National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2023, Lab Manual – 03

Course Code: AI-2002	Course: Artificial Intelligence Lab
Instructor(s):	Kariz Kamal, Saeeda Kanwal, Sania Urooj, Shakir Hussain, Omer Qureshi, Muhammad Ali Fatmi

Contents:

Objective

1. Introduction to Agents in different Environments, exposing students to different AI Problems and solutions.
2. Types of Agents and the reasons to use them in different environments.
3. Solve some basic AI problem using the python programming language.

Introduction to Agents

An intelligent agent (IA) is an entity that makes a decision that enables artificial intelligence to be put into action. It can also be described as a software entity that conducts operations in the place of users or programs after sensing the environment. It uses actuators to initiate action in that environment.

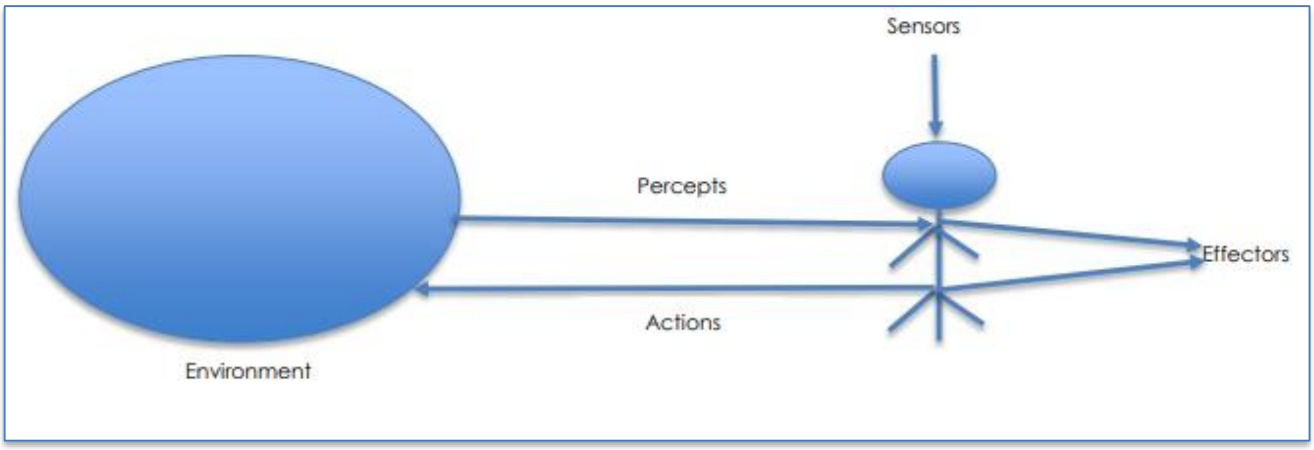
- This agent has some level of autonomy that allows it to perform specific, predictable, and repetitive tasks for users or applications.
- It's also termed as 'intelligent' because of its ability to learn during the process of performing tasks.
- The two main functions of intelligent agents include perception and action. Perception is done through sensors while actions are initiated through actuators.
- Intelligent agents consist of sub-agents that form a hierarchical structure. Lower-level tasks are performed by these sub-agents.
- The higher-level agents and lower-level agents form a complete system that can solve difficult problems through intelligent behaviors or responses.

How intelligent agents work

Intelligent agents work through three main components: sensors, actuators, and effectors. Getting an overview of these components can improve our understanding of how intelligent agents work.

- **Sensors:** These are devices that detect any changes in the environment. This information is sent to other devices. In artificial intelligence, the environment of the system is observed by intelligent agents through sensors.

- **Actuators:** These are components through which energy is converted into motion. They perform the role of controlling and moving a system. Examples include rails, motors, and gears.
- **Effectors:** The environment is affected by effectors. Examples include legs, fingers, wheels, display screen, and arms.



Example of Agents:

- A **software agent** has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.
- A **Human-agent** has eyes, ears, and other organs which act as sensors, and hands, legs, mouth, and other body parts acting as actuators.
- A **Robotic agent** has Cameras and infrared range finders which act as sensors and various motors acting as actuators.

Types of Agents:

The agents are mainly classified into five types

Simple Reflex Agent:

A **simple reflex agent** is the most basic of the intelligent agents out there. It performs actions based on a current situation. When something happens in the environment of a simple reflex agent, the agent quickly scans its knowledge base for how to respond to the situation at-hand based on pre-determined rules.

It would be like a home thermostat recognizing that if the temperature increases to 75 degrees in the house, the thermostat is prompted to kick on. It doesn't need to know what happened with the temperature yesterday or what might happen tomorrow. Instead, it operates based on the idea that if it happens, here is the response.

Simple reflex agents are just that: simple. They cannot compute complex equations or solve complicated problems. They work only in environments that are fully-observable in the current percept, ignoring any percept history. If you have a smart light bulb, for example, set to turn on at 6 p.m. every night, the light bulb will not recognize how the days are longer in summer and the lamp is not needed until much later. It will continue to turn the lamp on at 6 p.m. because that is the rule it follows. Simple reflex agents are built on the condition-action rule.

