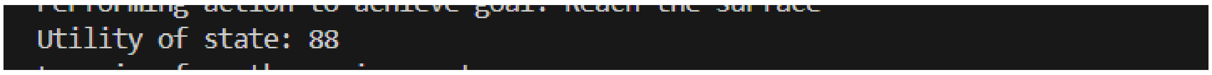
```python
# Fayda Tabanlı Ajan
class UtilityBasedAgent:
    def __init__(self):
        pass

    def evaluate(self, state):
        # Benefit value calculations for each situation
        utility = hash(state) % 100  # Simple utility value
        return f"Utility of state: {utility}"
```

Test Utility-Based Agent:



```python
# Test Utility-Based Agent
utility_agent = UtilityBasedAgent()
print(utility_agent.evaluate("some_state"))
```

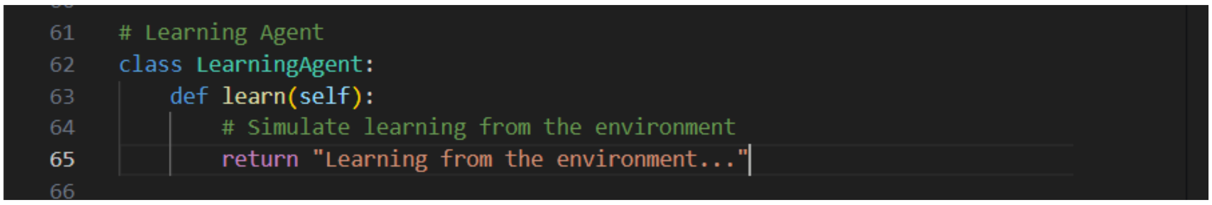
Expected Output:



```
Utility of state: 88
```

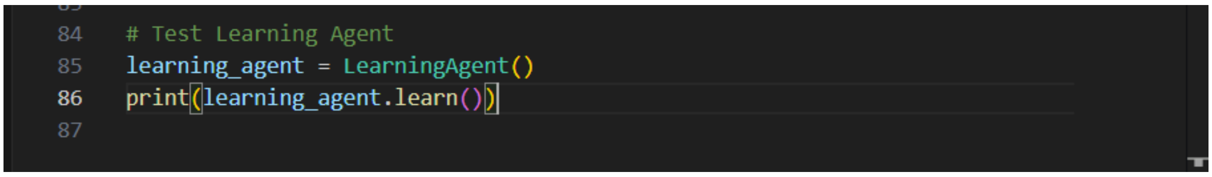
Step 6: Implement the Learning Agent

A **Learning Agent** improves its performance based on past experiences and adjusts to its environment. I implement this by creating the `LearningAgent` class with a `learn()` method that simulates the agent learning from its environment.



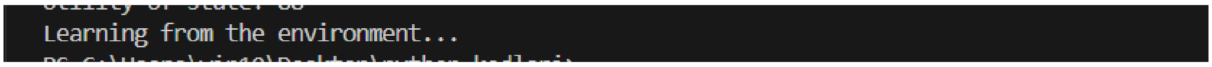
```python
# Learning Agent
class LearningAgent:
    def learn(self):
        # Simulate learning from the environment
        return "Learning from the environment..."
```

Test Learning Agent:



```python
# Test Learning Agent
learning_agent = LearningAgent()
print(learning_agent.learn())
```

Expected Output:



