Lab 3

Agents:

Simple Reflex Agents

Simple reflex agents act only on the basis of the current percept, ignoring the rest of the percept history. The agent function is based on the condition-action rule: if condition then action. This agent function only succeeds when the environment is fully observable. Some reflex agents can also contain information on their current state which allows them to disregard conditions whose actuators are already triggered.

Example 1

```
percept= ["Anum" , "Taimur" , "Ali" , "Saad"]
state = ["happy", "sad", "angry", "normal"]
rule= ["smile" , "cry" , "frown", "watch football"]
def GetState(cPercept):
    index=-1
    for p in percept:
        index=index+1
        if p==cPercept:
            return state[index]
def GetRule(cState):
    index=-1
    for s in state:
        index=index+1
        if s==cState:
            return rule[index]
def SimpleReflexAgent(cPercept):
    return GetRule(GetState(cPercept))
print ("MENU: ")
print (" 0:Anum   1:Taimur  2:Ali  3:Saad")
print (SimpleReflexAgent(percept[int(input("Input Number :"))]))
```

`def GetState(cPercept):`

This function takes in a percept (cPercept) and iterates over a list called "percept" to find the index of the percept. Once the index is found, it is used to return the corresponding state from a list called "state".

`def GetRule(cState):`

This function takes in a state (cState) and iterates over a list called "state" to find the index of the state. Once the index is found, it is used to return the corresponding rule from a list called "rule".

`def SimpleReflexAgent(cPercept):`

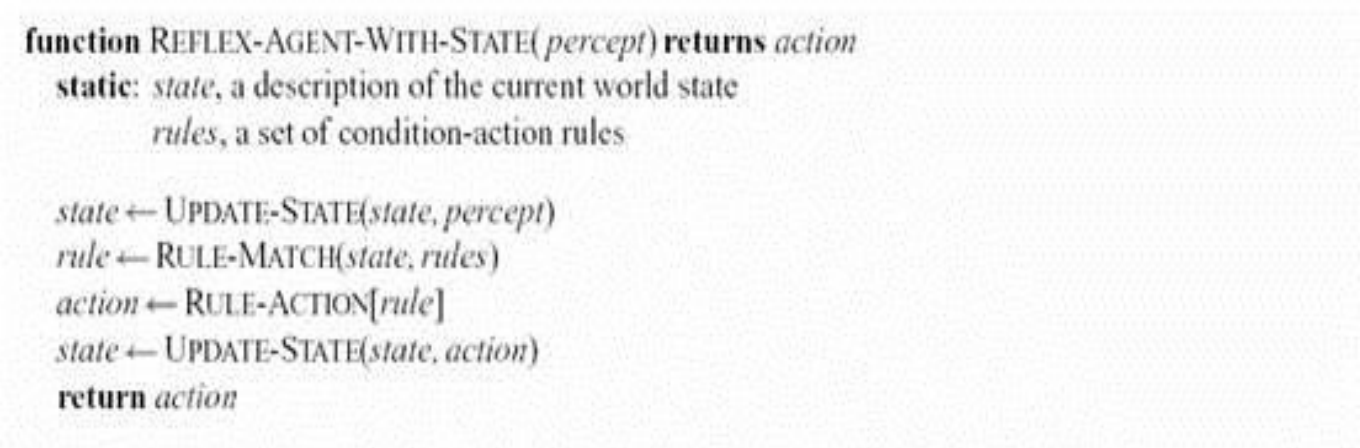
This function takes in a current percept (cPercept), calls GetState to determine the current state, and then calls GetRule to determine the appropriate action for that state. The function then returns the action.

This code prints a menu of options for the user to choose from, based on a list called "percept". The user is then prompted to input a number corresponding to one of the options, and that input is used as an index to pass the corresponding percept to the SimpleReflexAgent function. The resulting action is then printed to the console.

Overall, this code implements a simple reflex agent that uses a set of rules to determine an appropriate action based on the current percept. The agent is tested by allowing the user to input different percepts and observing the resulting actions.

Model based Reflex Agents

Model-based reflex agents are made to deal with partial accessibility; they do this by keeping track of the part of the world it can see now. It does this by keeping an internal state that depends on what it has seen before so it holds information on the unobserved aspects of the current state.




```
class ModelBasedReflexAgent:

    def __init__(self, state, rule):

        self.state = state

        self.rule = rule

        self.current_state = None
```

```
    def process(self, percept):
        self.update_state(percept)
        return self.choose_action()

    def update_state(self, percept):
        if "A" in percept:
            self.current_state = "A"
        elif "B" in percept:
            self.current_state = "B"
        elif "C" in percept:
            self.current_state = "C"
        elif "D" in percept:
            self.current_state = "D"
    def choose_action(self):
        if self.current_state is not None:
            action = self.rule[self.state.index(self.current_state)]
        else:
            action = "do nothing"
        return action
# Define the state and rule lists
```

```
state = ["A", "B", "C", "D"]

rule = ["move right", "move down", "move left", "move up"]



# Create a new model-based reflex agent

agent = ModelBasedReflexAgent(state, rule)



# Test the agent with different percepts

print(agent.process(["A"]))  # move right

print(agent.process(["B"]))  # move down

print(agent.process(["C"]))  # move left

print(agent.process(["D"]))  # move up

print(agent.process(["E"]))  # do nothing
```

Lab Journal 3:

1. Develop a medical diagnosis system, designed as a Simple reflex agent that diagnose the disease on the basis of provided symptoms and test reports. Symptoms and test reports should be taken from the user as percepts and agent has to display the diagnosed disease as its action.

