LAB-A: WRITE A PROGRAM TO IMPLEMENT

A. SIMPLE REFLEX AGENT

B. UTILITY BASED AGENT

C.GOAL BASED AGENT

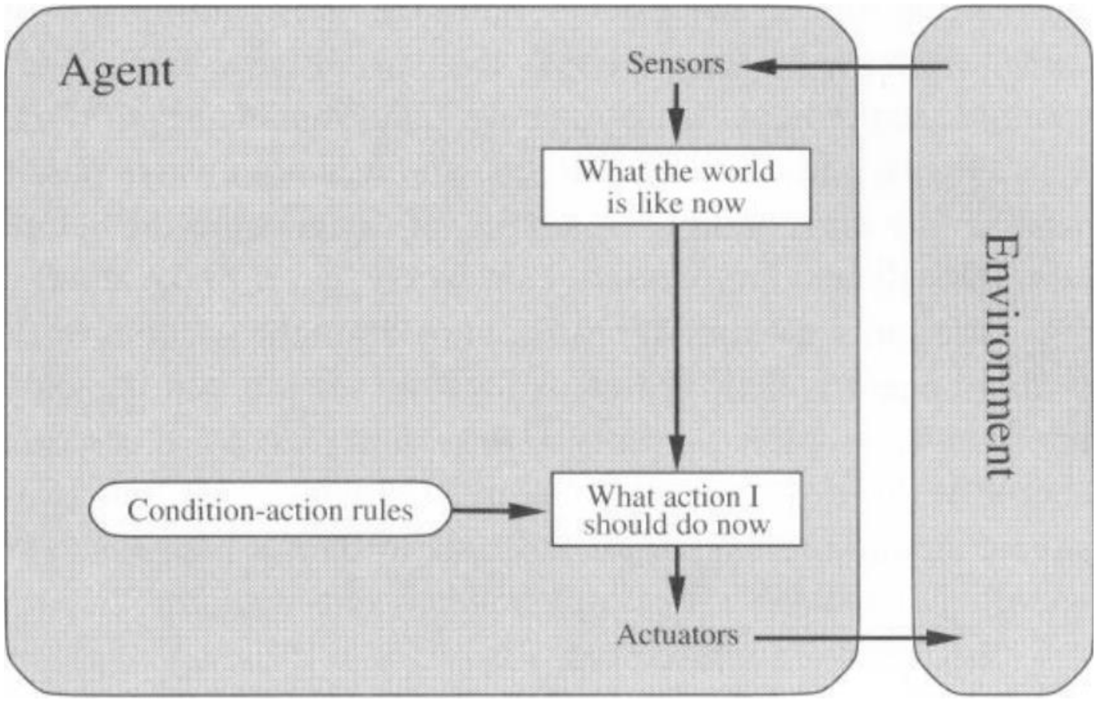
D.MODEL BASED AGENT

THE PROBLEM TAKEN SHOULD BE WUMPUS WORLD.

OBJECTIVE: To implement different agents using python language.

A:SIMPLE REFLEX AGENT

THEORY:



Simple reflex AI agents are those which respond to the current percepts. Which means they do not look into the history of percepts. Their decision making is purely based on what they see at that instance of time than what they understand from the past percepts. Given, that a Roomba is not a simple reflex AI in the first instance. Before they start the cleaning process, they should do a mapping of the room followed by path planning. Which means they should look into their past percepts to make current decisions, contrary to a simple reflex AI. However, these are higher level decisions related to planning. Some of the subfunctions are designed to respond to unknown changes in the environment which might pose threat to the robot itself like sudden obstacles in the form human/animal interventions etc., which are simple reflex AI like.
Coming to your answer of some examples.
An industrial robot working on part transfer across machines is a good example. An action or series of actions are triggered when a proximity sensor detects its presence at a spatial node. They usually do not store their past percepts and decide on the current actions.

Another such example is iDraw, a drawing robot. It directly converts the typed in characters into writing, without requiring to store past percept information.