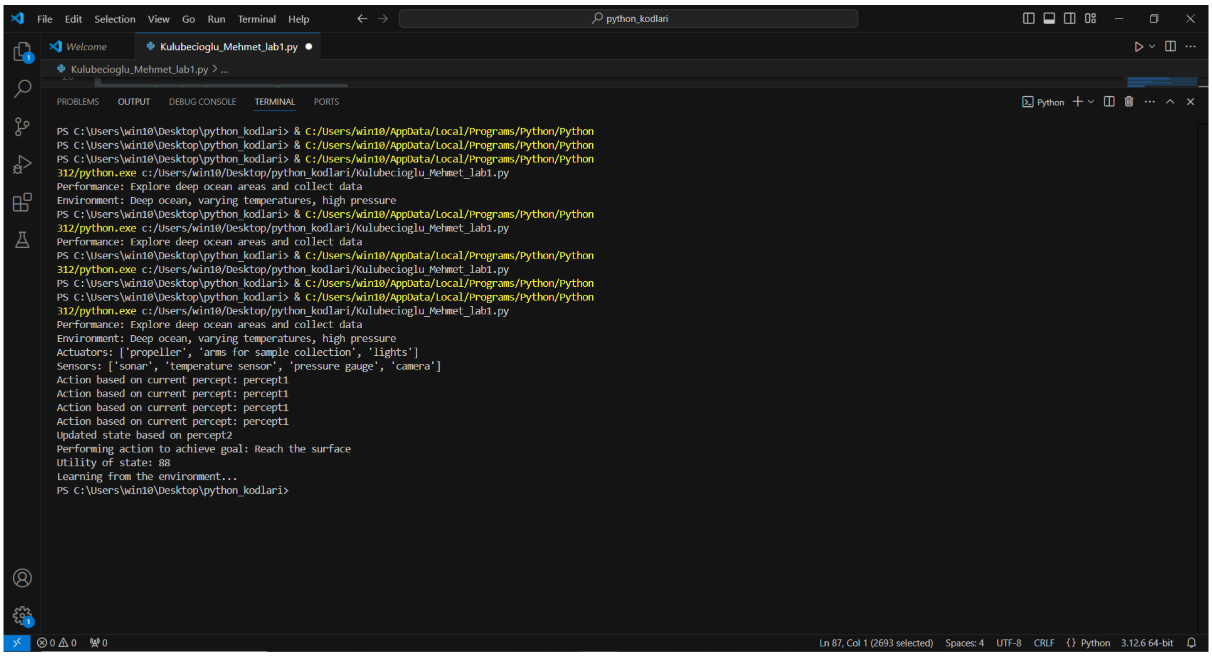
```
Learning from the environment...
```

MY FULL CODE:

```python
class Agent:
    def __init__(self, performance, environment, actuators, sensors):
        self.performance = performance
        self.environment = environment
        self.actuators = actuators
        self.sensors = sensors

    def describe(self):
        print(f"Performance: {self.performance}")
        print(f"Environment: {self.environment}")
        print(f"Actuators: {self.actuators}")
        print(f"Sensors: {self.sensors}")


# PEAS definition for Titanic Ocean Explorer Agent
performance = "Explore deep ocean areas and collect data"
environment = "Deep ocean, varying temperatures, high pressure"
actuators = ["propeller", "arms for sample collection", "lights"]
sensors = ["sonar", "temperature sensor", "pressure gauge", "camera"]

# Create the Titanic Ocean Explorer agent
```

```python
titanic_explorer = Agent(performance, environment, actuators, sensors)

# Output the PEAS description
titanic_explorer.describe()

# Basit Refleks Ajanı
class ReflexAgent:
    def action(self, percept):
        # Reflex agent acts based on the current percept
        return f"Action based on current percept: {percept}"

# Model Tabanlı Ajan
class ModelBasedAgent:
    def __init__(self):
        self.internal_state = "Initial State"

    def update_state(self, percept):
        # Update internal state based on the percept
        self.internal_state = f"Updated state based on {percept}"

# Amaç Tabanlı Ajan
class GoalBasedAgent:
    def __init__(self, goal):
        self.goal = goal

    def action(self):
        # Take action to achieve the goal
        return f"Performing action to achieve goal: {self.goal}"

# Fayda Tabanlı Ajan
class UtilityBasedAgent:
    def __init__(self):
        pass

    def evaluate(self, state):
        # Benefit value calculations for each situation
        utility = hash(state) % 100  # Simple utility value
        return f"Utility of state: {utility}"

# Learning Agent
class LearningAgent:
    def learn(self):
        # Simulate learning from the environment
        return "Learning from the environment..."

# Test Reflex Agent
reflex_agent = ReflexAgent()
print(reflex_agent.action("percept1"))

# Test Model-Based Agent
model_based_agent = ModelBasedAgent()
model_based_agent.update_state("percept2")
print(model_based_agent.internal_state)

# Test Goal-Based Agent
goal_based_agent = GoalBasedAgent("Reach the surface")
print(goal_based_agent.action())

# Test Utility-Based Agent
utility_agent = UtilityBasedAgent()
print(utility_agent.evaluate("some_state"))

# Test Learning Agent
learning_agent = LearningAgent()
print(learning_agent.learn())
```

