k224413 Syeda Fakhira Saghir - Lab 2

Q1

```
class Environment:
 def __init__(self):
   # components are a part of the environment
   import random
   # 0 VULNERABLE 1 SAFE
   # critical components
   self.components=['A','B','C','D','E','F','G','H','I']
   self.vulnerability=[]
   for i in range(len(self.components)):
     self.vulnerability.append(random.randint(0,1))
     if self.vulnerability[i]==1:
       print(f'component {self.components[i]} is safe ✔' )
     else :
       print(f'component {self.components[i]} is vulnerable ⚠')

class SecurityAgent:
 def __init__(self):
   self.patch_list=[]
   pass

 def scanComponent(self,percept,component):
   if percept == 1:
     print("warning! vulnerable component detected ⚠")
     self.patch_list.append(component)
   else:
     print("success! component is safe ✔")

 def patchComponents(self, Env):
   for component in self.patch_list:
     print(f'patching {component}')
     index = env.components.index(component)
     env.vulnerability[index] = 0
     print(f'patched {component}')

def run_agent(agent, env):
 for i in range(len(env.components)):
```

```
   agent.scanComponent(env.vulnerability[i],env.components[i])
 agent.patchComponents(env)

agent = SecurityAgent()
env=  Environment()
run_agent(agent,env)
for i in range(len(env.components)):
 if env.vulnerability[i]==1:
   print(f'component {env.components[i]} is unsafe ⚠')
 else:
   print(f'component {env.components[i]} is safe ✔')
```

```
    print(f'component {env.components[i]} is safe ✔')
```

```
component A is safe ✔
component B is safe ✔
component C is safe ✔
component D is vulnerable ⚠
component E is safe ✔
component F is vulnerable ⚠
component G is safe ✔
component H is vulnerable ⚠
component I is vulnerable ⚠
warning! vulnerable component detected ⚠
warning! vulnerable component detected ⚠
warning! vulnerable component detected ⚠
success! component is safe ✔
warning! vulnerable component detected ⚠
success! component is safe ✔
warning! vulnerable component detected ⚠
success! component is safe ✔
success! component is safe ✔
patching A
patched A
patching B
patched B
patching C
patched C
patching E
patched E
patching G
patched G
component A is safe ✔
component B is safe ✔
component C is safe ✔
component D is safe ✔
component E is safe ✔
component F is safe ✔
component G is safe ✔
component H is safe ✔
component I is safe ✔
```