It uses just condition-action rules

● The rules are like the form “if … then …”

● efficient but have narrow range of applicability

● Because knowledge sometimes cannot be stated explicitly

Works only :

● If the environment is fully observable

● Their actions are based on a direct mapping from states to actions .

● Cannot work well in other environments.

●With this mapping would be too large to store and would take too long to learn

ALGORITHM:

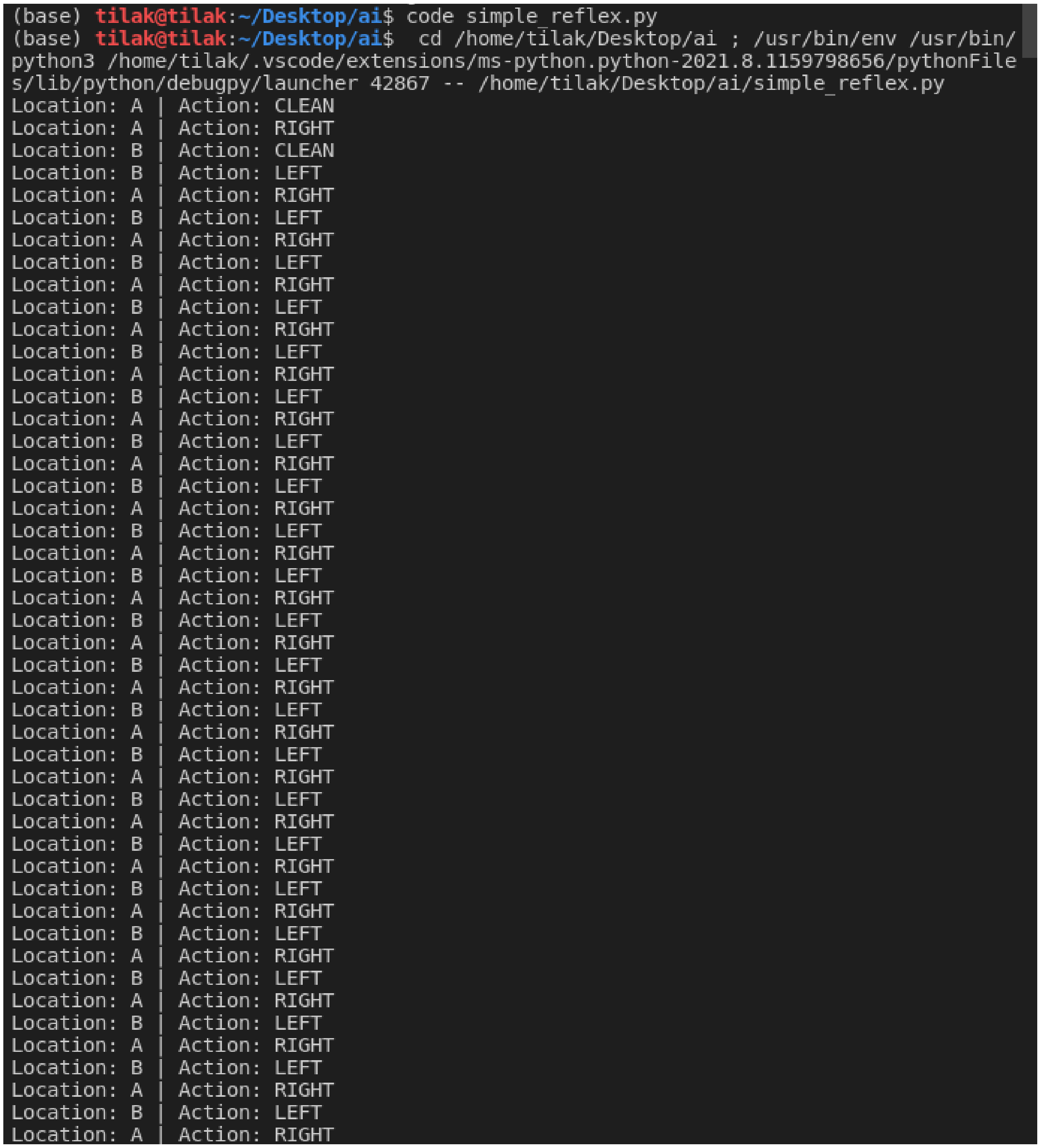
```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

PROGRAM:

```
import time


def reflex_agent(location, state):
if state=="DIRTY":
return 'CLEAN'
elif location=='A':
return 'RIGHT'
elif location=='B':
return 'LEFT'

def test(states):
while True:
location = states[0]
state = (states[2], states[1])[states[0]=='A']
action = reflex_agent(location, state)
print ("Location: "+location+" | Action: "+action)
if action == "CLEAN":
if location == 'A':
```

```
states[1]="CLEAN"
elif location == 'B':
states[2]="CLEAN"
elif action == "RIGHT":
states[0]='B'
elif action == "LEFT":
states[0]='A'
time.sleep(3)

test(['A','DIRTY','DIRTY'])
```

OUTPUT:

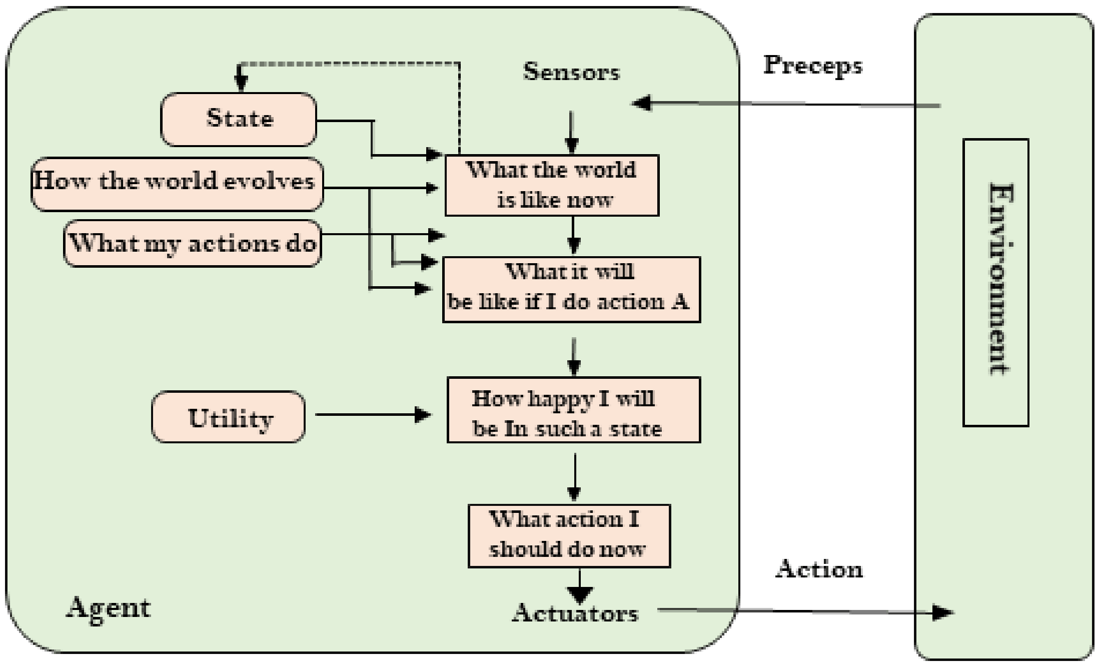