MY FULL OUTPUTS:


File Edit Selection View Go Run Terminal Help
python_kodlari
Welcome
Kulubecioglu_Mehmet_lab1.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
Ln 87, Col 1 (2693 selected) Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit

```
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
312/python.exe c:/Users/win10/Desktop/python_kodlari/Kulubecioglu_Mehmet_lab1.py
Performance: Explore deep ocean areas and collect data
Environment: Deep ocean, varying temperatures, high pressure
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
312/python.exe c:/Users/win10/Desktop/python_kodlari/Kulubecioglu_Mehmet_lab1.py
Performance: Explore deep ocean areas and collect data
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
312/python.exe c:/Users/win10/Desktop/python_kodlari/Kulubecioglu_Mehmet_lab1.py
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
PS C:\Users\win10\Desktop\python_kodlari> & C:/Users/win10/AppData/Local/Programs/Python/Python
312/python.exe c:/Users/win10/Desktop/python_kodlari/Kulubecioglu_Mehmet_lab1.py
Performance: Explore deep ocean areas and collect data
Environment: Deep ocean, varying temperatures, high pressure
Actuators: ['propeller', 'arms for sample collection', 'lights']
Sensors: ['sonar', 'temperature sensor', 'pressure gauge', 'camera']
Action based on current percept: percept1
Action based on current percept: percept1
Action based on current percept: percept1
Action based on current percept: percept1
Updated state based on percept2
Performing action to achieve goal: Reach the surface
Utility of state: 88
Learning from the environment...
PS C:\Users\win10\Desktop\python_kodlari>
```

Conclusion

In this lab, I implemented five different types of agents (Reflex, Model-Based, Goal-Based, Utility-Based, and Learning agents) and defined the PEAS framework for a Titanic Ocean Explorer agent. Each agent was designed to solve specific problems and interact with its environment in different ways. The outputs demonstrate the successful functioning of each agent type.

Tasks and exercises

1. To give an environment description for the following agents:

- **Titanic ocean explorer robot:** The Titanic ocean explorer robot is designed to explore deep ocean areas, so its performance goal is to collect data from deep underwater. Its environment is the deep ocean, where it encounters varying temperatures, high pressure, and possibly unpredictable underwater terrains. The actuators it would use are propellers for movement, mechanical arms for sample collection, and lights for visibility. Its sensors would include sonar for navigation, a temperature sensor, a pressure gauge, and cameras for visual data.
- **PEAS:**

    - **Performance:** Collect deep ocean data

    - **Environment:** Deep ocean, varying temperatures, high pressure

    - **Actuators:** Propeller, arms for sample collection, lights

    - **Sensors:** Sonar, temperature sensor, pressure gauge, camera

- **Playing soccer:** For an agent playing soccer, the environment is the soccer field with opponents, teammates, and the ball. The agent's performance is determined by its ability to move the ball towards the goal, pass to teammates, and avoid opponents. It uses its legs or other mechanical parts to kick the ball and possibly sensors like cameras or motion detectors to perceive the ball's location and the other players.

    - **PEAS:**

        - **Performance:** Score goals, defend, pass the ball

        - **Environment:** Soccer field with teammates, opponents, ball

        - **Actuators:** Legs (or mechanical parts for movement), kicking mechanism

        - **Sensors:** Cameras, motion detectors

- **Playing tennis:** A tennis-playing agent needs to return the ball successfully and position itself correctly on the court. The environment is the tennis court, with varying ball speeds and angles, and an opponent. The agent would have a racquet arm for hitting the ball, and sensors to detect the ball's speed, direction, and position on the court.