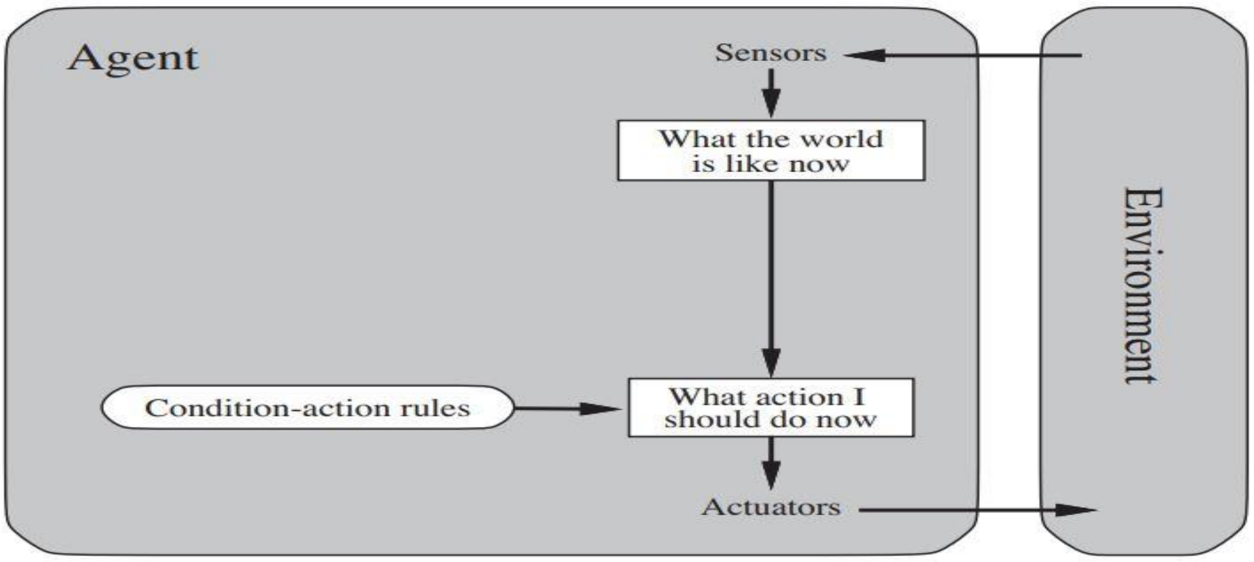


Figure 2. Inputs (percepts) from the environment are received by the intelligent agent through sensors. This agent uses artificial intelligence to make decisions using the acquired information/ observations. Actions are then triggered through actuators. Future decisions will be influenced by percept history and past actions.

To create our agent, we need two functions: INTERPRET-INPUT function, which generates an abstracted description of the current state from the percept and the RULE-MATCH function, which returns the first rule in the set of rules that matches the given state description.

```
from aima3.agents import *
from aima3.notebook import psource


loc_A = (0, 0)
loc_B = (1, 0)


"""We change the simpleReflexAgentProgram so that it doesn't make use of the Rule class"" "
def SimpleReflexAgentProgram():
"""This agent takes action based solely on the percept. [Figure 2.10]"""
def program(percept): loc, status = percept
return ('Suck' if status == 'Dirty' else'Right' if loc == loc_A
else'Left')
return program


# Create a simple reflex agent the two-state environment program =
SimpleReflexAgentProgram() simple_reflex_agent = Agent(program)
```

Now add the agent to the environment:

```
# These are the two locations for the two-state environment
loc_A, loc_B = (0, 0), (1, 0)

# Initialize the two-state environment
trivial_vacuum_env = TrivialVacuumEnvironment()

# Check the initial state of the environment
print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

```
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Dirty'}.
```

```
trivial_vacuum_env.add_thing(simple_reflex_agent)

int("SimpleReflexVacuumAgent is located at {}.".format(simple_reflex_agent.location))
```

```
SimpleReflexVacuumAgent is located at (0, 0).
```

```
for x in range(3):
  # Run the environment
  trivial_vacuum_env.step()

  # Check the current state of the environment
   print("State of the Environment: {}.".format(trivial_vacuum_env.status))

   print("SimpleReflexVacuumAgent is located at {}.".format(simple_reflex_agent.location )
  )
```

```
for x in range(3):
  # Run the environment
  trivial_vacuum_env.step()

  # Check the current state of the environment
   print("State of the Environment: {}.".format(trivial_vacuum_env.status))

   print("SimpleReflexVacuumAgent is located at   {}.".format(simple_reflex_agent.location )
  )
```

