Artificial Intelligence

CSL 411

Lab Journal 3

Student Name:
Enrolment No
Class and Section:

Department of Computer Science
BAHRIA UNIVERSITY, ISLAMABAD

Lab # 3: Rational Agents

Objectives:

To implement Simple Reflex & Model Based Agent in Vacuum World.

Tools Used:

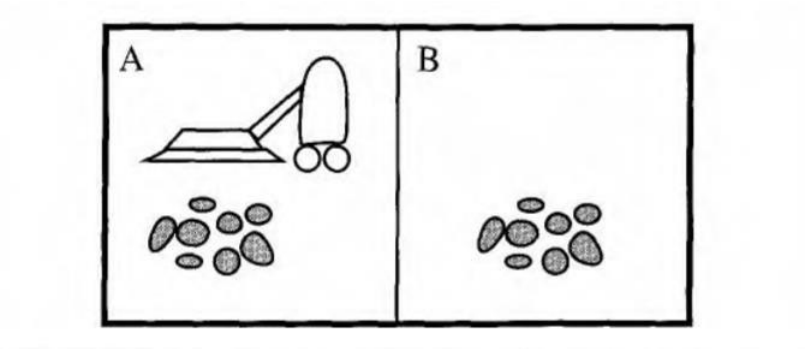
Python IDLE 3.4/Python IDLE 3.6

Submission Date:

Evaluation: **Signatures of Lab Engineer:**

Task # 1:

Consider the vacuum world shown in the figure below:



This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and1 whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. A simple program for the agent function of vacuum-world is shown below:

function REFLEX-VACUUM-AGENT([*location*,*status*]) **returns** an action

if *status* = *Dirty* **then return** *Suck*
else if *location* = *A* **then return** *Right*
else if *location* = **B then return Left**

Your task is to implement the above vacuum world and its agent program for a simple reflex agent. Also, suggest a performance measure and evaluate your program based on that performance measure. Modify your program accordingly.

Procedure/Program:

```
percept=['Dirty','Dirty']
actions=['Move Right','Move Left','Clean Dirt']
class Agent:
    def init (self):
        self.position=0
        self.currAction=0
        self.run_agent()
    def run_agent(self):
        if(percept[0]=='Dirty'):
            self.getAction(percept[0])
        elif(percept[1]=='Dirty'):
                self.getAction(percept[1])
        elif(percept[0]=='Clean'):
            self.getAction(percept[0])
        elif(percept[1]=='Clean'):
```

```
            self.getAction(percept[1])
    def getAction(self,cPercept):
        if(cPercept=='Dirty'):
            self.currAction=2
            print("Current Action : " + actions[self.currAction])
            self.updateRoom()
             self.updatePosition()
              if(self.position==0):
                 self.currAction=2
        elif(cPercept=='Clean'):
            self.currAction=1
            print("Current Action : " + actions[self.currAction])
            self.updateRoom()
            self.updatePosition()
    def updatePosition(self):
        print("Current Position : " , self.position)
        if(self.currAction==2):
            if(self.position==0):
                self.position=1
                print("Current Action : " + actions[self.position])
        elif(self.currAction==2):
            if(self.position==1):
                self.position=0
                print("Current Action : " + actions[self.position])
        elif(self.currAction==1):
            if(self.position==1):
                self.position=0
        elif(self.currAction==0):
            if(self.position==0):
                self.position=1
    def updateRoom(self):
        if(self.currAction==2 and self.position==0 ):
            percept[self.position]='Clean'
            print("Room A : " + percept[self.position])
        elif(self.currAction==2 and self.position==1 ):
            percept[self.position]='Clean'
            print("Room B : " + percept[self.position])
V_agent=Agent()
for i in percept:
    if(i=='Dirty'):
        V_agent.run_agent();
```

Result/Output:

```
In [80]: runfile('C:/Users/HP/.spyder-py3/
temp.py', wdir='C:/Users/HP/.spyder-py3')
Current Action : Clean Dirt
Room A : Clean
Current Position :  0
Current Action : Move Left
Current Action : Clean Dirt
Room B : Clean
Current Position :  1
```

Analysis/Conclusion:

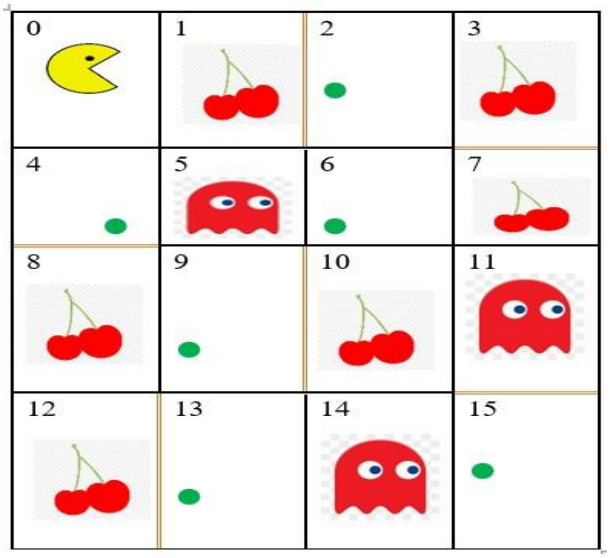
In this program I created a simple reflex agent which selects a random location A or B. If location A is dirty it cleans it and moves right towards B. If location B is dirty it cleans it and moves left towards A. If location A is not dirty but location B is dirty it will move to location B and cleans it and vice versa. If both the locations are clean it will do nothing and it also measures the performance of the program.

Task # 2:

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, i.e., if pacman enters the ghost cell its is not destroyed.
- Pacman keeps moving until all the power pallets are consumed.