Q2

```
# Q2
```

```
class Environment:
 def update_servers(self):
   for i in range(len(self.tasks)):
     if self.tasks[i] <= 4:
       self.servers[i]="Underloaded"
     elif self.tasks[i]>=6:
       self.servers[i]="Overloaded"
     else:
       self.servers[i]="Balanced"

 def __init__(self, n):
   self.tasks={}
   self.servers = {}
   import random
   for i in range(n):
     self.tasks[i]=random.randint(0,10)
   self.update_servers()

 def get_percept(self):
   return self.servers

class LoadBalancerAgent:
 def __init__(self):
   pass

 def balanceLoad(self, env):
       for i in range(len(env.servers)):
           if env.servers[i] == "Overloaded":
               balanced = False
               for j in range(len(env.servers)):
                   if env.servers[j] == "Underloaded" and i != j:
                       transfer = min(env.tasks[i] - 6, 4 -
env.tasks[j])+1
                       if transfer > 0:
                           env.tasks[i] -= transfer
                           env.tasks[j] += transfer
                           env.update_servers()
                           print(f"{transfer} tasks are transferred from
server {i} to server {j}")
                           balanced = True
```

```
                       if env.servers[i] != "Overloaded":
                           break
               if not balanced:
                   print(f"Balancing not possible at the moment for server
{i}")

def runAgent(agent, env):
 print(f"servers before balancing \n{env.servers}\n")
 print(f"tasks before balancing \n{env.tasks}\n")
 agent.balanceLoad(env)
 print(env.servers)
 print(f"servers after balancing \n{env.servers}\n")
 print(f"tasks after balancing \n{env.tasks}\n")

env = Environment(5)
agent = LoadBalancerAgent()
runAgent(agent, env)
```

```
agent = LoadBalancerAgent()
runAgent(agent, env)
```

```
servers before balancing
{0: 'Underloaded', 1: 'Underloaded', 2: 'Underloaded', 3: 'Overloaded', 4: 'Overloaded'}

tasks before balancing
{0: 0, 1: 2, 2: 2, 3: 10, 4: 8}

5 tasks are transferred from server 3 to server 0
3 tasks are transferred from server 4 to server 1
{0: 'Balanced', 1: 'Balanced', 2: 'Underloaded', 3: 'Balanced', 4: 'Balanced'}
servers after balancing
{0: 'Balanced', 1: 'Balanced', 2: 'Underloaded', 3: 'Balanced', 4: 'Balanced'}

tasks after balancing
{0: 5, 1: 5, 2: 2, 3: 5, 4: 5}
```

```
runAgent(agent, env)
```

```
servers before balancing
{0: 'Overloaded', 1: 'Overloaded', 2: 'Balanced', 3: 'Overloaded', 4: 'Underloaded'}

tasks before balancing
{0: 9, 1: 9, 2: 5, 3: 8, 4: 2}

3 tasks are transferred from server 0 to server 4
Balancing not possible at the moment for server 1
Balancing not possible at the moment for server 3
{0: 'Overloaded', 1: 'Overloaded', 2: 'Balanced', 3: 'Overloaded', 4: 'Balanced'}
servers after balancing
{0: 'Overloaded', 1: 'Overloaded', 2: 'Balanced', 3: 'Overloaded', 4: 'Balanced'}

tasks after balancing
{0: 6, 1: 9, 2: 5, 3: 8, 4: 5}
```

Q3

```
class environment:
 def __init__(self,n):
   self.tasks={}
   import random
   for i in range(n):
     self.tasks[i]=random.choice(["Completed","Failed"])

 def get_percept(self):
   return self.tasks

class BackupManagementAgent:
 def __init__(self):
   pass

 def retryFailedTasks(self, env):
```

```
    for i in range(len(env.tasks)):
      if env.tasks[i] == "Failed":
        env.tasks[i] = "Completed"
        print(f"Task {i} is completed ✅")
      else:
        print(f"Task {i} is already completed ❕")

def runAgent(agent, env):
 print(f"tasks before retrying \n{env.tasks}\n")
 agent.retryFailedTasks(env)
 print(f"tasks after retrying \n{env.tasks}\n")

env=environment(10)
agent=BackupManagementAgent()
runAgent(agent,env)
```

```
tasks before retrying
{0: 'Completed', 1: 'Failed', 2: 'Failed', 3: 'Failed', 4: 'Completed', 5: 'Failed', 6: 'Completed', 7: 'Completed', 8: 'Completed', 9: 'Completed'}

Task 0 is already completed ❕
Task 1 is completed ✅
Task 2 is completed ✅
Task 3 is completed ✅
Task 4 is already completed ❕
Task 5 is completed ✅
Task 6 is already completed ❕
Task 7 is already completed ❕
Task 8 is already completed ❕
Task 9 is already completed ❕
tasks after retrying
{0: 'Completed', 1: 'Completed', 2: 'Completed', 3: 'Completed', 4: 'Completed', 5: 'Completed', 6: 'Completed', 7: 'Completed', 8: 'Completed', 9: 'Comple
```