   Acute appendicitis:

   Symptoms: Fever, Pain in Abdomen especially ILIAC FOSSA, vomiting,

   Test: Blood CP with ESR… TLC (Total leucocyte count) will be high, DLC (Differential leucocyte count) Neutrophils will be high , ESR high

   Treatment: Surgery

   Pneumonia:

   Symptoms: Fever, Cough (with sputum), Pain in chest

   Blood CP with ESR… TLC (Total leucocyte count) will be high, DLC (Differential leucocyte count) Neutrophils will be high , ESR high

X-ray chest: pneumonic patch (sometimes)

Treatment: Antibiotics

Acute Tonsilitis:

Symptoms: Fever, Cough

Test: Examine throat: (Red enlarged tonsils, pus in tonsils)

Treatment: anti-allergic, paracetamol. If not gone, add antibiotics orally. If not gone, add antibiotics IV

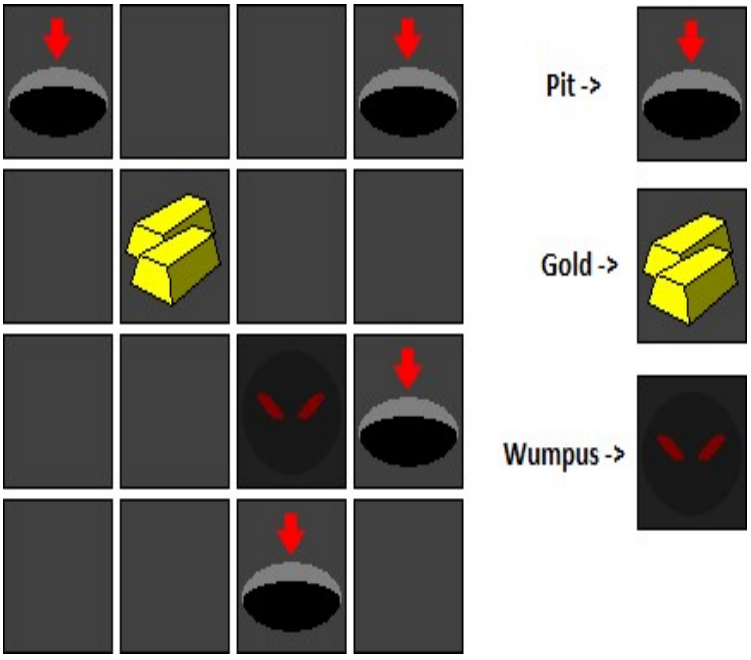
2. Develop a model based agent for the wumpus world shown in the image below. The agent has to find the gold while updating the state of the world after every action. Consider the wumpus world as deterministic and partially observable.
Goal: To find the gold and then to go back safely (avoiding pit and wumpus) to its starting location.

Percepts: pit, gold, empty, wumpus (can have percept of just one block ahead)

Actions: Grab gold, move left, move right, move up, move down, and do nothing.

You will need a class for Agent. It might be useful to define a list of nested lists depicting which blocks are adjacent to each block (in order to limit percepts to one block ahead in each adjacent direction) and update states accordingly.
First agent will check state, find out which block he's in.
If block has gold, agent gets the current Action 'Grab gold', then Grabs Gold, then returns to starting position.
Otherwise, then agent will look at one block ahead (either up, down, left, right) and get percept. Then agent will update state according to that block & percepts and decide action
Repeat steps until agent gets to gold or exhausts maximum steps =50

When looking for gold, if action is DO NOTHING then return failure and stop execution (this means that all adjacent blocks have either wumpus or pits.)
Return success if and only if agent finds gold, grabs gold, AND returns to starting position successfully
Print current position of agent as well as state after each repetition. You may find it useful to use input() function to prompt user after every repetition so that the output doesn‘t flash by too fast and can be viewed by us.



At some stage, if needed, you may also use the built-in random module to randomise the order in which you access adjacent block percepts. The random.shuffle (list_as_an_argument) function might come in handy.

3. Given a simple pacman game in figure below that consisting of 4*4 grid. The starting point of pacman is cell 0 and its goal is to consume/eat maximum food pallets, while considering following given limitations.
    - Pacman can move up, down, left right (keeping in view walls).
    - Pacman can eat power pallets, i.e., cherries to keep ghost scared such that if pacman enters the ghost cell its is not destroyed.
    - Pacman keeps moving until all the power pallets and food are consumed.

You need to devise a **model/goal-based agent** for the above given problem.