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



Procedure/Program:

```
import sys
```

```
percept=['Down','Up','Right','Down','Right','Up','Right','Left','Down','Right','Down','Left','Down','Right','Left','L
eft','Up','Left','Down']
actions=['Move Right','Move Left','Move Up','Move Down','Eat Food','Eat Power Pallet','Avoid Ghost','Eat
Ghost']
Pacman=[['Pacman','PowerPallet','Food','PowerPallet'],['Food','Ghost','Food','PowerPallet'],['PowerPallet','Food'
,'PowerPallet','Ghost'],['PowerPallet','Food','Ghost','Food']]
class Agent:
    def init (self):
        self.currAction=0
        self.r=0
        self.c=0
        self.run_Agent()
    def run_Agent(self):
        print("Game Starting!")
        for i in range(len(percept)):
            self.getAction(percept[i])
        print("Game Finished! Pacman current position is Row:",self.r,"& column:",self.c)
    def getAction(self,cPercept):
         if cPercept=="Right":
            self.removePacman()
            self.c+=1
            if (self.c>=0 and self.c<=3) and (not(self.r==0 and self.c==1) or not(self.r==2 and self.c==1) or
not(self.r==3 and self.c==0)):
                if Pacman[self.r][self.c]=='Food':
                    print("Moving right.")
                    self.currAction = actions[4]
                    print("Food consumed.Yummy!")
                    self.updateGrid()
                    self.updatePacman()
                elif Pacman[self.r][self.c]=='PowerPallet':
                    print("Moving right.")
                    self.currAction = actions[5]
                    print("Power pallet consumed.")
                    self.updateGrid()
                    self.updatePacman()
                elif Pacman[self.r][self.c]=='Ghost':
                    if self.currAction == "Eat Power Pallet":
                        print("Moving right.")
                        self.currAction = actions[7]
                        print("Ghost consumed.")
                        self.updateGrid()
                        self.updatePacman()
                    else:
                        self.currAction = actions[6]
                        self.c-=1
                elif Pacman[self.r][self.c]==":
                    print("Moving right.")
                    print("Empty cell.")
            else:
                sys.exit("Incorrect Percept!")
         elif cPercept=="Left":
```

```
            self.removePacman()
            self.c-=1
            if (self.c>=0 and self.c<=3) and (not(self.r==0 and self.c==2) or not(self.r==2 and self.c==2) or
not(self.r==3 and self.c==1)):
                if Pacman[self.r][self.c]=='Food':
                    print("Moving left.")
                    self.currAction = actions[4]
                    print("Food consumed.Yummy!")
                    self.updateGrid()
                    self.updatePacman()
                elif Pacman[self.r][self.c]=='PowerPallet':
                    print("Moving left.")
                    self.currAction = actions[5]
                    print("Power pallet consumed.")
                    self.updateGrid()
                    self.updatePacman()
                elif Pacman[self.r][self.c]=='Ghost':
                    if self.currAction == "Eat Power Pallet":
                        print("Moving left.")
                        self.currAction = actions[7]
                        print("Ghost consumed.")
                        self.updateGrid()
                        self.updatePacman()
                    else:
                        self.currAction = actions[6]
                        self.c+=1
                elif Pacman[self.r][self.c]=='':
                    print("Moving left.")
                    print("Empty cell.")
            else:
                sys.exit("Incorrect Percept!")
        elif cPercept=="Down":
            self.removePacman()
            self.r+=1
            if (self.r>=0 and self.r<=3) and (not(self.r==1 and self.c==0) or not(self.r==0 and self.c==3) or
not(self.r==2 and self.c==3)):
                if Pacman[self.r][self.c]=='Food':
                    print("Moving down.")
                    self.currAction = actions[4]
                    print("Food consumed.Yummy!")
                    self.updateGrid()
                    self.updatePacman()
                elif Pacman[self.r][self.c]=='PowerPallet':
                    print("Moving down.")
                    self.currAction = actions[5]
                    print("Power pallet consumed.")
                    self.updateGrid()
                    self.updatePacman()
                elif Pacman[self.r][self.c]=='Ghost':
                    if self.currAction == "Eat Power Pallet":
                        print("Moving down.")
```

```
                    self.currAction = actions[7]
                    print("Ghost consumed.")
                    self.updateGrid()
                    self.updatePacman()
                else:
                    self.currAction = actions[6]
                    self.r-=1
            elif Pacman[self.r][self.c]==":
                print("Moving down.")
                print("Empty Cell.")
        else:
            sys.exit("Incorrect Percept!")
    elif cPercept=="Up":
        self.removePacman()
        self.r-=1
        if (self.r>=0 and self.r<=3) and (not(self.r==2 and self.c==0) or not(self.r==1 and self.c==3) or
not(self.r==3 and self.c==3)):
            if Pacman[self.r][self.c]=='Food':
                print("Moving up.")
                self.currAction = actions[4]
                print("Food consumed.Yummy!")
                self.updateGrid()
                self.updatePacman()
            elif Pacman[self.r][self.c]=='PowerPallet':
                print("Moving up")
                self.currAction = actions[5]
                print("Power pallet consumed.")
                self.updateGrid()
                self.updatePacman()
            elif Pacman[self.r][self.c]=='Ghost':
                if self.currAction == "Eat Power Pallet":
                    print("Moving up.")
                    self.currAction = actions[7]
                    print("Ghost consumed")
                    self.updateGrid()
                    self.updatePacman()
                else:
                    self.currAction = actions[6]
                    self.r+=1
            elif Pacman[self.r][self.c]==":
                print("Moving up.")
                print("Empty cell.")
        else:
            sys.exit("Incorrect Percept!")
    def removePacman(self):
        Pacman[self.r][self.c]=""
    def updatePacman(self):
        Pacman[self.r][self.c]="Pacman"
    def updateGrid(self):
        if self.currAction=="Eat Food":
            Pacman[self.r][self.c]=""
```

```
        elif self.currAction=="Eat Power Pallet":
            Pacman[self.r][self.c]=""
        elif self.currAction=="Eat Ghost":
            Pacman[self.r][self.c]=""
V_agent=Agent()
```

Result/Output:

```
In [32]: runfile('C:/Users/HP/.spyder-py3/
temp.py', wdir='C:/Users/HP/.spyder-py3')
Game Starting!
Moving down.
Food consumed.Yummy!
Moving up.
Empty cell.
Moving right.
Power pallet consumed.
Moving down.
Ghost consumed.
Moving right.
Food consumed.Yummy!
Moving up.
Food consumed.Yummy!
Moving right.
Power pallet consumed.
Moving left.
Empty cell.
Moving down.
Empty Cell.
Moving right.
Power pallet consumed.
Moving down.
Ghost consumed.
Moving left.
Power pallet consumed.
Moving down.
Ghost consumed.
Moving right.
Food consumed.Yummy!
Moving left.
Empty cell.
Moving left.
Food consumed.Yummy!
Moving up.
Food consumed.Yummy!
Moving left.
Power pallet consumed.
Moving down.
Power pallet consumed.
Game Finished! Pacman current position is Row: 3
& column: 0
```

Analysis/Conclusion:

In the above program I created a pacman game for 4*4 grid with the mentioned restriction.