- **PEAS:**

    - **Performance:** Return ball, aim for opponent's weak spots

    - **Environment:** Tennis court, opponent, varying ball speeds and angles

    - **Actuators:** Racquet arm

    - **Sensors:** Ball-tracking sensors, court position sensors

- Knitting a sweater: A knitting agent operates in an environment where it must follow a knitting pattern. Its performance goal is to produce a correctly knit sweater. It has actuators in the form of mechanical hands or needles, and sensors that track the thread's tension, needle position, and the pattern.

- **PEAS:**

    - **Performance:** Knit a sweater following a pattern

    - **Environment:** Workspace, yarn, needles

    - **Actuators:** Mechanical hands, knitting needles

    - **Sensors:** Tension sensors, pattern recognition sensors

- Medical diagnostic system: A medical diagnostic system operates in a hospital or clinical environment. Its performance goal is to diagnose diseases based on patient data. It uses actuators to interact with databases or diagnostic tools, and sensors might include patient data inputs (e.g., test results, patient symptoms) through electronic health records.

- **PEAS:**

    - **Performance:** Diagnose diseases accurately

    - **Environment:** Hospital, patient records, test results

    - **Actuators:** Software interactions with databases, diagnostic tools

    - **Sensors:** Input from patient records, test data

2. To give a description of the environment type for the agents from Exercise 1:

- **Titanic ocean explorer robot:**
  The environment is **partially observable** (as sensors provide only limited information about the surroundings), **stochastic** (unpredictable ocean conditions), **sequential** (actions taken now affect future outcomes), **dynamic** (the environment changes independently of the robot), **continuous** (smooth changes in depth and pressure), and **single-agent** (only one agent operating).
- **Playing soccer:**
  The environment is **partially observable** (not all player movements can be perceived), **stochastic** (the outcome of movements is influenced by other players), **sequential**, **dynamic**, **continuous**, and **multi-agent** (with multiple players).
- **Playing tennis:**
  The environment is **fully observable** (the ball and opponent's movements are visible), **stochastic** (the opponent's behavior is not fully predictable), **sequential**, **dynamic**, **continuous**, and **multi-agent**.
- **Knitting a sweater:**
  The environment is **fully observable** (the knitting pattern and materials are visible), **deterministic** (knitting follows a fixed pattern), **sequential**, **static** (the knitting process itself doesn't change unless interrupted), **discrete** (knitting involves specific steps), and **single-agent**.
- **Medical diagnostic system:**
  The environment is **partially observable** (not all aspects of patient health are visible), **deterministic** (based on data provided), **sequential**, **semi-dynamic** (the diagnosis remains stable, though a patient's condition might change), **discrete** (specific data inputs), and **single-agent**.

3. Give examples for each type of agent:

- **Simple reflex agent:**
  A thermostat that turns on the heater when the temperature falls below a certain level.
- **Model-based agent:**
  A robotic vacuum cleaner that maps the layout of the house and adjusts its path accordingly.
- **Goal-based agent:**
  A self-driving car that navigates towards a set destination using GPS and real-time traffic data.
- **Utility-based agent:**
  A financial trading algorithm that evaluates different investments based on predicted returns and risk.
- **Learning agent:**
  A recommendation system, such as Netflix, which improves its movie recommendations as it learns from user preferences.

4. Is it true that a rational agent needs to know all the consequences of its own actions?

No, a rational agent does not necessarily need to know all the consequences of its actions in advance. Rational agents make decisions based on the information they currently have, and they aim to achieve the best possible outcome given the uncertainty in the environment. In many real-world scenarios, it is impossible for an agent to predict every consequence, especially in stochastic or partially observable environments. Therefore, they often work based on probabilities and expected outcomes.

5. Why do we need a description of the PEAS environment?

Describing the PEAS environment is essential because it helps in understanding the requirements and constraints of the agent's task. PEAS outlines the agent's goals, the environment in which it operates, the tools it uses to interact with that environment, and the information it can perceive. This detailed description provides a clear framework for designing an agent, ensuring that it is well-suited to perform its intended function. Without a clear PEAS model, it would be difficult to determine how an agent should behave and what mechanisms are needed to accomplish its tasks effectively.