```
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Dirty'}.
SimpleReflexVacuumAgent is located at (1, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
SimpleReflexVacuumAgent is located at (1, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
SimpleReflexVacuumAgent is located at (0, 0).
```

Model Based Reflex Agents:

A **model-based reflex agent** is one that uses its percept history and its internal memory to make decisions about an internal "model" of the world around it. Internal memory allows these agents to store some of their navigation history, and then use that semi-subjective history to help understand things about their current environment--even when everything they need to know cannot be directly observed. What's a percept you ask? A **percept** is a lasting result of something we have perceived, which, rather than being immediately perceived, is something we know we could possibly perceive.

With Waymo, for example, the model-based agent uses GPS to understand its location and predict upcoming drivers. You and I take for granted that, when the brake lights of the car ahead of us come on, the driver has hit the brakes and so the car in front of us is going to slow down. But there's no reason to associate a red light with the deceleration of a vehicle, unless you are used to seeing those two things happen at the same time. So the Waymo can learn that it needs to hit the brakes by drawing on its perceptual history. Waymo can learn to associate red brake lights just ahead with the need to slow itself down. Another common task is when the Waymo car decides it needs to change lanes. Should it just shimmy over as though it's the only car in the world? No, it uses the same processes to estimate whether any other cars might be in the path of its intended lane change, to avoid causing a collision.

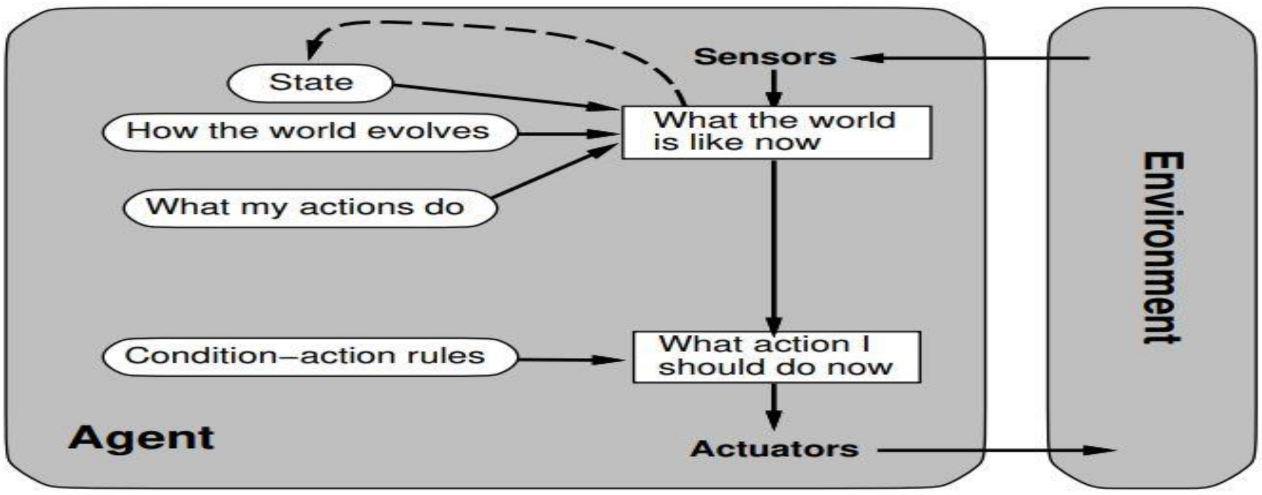


Figure 3. Inputs (percepts) from the environment are received by the intelligent agent through sensors. This agent uses artificial intelligence based on history of the actions and analyze which action to take place. Actions are then triggered through actuators. Future decisions will be influenced by percept history and past actions.

We will now create a model-based reflex agent for the environment:

```
 # Delete the previously added simple reflex agent
trivial_vacuum_env.delete_thing(simple_reflex_agent)
```

We need another function UPDATE-STATE which will be responsible for creating a new state description

```
  # Delete the previously added simple reflex agent
 trivial_vacuum_env.delete_thing(simple_reflex_agent)
  Check the initial state of the environment
  print("State of the Environment: {}.".format(trivial_vacuum_env.status))
# Check the initial state of the environment
  print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Dirty'}.

```
# TODO: Implement this function for the two-dimensional environment
def update_state(state, action, percept, model): pass
# Create a model-based reflex agent
model_based_reflex_agent = ModelBasedVacuumAgent()
# Add the agent to the environment
trivial_vacuum_env.add_thing(model_based_reflex_agent)
    print("ModelBasedVacuumAgent is located at {}.".format(model_based_reflex_agent.location)
    )
```

Model Based Vacuum Agent is located at (1, 0).

```
 #    for x in range(3):
 # Run the environment
       trivial_vacuum_env.step()

 # Check the current state of the environment
         print("State of the Environment: {}.".format(trivial_vacuum_env.status))
```

```
print("ModelBasedVacuumAgent is located at {}.".format(model_based_reflex_agent.locat ion))
ModelBasedVacuumAgent is located at (1, 0).
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
ModelBasedVacuumAgent is located at (0, 0).
State of the Environment: {(0, 0): 'Dirty', (1, 0): 'Clean'}.
ModelBasedVacuumAgent is located at (0, 0).
State of the Environment: {(0, 0): 'Clean', (1, 0): 'Clean'}.
```

Goal-Based Agents:

A **goal-based agent** has an agenda, you might say. It operates based on a goal in front of it and makes decisions based on how best to reach that goal. Unlike a simple reflex agent that makes decisions based solely on the current environment, a goal-based agent is capable of thinking beyond the present moment to decide the best actions to take in order to achieve its goal. In this regard, a goal-based agent operates as a **search and planning** function, meaning it targets the goal ahead and finds the right action in order to reach it. This helps a goal-based agent to be proactive rather than simply reactive in its decision-making.