Task # 3:

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. Also suggest that how can you convert this agent into **model-based agent**, what changes from implementation perspective can be done to convert it into model based.

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

Procedure/Program:

```
Symptoms=["Pain_in_Abdomen_especially_ILIAC_FOSSA","Vomiting","Cough(with_sputum)","Pa
in_in_chest" , "Cough" ]
Test=["Blood_CP_with_ESR" , "ESR","Blood_CP_with_ESR" , "X_ray_Chest" , "Examine_Throat"
]
Treatment=["Surgery","Antibiotics" , "Anti-Allergic " , "Paracetamol" ]
def Disease(disease):
    disease=disease-1
    for i in range(len(Symptoms)):
        if(i==disease):
            return Test[i]
def Diagnose(test):
    if(test=="Blood_CP_with_ESR"):
        s=str(input("Are You Done with Blood_CP_with_ESR ? Yes/No : "))
```

```
        if(s=="yes"):
            x=int(input("Enter Total Leucocyte : "))
            y=int(input("Enter Total Neutrophils : "))
            if(x >= 5000 or y>=5000):
                return Treatment[0]
            else:
                return Treatment[1]
        elif(s=="no"):
            print("Please go for Blood_CP_with_ESR ")
    elif(test=="ESR"):
        s=str(input("Are You Done with ESR ? Yes/No : "))
        if(s=="yes"):
            x=str(input("Is ESR is High ? Yes/No : "))
            if(s=="yes"):
                return Treatment[0]
            elif(s=="no"):
                return Treatment[1]
        elif(s=="no"):
            print("Please go for ESR ")
    elif(test=="X_ray_Chest"):
        s=str(input("Are You Done with Chest X_ray ? Yes/No : "))
        if(s=="yes"):
            return Treatment[1]
        elif(s=="no"):
            print("Please go for Chest X_ray ")
    elif(test=="Examine_Throat"):
        s=str(input("Are You Done with Examine_Throat ? Yes/No : "))
        if(s=="yes"):
            x=str(input("Do you have Red Enlarged Tonsils ? Yes/No : "))
            y=str(input("Do you have Pus in Tonsils ? Yes/No : "))
            if(x=="yes" or y=="yes"):
                return Treatment[2]+Treatment[3]
            elif(x=="no"):
                return Treatment[1]
            elif(y=="no"):
                return Treatment[1]+" IV"
            elif(s=="no"):
                print("Please go for Examine_Throat ")
def Medical_Diagnosis(disease):
    return Diagnose(Disease(disease))
print(" 1 : Pain_in_Abdomen_especially_ILIAC_FOSSA\n"+" 2 : Vomiting \n","3 :
Cough(with_sputum)\n","4 : Pain_in_chest\n" , "5 : Cough" )
print("Your Treatment Should be" , Medical_Diagnosis(int(input("Enter Disease : "))))
```

Result/Output:

```
In [94]: runfile('C:/Users/HP/.spyder-py3/temp.py', wdir='C:/Users/HP/.spyder-py3')
 1 : Pain_in_Abdomen_especially_ILIAC_FOSSA
 2 : Vomiting
 3 : Cough(with_sputum)
 4 : Pain_in_chest
 5 : Cough

Enter Disease : 1

Are You Done with Blood_CP_with_ESR ? Yes/No : no
Please go for Blood_CP_with_ESR
Your Treatment Should be None

In [95]: runfile('C:/Users/HP/.spyder-py3/temp.py', wdir='C:/Users/HP/.spyder-py3')
 1 : Pain_in_Abdomen_especially_ILIAC_FOSSA
 2 : Vomiting
 3 : Cough(with_sputum)
 4 : Pain_in_chest
 5 : Cough

Enter Disease : 1

Are You Done with Blood_CP_with_ESR ? Yes/No : yes

Enter Total Leucocyte : 6

Enter Total Neutrophils : 3
Your Treatment Should be Antibiotics
```

Analysis/Conclusion:

I have designed a medical diagnosis system as a simple reflex agent. It is converted to model based reflex agent by the treatment prescribed as it affects the person.