UNIVERSITY OF KARACHI

Lab # 04

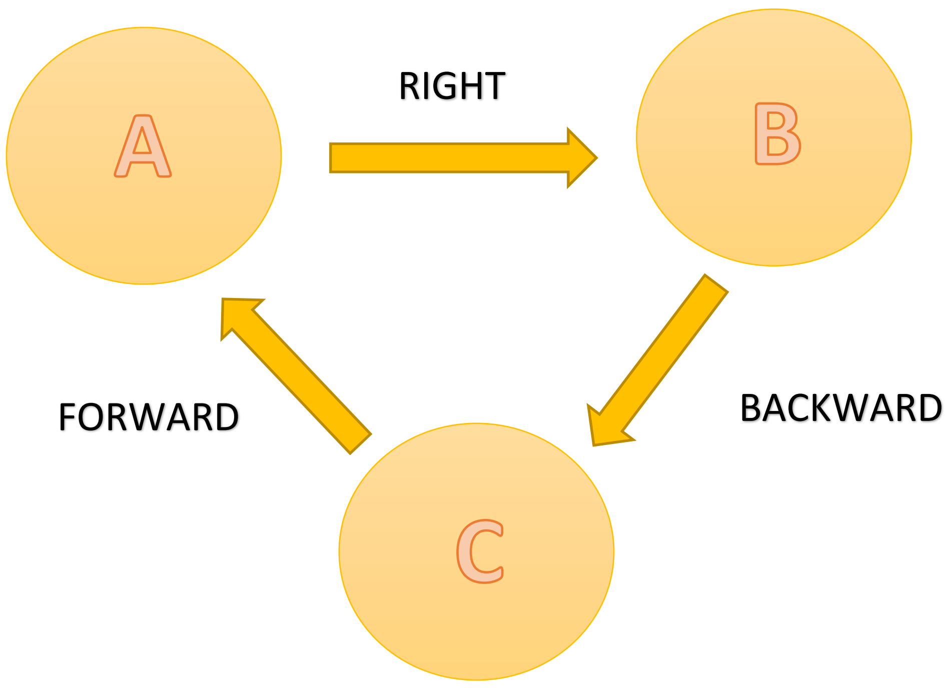
Name	:	Muhammad Habib Khan
Father Name	:	Muhammad Hanif
Submitted to	:	Dr. Humera Bashir
Submitted on	:	15/03/2023
Department	:	DCS (UBIT)
Discipline	:	Computer Science
Section	:	B
Semester	:	5th Semester (Spring 2023)
Course	:	BSCS-515 (AI)
Seat No.	:	B20102088

[illegible]
____
Signature of Student

____
Signature of Teacher

Run the two room vacuum cleaner agent program and understand it. Convert the program to a three-room environment.