You may take a goal-based approach to tasks at work. For example, you might set a goal for yourself to become a more efficient typist, which will help you in completing assignments more quickly. A step toward that goal, then, might be to enroll in a typing course or to devote 15 minutes a day to practice in order to increase your word count per minute. Your decisions are flexible, a hallmark of goal-based agents, but the focus is always on achieving the goal ahead.

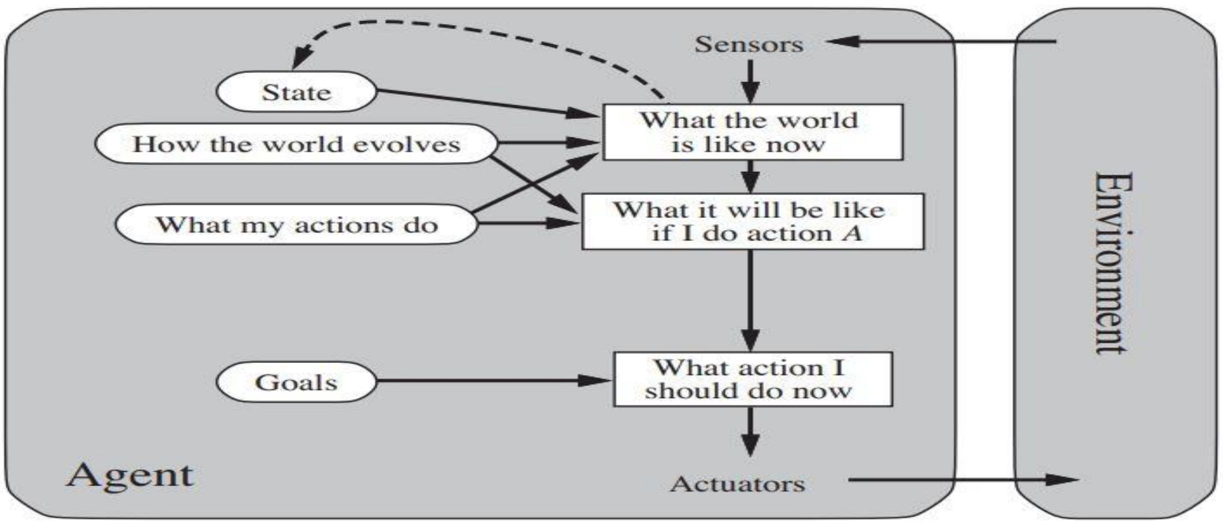

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Figure 4. Inputs (percepts) from the environment are received by the intelligent agent through sensors. This agent uses artificial intelligence based to analyze how to achieve the given goal. Actions are then triggered through actuators. Future decisions will be influenced by percept history and past actions.

```
# Delete the previously added simple reflex agent
 trivial_vacuum_env.delete_thing(simple_reflex_agent)
#Write a python code for Goal based Agent program.
```

Utility-Based Agents:

The agents which are developed having their end uses as building blocks are called utility-based agents. When there are multiple possible alternatives, then to decide which one is best, utility-based agents are used. They choose actions based on a **preference (utility)** for each state. Sometimes achieving the desired goal is not enough. We may look for a quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how **"happy"** the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.