Q4

```
# Task # 4 (utility based agent)
# A cybersecurity exercise is being conducted for a company’s security system, which consists
# of nine critical components (A through I). Each component of the system can either be Safe
# or have Vulnerabilities of varying severity. The company wants to ensure that its system
# remains secure, but it only has access to a basic security service that can patch Low Risk
# Vulnerabilities. High Risk Vulnerabilities require purchasing a premium security service to
# patch.
# In this scenario, the goal is to simulate how a Utility-Based Security Agent scans and
# patches the system based on the vulnerabilities detected and the available resources
```

```
# (limited patching service).
# ● Initial System Check:
# ○ Initialize the system environment with random vulnerabilities (Safe,
Low Risk
# Vulnerable, and High Risk Vulnerable).
# ○ Display the initial state of the system, showing which components are
Safe
# and which have Vulnerabilities.

# ● System Scan:
# ○ The security agent will scan each component.
# ○ If a component is Vulnerable, the agent logs a warning.
# ○ If it is Safe, a success message is logged.
# ● Patching Vulnerabilities:
# ○ The agent will patch all Low Risk Vulnerabilities.
# ○ The agent will log a message for High Risk Vulnerabilities indicating
the need
# for premium service to patch them.

# ● Final System Check:
# ○ Display the system’s final state to confirm that all Low Risk
Vulnerabilities
# have been patched.
# ○ The High Risk Vulnerabilities will remain unresolved unless the
premium
# service is purchased.

class Environment:
 def displayEnvironment(self):
     for i in range(len(self.components)):
       if self.vulnerability[i]=='Safe':
         print(f'{self.components[i]} : safe ✔' )
       elif self.vulnerability[i]=='Low Risk Vulnerable':
         print(f'{self.components[i]} : Low Risk Vulnerable ⚠')
       else:
         print(f'{self.components[i]} : High Risk Vulnerable ❌')
 def __init__(self):
   # components are a part of the environment
   import random
   # 0 VULNERABLE 1 SAFE
```

```
    # critical components
    self.components=['A','B','C','D','E','F','G','H','I']
    self.vulnerability={}
    for i in range(len(self.components)):
      self.vulnerability[i]=random.choice(['Safe', 'Low Risk Vulnerable',
"High Risk Vulnerable"])
    self.displayEnvironment()


class SecurityAgent:
  def __init__(self):
    self.patch_list=[]
    pass

  def scanComponent(self,env):
    for i in range(len(env.components)):
        if env.vulnerability[i]=='Safe':
          print(f'success {env.components[i]} is safe ✅' )
        elif env.vulnerability[i]=='Low Risk Vulnerable':
          print(f'warning!! component {env.components[i]} is Low Risk
Vulnerable ⚠')
          self.patch_list.append(env.components[i])
        else:
          print(f'WARNING!!!! component {env.components[i]} is High Risk
Vulnerable ❌')
          self.patch_list.append(env.components[i])

  def patchComponents(self, env):
    for i in range(len(env.components)):
        if env.vulnerability[i]=='Low Risk Vulnerable':
          print(f'patching {env.components[i]} \n patched
{env.components[i]}')
          env.vulnerability[i]=='Safe'
        else:
          print(f'premium service needed to patch {env.components[i]}')

  def final_check(self,env):
    for i in range(len(env.components)):
        if env.vulnerability[i]=='Safe':
          print(f'{env.components[i]} is safe ✅' )
```

```
      elif env.vulnerability[i]=='Low Risk Vulnerable':
        print(f'component {env.components[i]} is Low Risk Vulnerable and
has not been patched')

      else:
        print(f'WARNING!!!! component {env.components[i]} needs premiun
service')


def runAgent(agent, env):
 for i in range(len(env.components)):
   agent.scanComponent(env)
 agent.patchComponents(env)

agent = SecurityAgent()
env=  Environment()
runAgent(agent,env)
for i in range(len(env.components)):
 if env.vulnerability[i]==1:
   print(f'component {env.components[i]} is unsafe ⚠')
 else:
   print(f'component {env.components[i]} is safe ✔')
```

```
A : High Risk Vulnerable ❌
B : High Risk Vulnerable ❌
C : Low Risk Vulnerable ⚠
D : Low Risk Vulnerable ⚠
E : High Risk Vulnerable ❌
F : Low Risk Vulnerable ⚠
G : Low Risk Vulnerable ⚠
H : Low Risk Vulnerable ⚠
I : safe ✔
WARNING!!!! component A is High Risk Vulnerable ❌
WARNING!!!! component B is High Risk Vulnerable ❌
warning!! component C is Low Risk Vulnerable ⚠
warning!! component D is Low Risk Vulnerable ⚠
WARNING!!!! component E is High Risk Vulnerable ❌
warning!! component F is Low Risk Vulnerable ⚠
warning!! component G is Low Risk Vulnerable ⚠
warning!! component H is Low Risk Vulnerable ⚠
success I is safe ✅
WARNING!!!! component A is High Risk Vulnerable ❌
WARNING!!!! component B is High Risk Vulnerable ❌
warning!! component C is Low Risk Vulnerable ⚠
warning!! component D is Low Risk Vulnerable ⚠
WARNING!!!! component E is High Risk Vulnerable ❌
warning!! component F is Low Risk Vulnerable ⚠
warning!! component G is Low Risk Vulnerable ⚠
warning!! component H is Low Risk Vulnerable ⚠
success I is safe ✅
WARNING!!!! component A is High Risk Vulnerable ❌
WARNING!!!! component B is High Risk Vulnerable ❌
warning!! component C is Low Risk Vulnerable ⚠
warning!! component D is Low Risk Vulnerable ⚠
WARNING!!!! component E is High Risk Vulnerable ❌
warning!! component F is Low Risk Vulnerable ⚠
warning!! component G is Low Risk Vulnerable ⚠
warning!! component H is Low Risk Vulnerable ⚠
```