For the given problem, I have assumed a three-room environment in a triangular manner. Room A is on the left, Room B is right to A and Room C is behind rooms A, B. The following diagram depicts their placement and actions required to reach one from another.



```
from abc import abstractmethod

class Environment(object):

    @abstractmethod
    def __init__(self, n):
        self.n = n
    def executeStep(self,n=1):
        raise NotImplementedError('action must be defined!')
    def executeAll(self):
        raise NotImplementedError('action must be defined!')
    def delay(self,n=100):
        self.delay = n

class Room:
 def __init__(self,location,status="dirty"):
    self.location = location
```

```python
    self.status = status

#Abstract agent
class Agent(object):
 @abstractmethod
 def __init__(self):
    pass

 @abstractmethod
 def sense(self,environment):
    pass

 @abstractmethod
 def act(self):
    pass


class VaccumAgent(Agent):

 def __init__(self):
    pass
 def sense(self,env):
    self.environment = env
 def act(self):
    if self.environment.currentRoom.status == 'dirty':
        return 'clean'
    if self.environment.currentRoom.location =='A':
        return 'right'
    if self.environment.currentRoom.location =='B':
        return 'backward'
    if self.environment.currentRoom.location =='C':
        return 'forward'

class ThreeRoomVaccumCleanerEnvironment(Environment):

 def __init__(self, agent):

    self.r1 = Room('A','dirty')
    self.r2 = Room('B','dirty')
    self.r3 = Room('C','dirty')
    self.agent = agent
    self.currentRoom = self.r1
    self.delay = 1000
    self.step = 1
    self.action = ""
 def executeStep(self,n=1):

    for _ in range(0,n):
        self.displayPerception()
        self.agent.sense(self)
        res = self.agent.act()
        self.action = res

        if res == 'clean':
            self.currentRoom.status = 'clean'

        elif res == 'right':
            self.currentRoom = self.r2

        elif res == 'backward':
            self.currentRoom = self.r3

        elif res == 'forward':
            self.currentRoom = self.r1

        # else:
        #     self.currentRoom = self.r1
        self.displayAction()
        self.step += 1

 def executeAll(self):
    raise NotImplementedError('action must be defined!')

 def displayPerception(self):
    print("Perception at step %d is [%s,%s]"
%(self.step,self.currentRoom.status,self.currentRoom.location))

 def displayAction(self):
    print("------- Action taken at step %d is [%s]"
%(self.step,self.action))

 def delay(self,n=100):
    self.delay = n


# Test program

if __name__ == '__main__': vcagent = VaccumAgent()
env = ThreeRoomVaccumCleanerEnvironment(vcagent)
env.executeStep(50)
```

2. Convert the environment to a ‘n’ room environment where n >= 2.

The n rooms are considered to be placed in a linear fashion and as soon as the last (nth) room is encountered, the agent goes back to the first room.

```python
from abc import abstractmethod

class Environment(object):

    @abstractmethod
    def __init__(self, n):
        self.n = n
    def executeStep(self,n=1):
        raise NotImplementedError('action must be defined!')
    def executeAll(self):
        raise NotImplementedError('action must be defined!')
    def delay(self,n=100):
        self.delay = n

class Room:
 def __init__(self,location,status="dirty"):
    self.location = location
    self.status = status

#Abstract agent
class Agent(object):

 @abstractmethod
 def __init__(self):
    pass

 @abstractmethod
 def sense(self,environment):
    pass

 @abstractmethod
 def act(self):
    pass

class VaccumAgent(Agent):

 def __init__(self):
    pass
 def sense(self,env):
    self.environment = env
 def act(self, n_agent):
    if self.environment.currentRoom.status == 'dirty':
        return 'clean'
    if self.environment.currentRoom.location == str(n_agent):
        return 'Back to 1'
    #if self.environment.currentRoom.location =='B':
    return 'forward'

class NRoomVaccumCleanerEnvironment(Environment):

 def __init__(self, agent, agent_num):
    self.n_agent = agent_num
    self.r = []
    for x in range(self.n_agent):
        self.r.append(Room(str(x+1), 'dirty'))
    self.agent = agent
    self.currentRoom = self.r[0]
    self.delay = 1000
    self.step = 1
    self.action = ""

 def executeStep(self,n=1):
    if self.n_agent < 2:
       print ("Number of agents must be equal or greater than 2")
       return
    counter = 0
    for _ in range(0,n):
        self.displayPerception()
        self.agent.sense(self)
        res = self.agent.act(self.n_agent)
        self.action = res
        if res == 'clean':
            self.currentRoom.status = 'clean'
            counter -=1
        elif res == 'forward':
            self.currentRoom = self.r[counter]
        else:
            self.currentRoom = self.r[0]
            counter = 0
        self.displayAction()
        self.step += 1
        counter += 1
        # if counter == self.n_agent:
        #    counter = 0
 def executeAll(self):
    raise NotImplementedError('action must be defined!')

 def displayPerception(self):
    print("Perception at step %d is [%s,%s]"
%(self.step,self.currentRoom.status,self.currentRoom.location))

 def displayAction(self):
    print("------- Action taken at step %d is [%s]"
%(self.step,self.action))

 def delay(self,n=100):
    self.delay = n

# Test program

if __name__ == '__main__': vcagent = VaccumAgent()
env = NRoomVaccumCleanerEnvironment(vcagent,1)
env.executeStep(50)
```