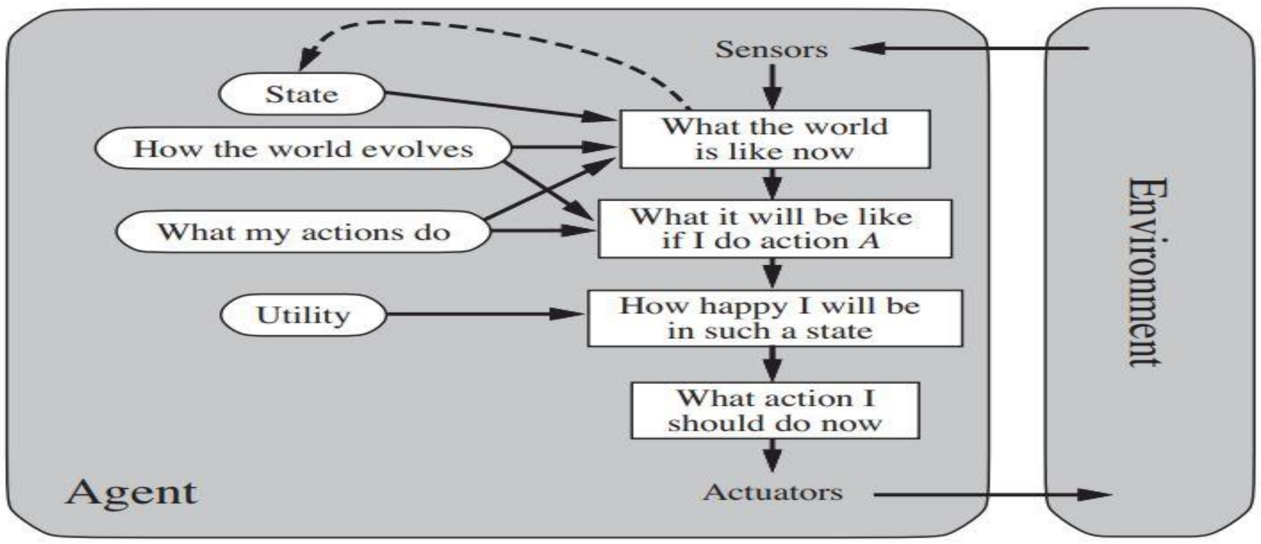

Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure 4. Inputs (percepts) from the environment are received by the intelligent agent through sensors. This agent uses artificial intelligence based to analyze which option is the best to choose. Actions are then triggered through actuators. Future decisions will be influenced by percept history and past actions.

```
 # Delete the previously added simple reflex agent
trivial_vacuum_env.delete_thing(simple_reflex_agent)
#Write a python code for Utility based Agent program.
```

Learning Agents:

When we expand our environments, we get a larger and larger amount of tasks, eventually we are going to have a very large number of actions to pre-define. Another way of going about creating an agent is to get it to learn new actions as it goes about its business, this still requires some initial knowledge but cuts down on the programming greatly. This will allow the agent to work in environments that are unknown.

A learning agent can be split into the 4 parts shown in the diagram.

The learning element is responsible for improvements this can make a change to any of the knowledge components in the agents. One way of learning is to observe pairs of successive states in the percept sequence; from this the agent can learn how the world evolves. For utility based agents an external performance standard is needed to tell the critic if the agent’s action has a good or a bad effect on the world.

The performance element is responsible for selecting external actions, and this is considered to be the previous agents discussed.

The learning agent gains feedback from the critic on how well the agent is doing and determines how the performance element should be modified if at all to improve the agent.

For example when you were in school you would do a test and it would be marked the test is the critic. The teacher would mark the test and see what could be improved and instructs you how to do better next time, the teacher is the learning element and you are the performance element.
The last component is the problem generator, the performance generator only suggests actions that it can already do so we need a way of getting the agent to experience new situations, and this is what the performance generator is for. This way the agent keeps on learning.