```
(base) tilak@tilak:~/Desktop/ai$ code simple_reflex.py
(base) tilak@tilak:~/Desktop/ai$  cd /home/tilak/Desktop/ai ; /usr/bin/env /usr/bin/
python3 /home/tilak/.vscode/extensions/ms-python.python-2021.8.1159798656/pythonFile
s/lib/python/debugpy/launcher 42867 -- /home/tilak/Desktop/ai/simple_reflex.py
Location: A | Action: CLEAN
Location: A | Action: RIGHT
Location: B | Action: CLEAN
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
Location: B | Action: LEFT
Location: A | Action: RIGHT
```

CONCLUSION:
In this we can implemented a vaccum cleaner using simplex reflex agent in wumpus world.

B. UTILITY BASED AGENT

THEORY:


Sensors
Preceps
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
Utility
How happy I will be In such a state
What action I should do now
Action
Agent
Actuators
Environment

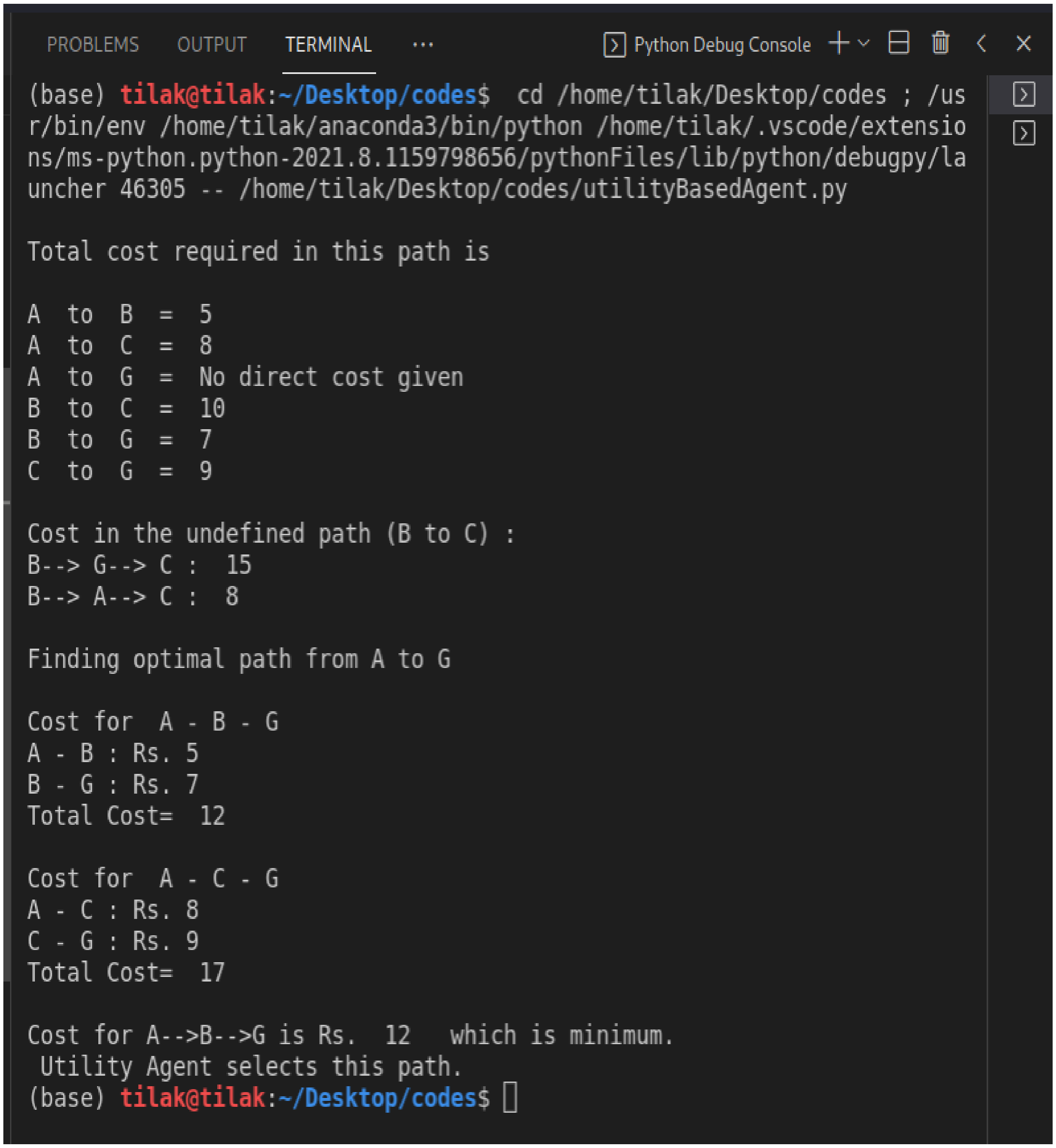
- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

PROGRAM:

```
#This program is for the utility agent that chooses the minimum cost path from source to
#destination.
def name(n):
        if n==0:
                return 'A'
        elif n==1:
                return 'B'
        elif n==2:
                return 'C'
        elif n==3:
                return 'G'
def costcheck(cost):
        if cost==0:
```

```
        return 'No direct cost given'
    else:
        return cost

def total(i,j,k):
    print("\nCost for ",name(i),"-",name(j),"-",name(k))
    print(name(i),"-",name(j),": Rs.",costs[i][j])
    print(name(j),"-",name(k),": Rs.",costs[j][k])
    print("Total Cost= ",costs[i][j]+costs[j][k])
    return costs[i][j]+costs[j][k]


costs=[[1,5,8,0],

[0,3,10,7],

[7,1,2,9],

[1,4,8,0]]

print("\nTotal cost required in this path is \n")
for i in range(0,4):
    for j in range(0,4):
        if i>=j:
            pass
        else:
            print(name(i)," to ",name(j)," = ",costcheck(costs[i][j]))
print("\nCost in the undefined path (B to C) : ")
print("B--> G--> C : ",costs[1][3]+costs[3][2])
print("B--> A--> C : ",costs[1][0]+costs[0][2])

## Checking for the shortest path
print("\nFinding optimal path from A to G")
a=total(0,1,3)
b=total(0, 2, 3)
if min(a,b)==a:
    print("\nCost for A-->B-->G is Rs. ", a," which is minimum.\n Utility Agent selects
    this path.")
else:
    print("\nCost for A-->C-->G is Rs. ", b," which is minimum. \n Ulility Agent selects
    this path.")
```

OUTPUT:



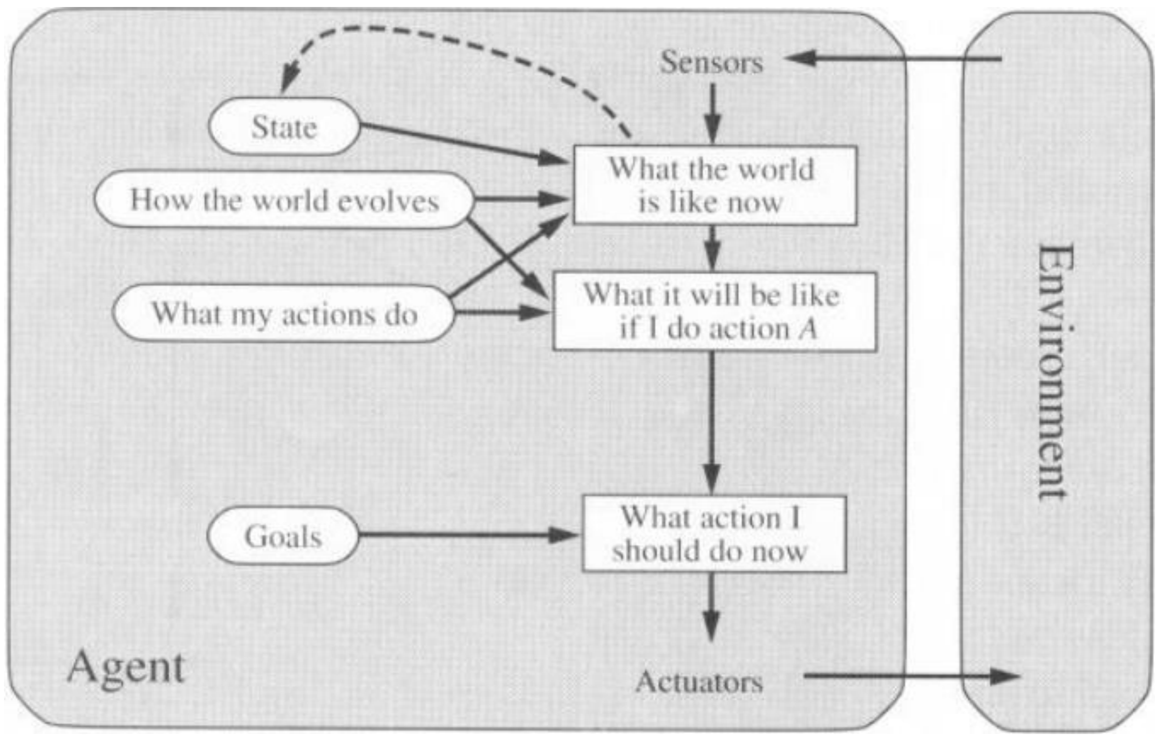
```
(base) tilak@tilak:~/Desktop/codes$  cd /home/tilak/Desktop/codes ; /us
r/bin/env /home/tilak/anaconda3/bin/python /home/tilak/.vscode/extensio
ns/ms-python.python-2021.8.1159798656/pythonFiles/lib/python/debugpy/la
uncher 46305 -- /home/tilak/Desktop/codes/utilityBasedAgent.py