3. Does the agent ever stop? If no, can you make it stop? Is your program rational?

Our simple reflex agent is rational in the sense that it does logical reasoning (clean or dirty) to identify the action it is going to perform that is clean, left, etc. However, the agent currently stops as per the number of times (number of steps) the agent should be executed and not when all the squares are clean. We can make the following changes to the code to do so;

Changes are made to the three-room environment.

```
from abc import abstractmethod

class Environment(object):

    @abstractmethod
    def __init__(self, n):
        self.n = n
    def executeStep(self,n=1):
        raise NotImplementedError('action must be defined!')
    def executeAll(self):
        raise NotImplementedError('action must be defined!')
    def delay(self,n=100):
        self.delay = n

class Room:
 def __init__(self,location,status="dirty"):
    self.location = location
    self.status = status

#Abstract agent
class Agent(object):
 @abstractmethod
 def __init__(self):
    pass

 @abstractmethod
 def sense(self,environment):
    pass

 @abstractmethod
 def act(self):
    pass


class VaccumAgent(Agent):

 def __init__(self):
    pass
 def sense(self,env):
    self.environment = env
 def act(self):
    if self.environment.currentRoom.status == 'dirty':
        return 'clean'
    if self.environment.currentRoom.location =='A':
        return 'right'
    if self.environment.currentRoom.location =='B':
        return 'backward'
    if self.environment.currentRoom.location =='C':
        return 'forward'

class ThreeRoomVaccumCleanerEnvironment(Environment):

 def __init__(self, agent):

    self.r1 = Room('A','dirty')
    self.r2 = Room('B','dirty')
    self.r3 = Room('C','dirty')
    self.agent = agent
```

```python
        self.currentRoom = self.r1
        self.delay = 1000
        self.step = 1
        self.action = ""

    def check_status(self):
        if self.r1.status == self.r2.status == self.r3.status == 'clean':
            return True

    def executeStep(self,n=1):

        for _ in range(0,n):
            self.displayPerception()
            self.agent.sense(self)
            res = self.agent.act()
            self.action = res

            if self.check_status() == True:
                print("All rooms are clean")
                return

            if res == 'clean':
                self.currentRoom.status = 'clean'

            elif res == 'right':
                self.currentRoom = self.r2

            elif res == 'backward':
                self.currentRoom = self.r3

            elif res == 'forward':
                self.currentRoom = self.r1

            # else:
            #     self.currentRoom = self.r1
            self.displayAction()
            self.step += 1

    def executeAll(self):
        raise NotImplementedError('action must be defined!')

    def displayPerception(self):
        print("Perception at step %d is [%s,%s]"
%(self.step,self.currentRoom.status,self.currentRoom.location))

    def displayAction(self):
```

```
    print("------- Action taken at step %d is [%s]"
%(self.step,self.action))

 def delay(self,n=100):
    self.delay = n


# Test program

if __name__ == '__main__': vcagent = VaccumAgent()
env = ThreeRoomVaccumCleanerEnvironment(vcagent)
env.executeStep(50)
```

4. Score your agent, -1 points for moving from a room, +25 points to clean a room that is dirty, and -10 points if a room is dirty. The scoring will take place after every 1 second.