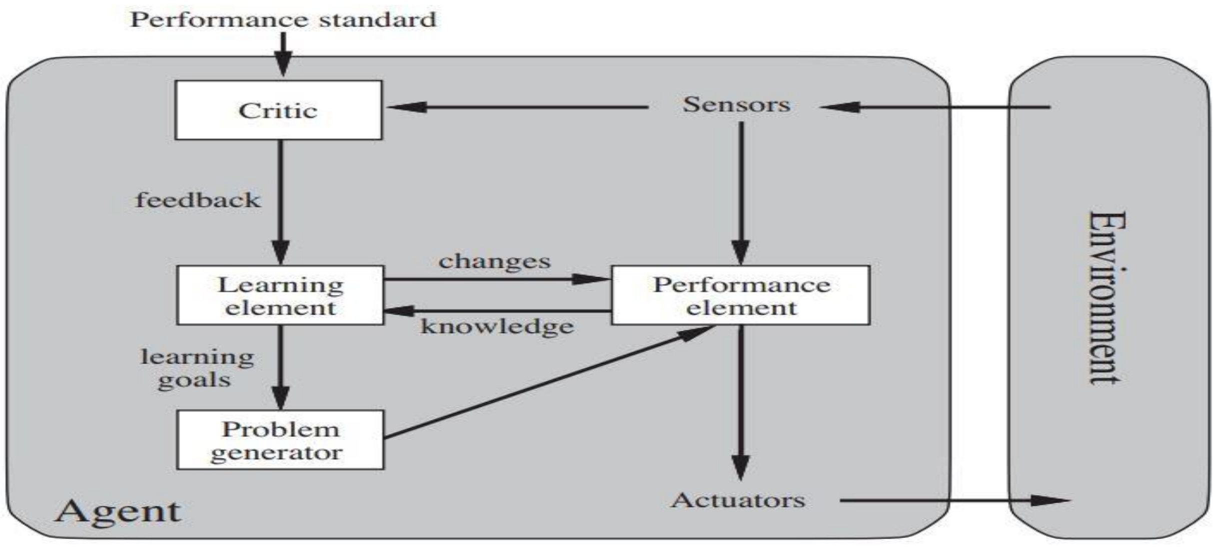


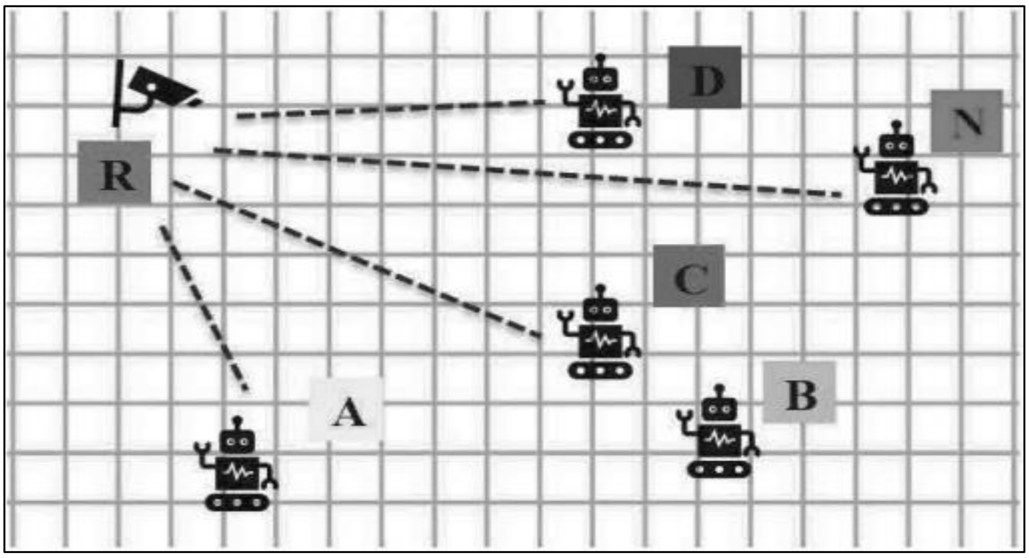
Figure 3. Features of Python: Python is one of the most dynamic and versatile programming languages available in the industry today.

```
 # Delete the previously added simple reflex agent
trivial_vacuum_env.delete_thing(simple_reflex_agent)
#Write a python code for Learning based Agent program.
```

TASKS

Task#01: Simple Reflex Agents

Consider an interactive and cognitive environment (ICE) in which a smart camera is monitoring robot movement from one location to another. Let a robot be at location A for some time instant and then moves to point B and eventually reaches at point C and so on and so forth shown in the Fig. Develop a Python code to calculate a distance between reference point R (4, 0) of a camera and A, B, and C and N number of locations.



Task#02 Simple Reflex Agents

Consider a scenario, cameras placed on every side of the car — front, rear, left and right — to stitch together a 360-degree view of the environment. For a three-lane road a car is moving on a middle lane, consider the below scenario

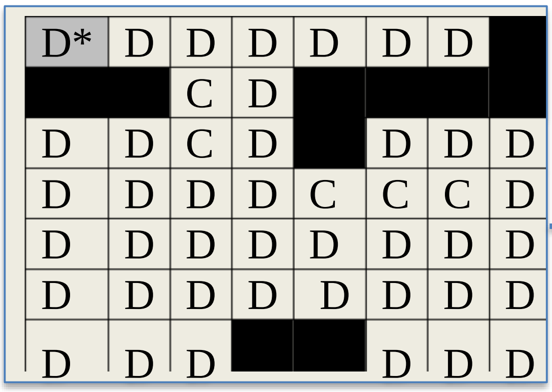
- If the front camera detects the object within range of 8 meters breaks are applied automatically.
- If the left camera detects the object within range of 2 meters car moves to the right lane.
- If the right camera detects the object within range of 2 meters car moves to the left lane.
- For parking the car if the rear camera detects the object within 5 cm breaks are applied.

Task#03 Simple Reflex Agents

Consider the following scenario where the UAV receives temperature data from the installed sensors in a residential area. Assume that there are nine sensors installed that are measuring temperature in centigrade. Develop a Python code to calculate the average temperature in F.

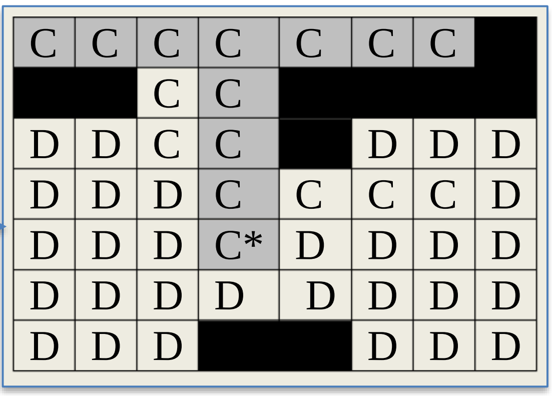
Task#04 Model Based Reflex Agents

An AI startup has approached you to write a program for their automatic vacuum cleaner. For the vacuum cleaner the room appears to be a matrix of n * m. Each index referred to as a cell of the matrix is valued as dirty "D", clean "C". The cells which are occupied by the stuff in the room are blocked and valued "B". The vacuum can move in all four directions (up, down, left, right), and if the cell status is D, it will clean the cell and change the status to "C", if the cell status is either C, it will not enter the cell. The vacuum will stop working if the surrounding of its positions is cleaned (Up, Left, Right, Down), i.e., the status of all the cells is either C. The vacuum may start cleaning the room from the first cell (0, 0) or any random location. You will trace the path of the vacuum and display at the end of the program.