Total cost required in this path is

A  to  B  =  5
A  to  C  =  8
A  to  G  =  No direct cost given
B  to  C  =  10
B  to  G  =  7
C  to  G  =  9

Cost in the undefined path (B to C) :
B--> G--> C :  15
B--> A--> C :  8

Finding optimal path from A to G

Cost for  A - B - G
A - B : Rs. 5
B - G : Rs. 7
Total Cost=  12

Cost for  A - C - G
A - C : Rs. 8
C - G : Rs. 9
Total Cost=  17

Cost for A-->B-->G is Rs.  12   which is minimum.
 Utility Agent selects this path.
(base) tilak@tilak:~/Desktop/codes$ 
```

CONCLUSION:

A utility-based agent acts based on how to most efficiently complete a goal.

C.GOAL BASED AGENT/ D.MODEL BASED AGENT

THEORY:


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

Current state of the environment is always not enough

The goal is another issue to achieve

- Judgment of rationality / correctness

Actions(chosen) → **goals,** based on the current state and the current percept.

ALGORITHM:

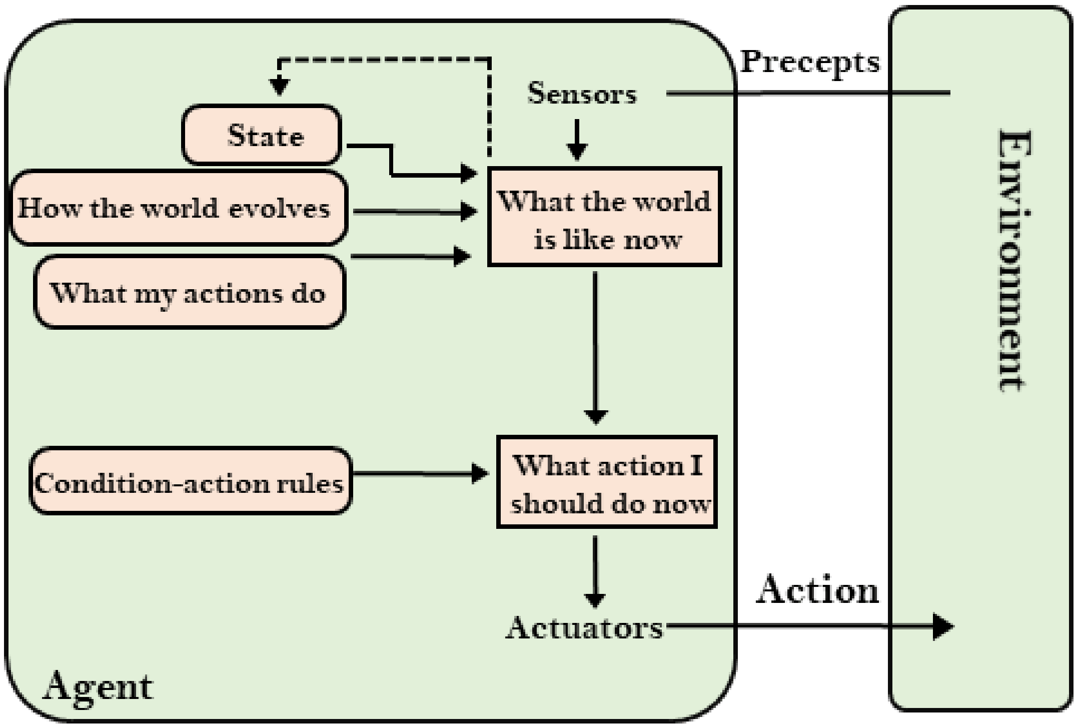
```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              transition_model, a description of how the next state depends on
                      the current state and action
              sensor_model, a description of how the current world state is reflected
                      in the agent's percepts
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, transition_model, sensor_model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Conclusion