The following changes were made to the n-rooms environment code. time.sleep() function was added to the delay function already provided to halt the execution for a second each time score took place.

```
from abc import abstractmethod
import time

score = 0

class Environment(object):

    @abstractmethod
    def __init__(self, n):
        self.n = n
    def executeStep(self,n=1):
        raise NotImplementedError('action must be defined!')
    def executeAll(self):
        raise NotImplementedError('action must be defined!')
    def delay(self,n=100):
        self.delay = n

class Room:
 def __init__(self,location,status="dirty"):
    self.location = location
    self.status = status

#Abstract agent
```

```
class Agent(object):

 @abstractmethod
 def __init__(self):
    pass

 @abstractmethod
 def sense(self,environment):
    pass

 @abstractmethod
 def act(self):
    pass

class VaccumAgent(Agent):

 def __init__(self):
    pass
 def sense(self,env):
    self.environment = env

 def act(self, n_agent):
    if self.environment.currentRoom.status == 'dirty':
        time.sleep(1)
        global score
        score -= 10
        return 'clean'
    if self.environment.currentRoom.location == str(n_agent):
        return 'Back to 1'
    #if self.environment.currentRoom.location =='B':
    return 'forward'

class NRoomVaccumCleanerEnvironment(Environment):

 def __init__(self, agent, agent_num):
    self.n_agent = agent_num
    self.r = []
    for x in range(self.n_agent):
        self.r.append(Room(str(x+1), 'dirty'))
    self.agent = agent
    self.currentRoom = self.r[0]
    self.delay_time = 1
    self.step = 1
    self.action = ""

 def executeStep(self,n=1):
```

```python
    if self.n_agent < 2:
       print ("Number of agents must be equal or greater than 2")
       return
    counter = 0
    for _ in range(0,n):
        self.displayPerception()
        self.agent.sense(self)
        res = self.agent.act(self.n_agent)
        self.action = res
        if res == 'clean':
            self.delay(1)
            global score
            score += 25
            self.currentRoom.status = 'clean'
            counter -=1
        elif res == 'forward':
            self.currentRoom = self.r[counter]
            self.delay(1)
            score -= 1
        else:
            self.currentRoom = self.r[0]
            counter = 0
        self.displayAction()
        self.step += 1
        counter += 1
        # if counter == self.n_agent:
        #    counter = 0
        print("Score:" + str(score))

 def executeAll(self):
    raise NotImplementedError('action must be defined!')

 def displayPerception(self):
    print("Perception at step %d is [%s,%s]"
%(self.step,self.currentRoom.status,self.currentRoom.location))

 def displayAction(self):
    print("------- Action taken at step %d is [%s]"
%(self.step,self.action))

 def delay(self,n=100):
    self.delay_time = n
    time.sleep(self.delay_time)

# Test program
```

```
if __name__ == '__main__': vcagent = VaccumAgent()
env = NRoomVaccumCleanerEnvironment(vcagent,2)
env.executeStep(10)
```

5. Convert the agent to a reflex based agent with a model. Afterwards take the sensors away from the agents i.e., now the agent cannot perceive anything. Does your agent still work? if so, then why?

COVERTING INTO A MODEL_BASED REFLEX AGENT:

We can do the following changes to convert the simple reflex vacuum agent into a model-based reflex agent.

Start by defining a model which consists of all the possible states of environment, in our case Room A/B/C, and the possible actions the agent may take (left, right, forward, backward).

Model should also consist of the effect the agent's action have on the environment (cleaning A removes dirt from A) hence storing past information on the environment.

The model must also consider and keep track of how the world evolved on its own independent of the action taken to keep track of the environment and the state it is in which is currently not being observed.

Each new percept (state of environment) must update the model. Then according to the model, the agent takes the optimal action based on current percept and past data. (lets say we could move to either B and C from A but if we knew B was clean and C was dirty, we would move to C according to the data provided)

REMOVING THE ABILITY OF PERCEPTION:

The ability of perception is an integral part of any intelligent agent. Taking it away may render an intelligent agent ineffective. However, if the model-based reflex agent was run enough time to map out the environment completely, it may still be able to operate blindly on the basis of stored knowledge and previous state of the environment. But doing this will not allow the agent to learn anymore and any changes in the state of environment will not be registered.