D*	D	D	D	D	D	D	■
■	■	C	D	■	■	■	■
D	D	C	D	■	D	D	D
D	D	D	D	C	C	C	D
D	D	D	D	D	D	D	D
D	D	D	D	D	D	D	D
D	D	D	■	■	D	D	D

Initial State: Agent (*) at location (0,0)

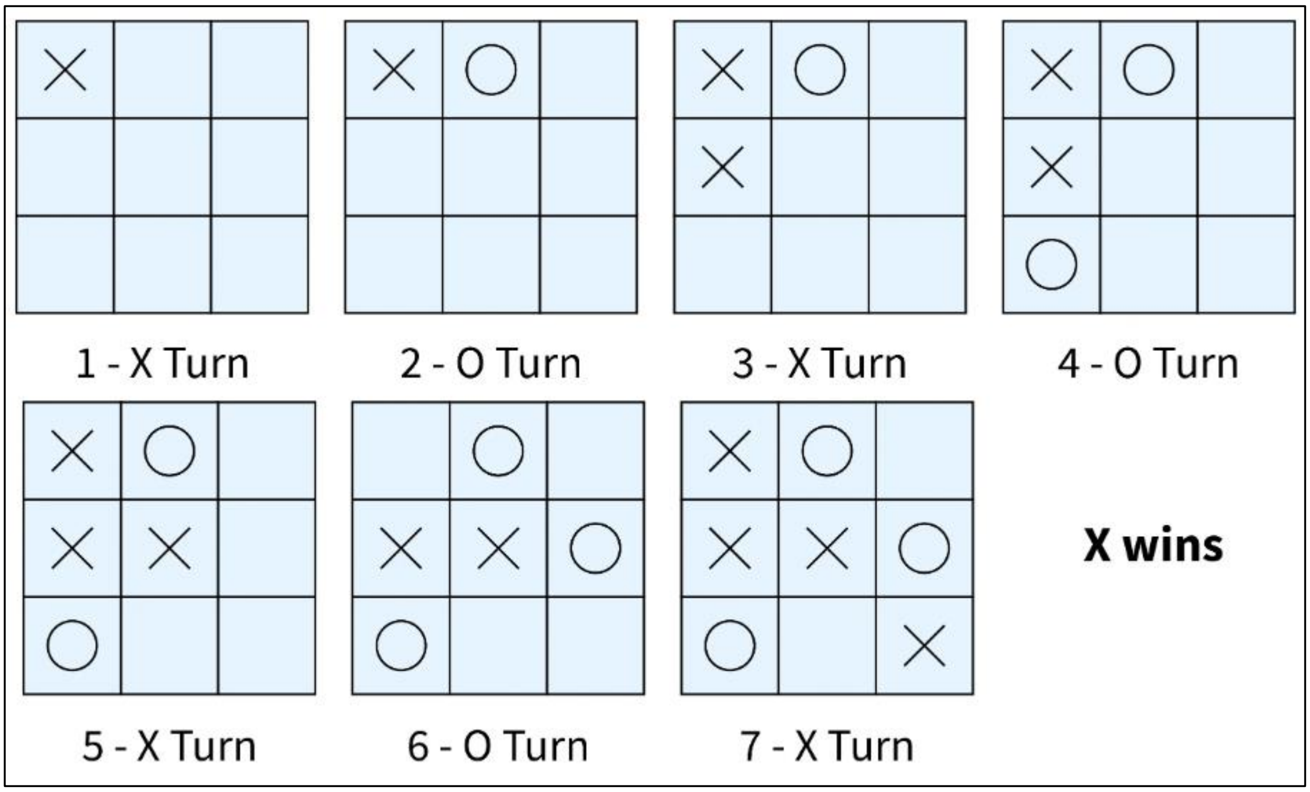


C	C	C	C	C	C	C	■
■	■	C	C	■	■	■	■
D	D	C	C	■	D	D	D
D	D	D	C	C	C	C	D
D	D	D	C*	D	D	D	D
D	D	D	D	D	D	D	D
D	D	D	■	■	D	D	D

Transition State: after 10 steps
Agent (*) at location (3,4)