- Goal-based agents are less efficient but more flexible
- Agent <--Different goals <-- different tasks

● Search and planning (two other sub-fields in AI) to find out the action sequences to achieve its goal

THEORY: MODEL BASED AGENT

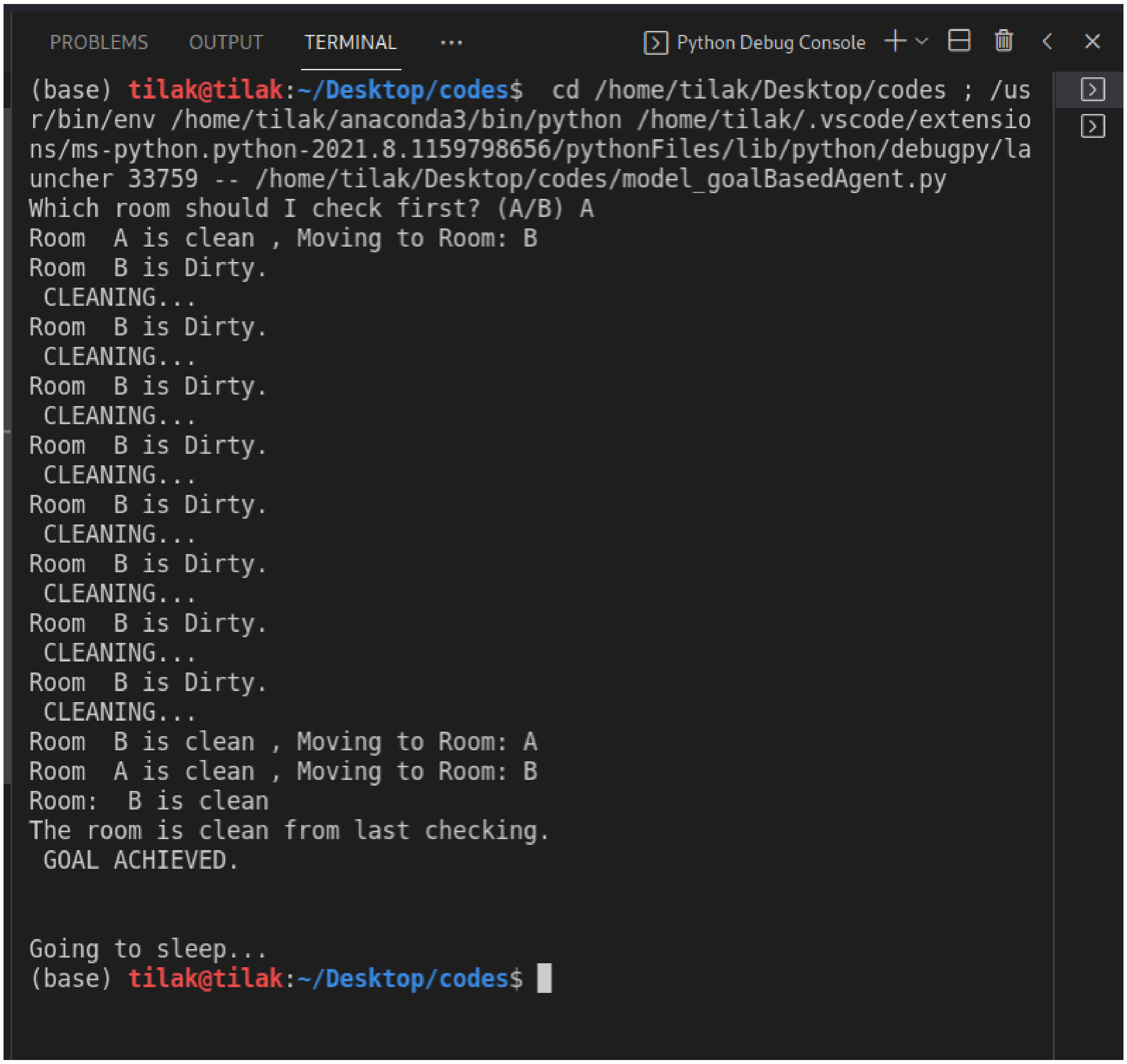


- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
    - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
    - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
    - How the world evolves
    - How the agent's action affects the world.

CODE FOR MODEL/GOAL BASED AGENTS:

```
import random
from time import sleep
from sys import exit
from random import randint
def main(roomNum):
        status=[1,1,1];
        while(1):
                status[2]=random.randint(0,1)#current state
                if status[2]==0:
                        goal(status,roomNum)
                        print("Room ",roomNum,"is clean , Moving to
                        Room:",anotherRoom(roomNum))
                        roomNum=anotherRoom(roomNum)
                        sleep(1)
                elif status[2]==1:
                        print("Room ",roomNum,"is Dirty.\n CLEANING...")
                        sleep(1)
                temp =status[1]
                status[1]=status[2]#previous states
                status[0]=temp #before previous
def anotherRoom(roomNum):
        if roomNum=='A' or roomNum=='a':
                return 'B'
        else:
                return 'A'
def goal(status,roomNum):
        if status[0]==0 and status[1]==0 and status[2]==0:
                print("Room: ",roomNum,"is clean")
                print("The room is clean from last checking. \n GOAL ACHIEVED.")
                print("\n\nGoing to sleep...")
                sleep(3)
                exit(0)

count=0
roomNum=input("Which room should I check first? (A/B) ")
main(roomNum)
```

OUTPUT:



```
(base) tilak@tilak:~/Desktop/codes$  cd /home/tilak/Desktop/codes ; /us
r/bin/env /home/tilak/anaconda3/bin/python /home/tilak/.vscode/extensio
ns/ms-python.python-2021.8.1159798656/pythonFiles/lib/python/debugpy/la
uncher 33759 -- /home/tilak/Desktop/codes/model_goalBasedAgent.py
Which room should I check first? (A/B) A
Room  A is clean , Moving to Room: B
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is Dirty.
 CLEANING...
Room  B is clean , Moving to Room: A
Room  A is clean , Moving to Room: B
Room:  B is clean
The room is clean from last checking.
 GOAL ACHIEVED.


Going to sleep...
(base) tilak@tilak:~/Desktop/codes$ 
```

Conclusion

This is the answer to the question of how many types of agents are there in artificial intelligence. All of these five agents are part of artificial intelligence agents. Each of the functions of the agents is different. There may exist other types of agents in real world too.we just implemented just four basic types of agents above.