Task#05 Goal-Based Agents

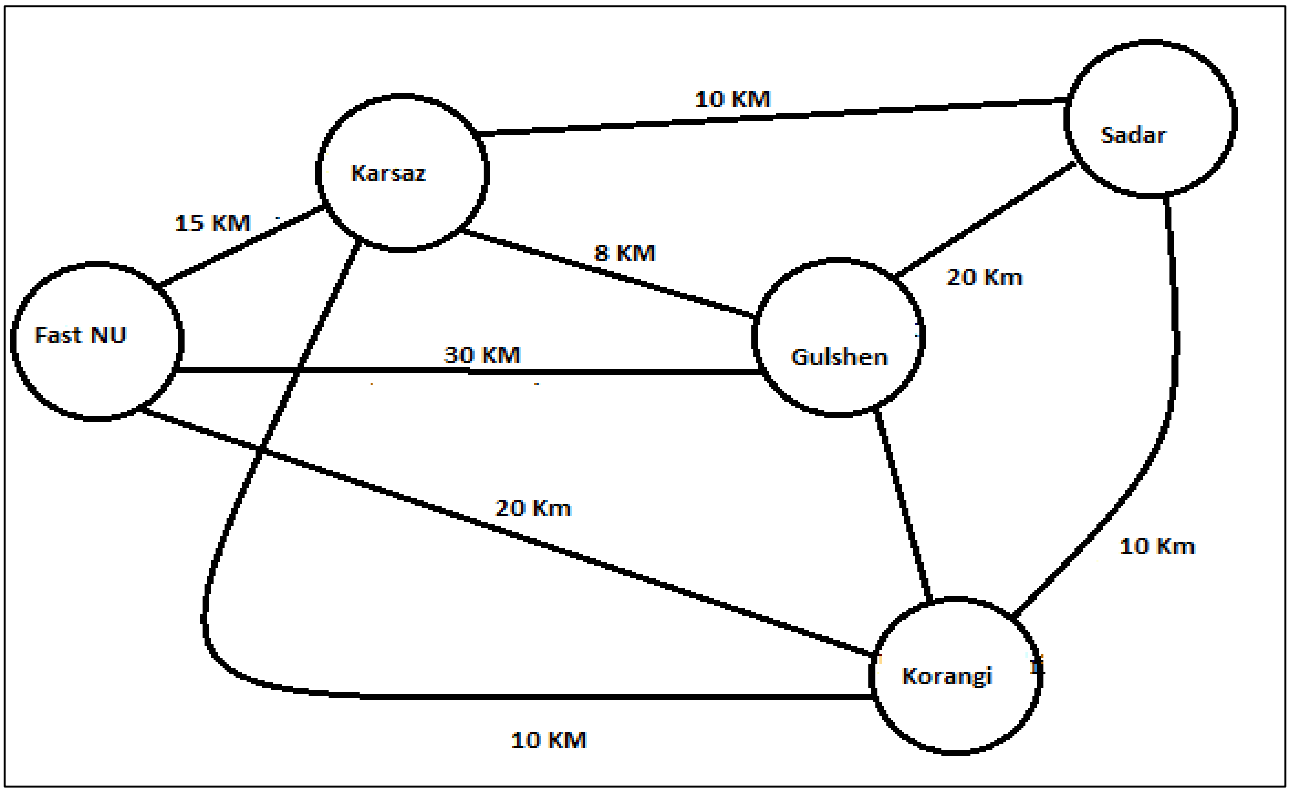
Tic tac toe is a very simple two-player game where both the player gets to choose any of the symbols between X and O. This game is played on a 3X3 grid board and one by one each player gets a chance to mark its respective symbol on the empty spaces of the grid. Once a player is successful in marking a strike of the same symbol either in the horizontal, vertical or diagonal way as shown in the picture below is created that player wins the game else the game goes on a draw if all the spots are filled.



write a program for a tic tac toe game for a single-player game, in this program user plays against a computer agent. computer agent and player get multiple chances to mark their respective symbol on the empty spaces of the grid. The goal of the computer agent is to win the game. You can assign (X or O) to a computer agent. On every move, the computer must strike its marking where necessary to win the game. If there is no winning move on any strike then the computer marks anywhere on the board.

Task#06 Utility-Based Agents

When planning a road trip, we are trying to minimize our costs in many different areas - gas, time, overnight stays, traffic costs, etc. Calculating these costs can take a lot of effort and time, Pathfinding algorithms are one of the utility-based agent and classical graph problem. The Shortest Path algorithm is an algorithm that calculates a path between two nodes in a weighted graph such as the sum of the values on the edges that form a path is minimized. Starting from the source node, the algorithm looks up the weights on the (out-)going (in weighted graphs) edges. It chooses the edge which, summed to the previous total sum, gives the lowest result. The algorithm runs through every node up until the destination point. Results are a path and the total sum of the shortest path. Your task is to write a program of utility-based agent that find the best route (minimum Distance, shortest path) from source to destination node.



Task#07 Learning Agent

Any agent, model, utility or goal-based agent can be transformed into learning agent. Few examples are snake game, Pac man, Self-driving cars, Ad recommendation system.

Read the below article and implement python code that how you can transform task 04, 05 and 06 into learning agent.

https://vitalflux.com/reinforcement-learning-real-world-examples.