Sugge

```
class S
  def _
    sel
    pas

  def s
    for
```

Use code

Explanatio

Enter a pro

Responses may di
Learn more

✓ 0s completed at 15:36

```
warning!! component D is Low Risk Vulnerable ⚠
WARNING!!!! component E is High Risk Vulnerable ❌
warning!! component F is Low Risk Vulnerable ⚠
warning!! component G is Low Risk Vulnerable ⚠
warning!! component H is Low Risk Vulnerable ⚠
success I is safe ✅
premium service needed to patch A
premium service needed to patch B
patching C
 patched C
patching D
 patched D
premium service needed to patch E
patching F
 patched F
patching G
 patched G
patching H
 patched H
premium service needed to patch I
component A is safe ✔
component B is safe ✔
component C is safe ✔
component D is safe ✔
component E is safe ✔
component F is safe ✔
component G is safe ✔
component H is safe ✔
component I is safe ✔
```

Q5

```
#Task # 5 (Goal based agent)
# In a hospital, a delivery robot is tasked with delivering medicines to
patients, assisting
# nurses, and performing other related activities in an efficient manner.
The goal of the robot is
# to automatically move through hospital corridors, pick up medicines,
deliver them to the
# correct patient rooms, and perform various tasks such as scanning
patient IDs or alerting
# staff.
# ● Components:
# ○ Agent: The hospital delivery robot, which can move around, interact
with
# patient rooms, pick up medicines, deliver them, and alert nurses or
doctors
# when needed.
# ● Environment: The hospital layout, including:
```

```
# ○ Corridors
# ○ Patient rooms
# ○ Nurse stations
# ○ Medicine storage areas
# ● Actions:
# ○ Move to a location (room, station, etc.).
# ○ Pick up medicine from storage.
# ○ Deliver medicine to the patient’s room.
# ○ Scan patient ID for verification.
# ○ Alert staff for critical situations.
# ● Perceptions:
# ○ Room numbers (where the robot should deliver the medicine).
# ○ Patient schedules (timing for when patients need their medicines).
# ○ Medicine type (specific medicines to be delivered to patients).
# ○ Staff availability (alerts if staff assistance is needed).
# Goal-Based Agent Approach:
# Goal: Deliver medicine to patients based on a schedule and room number,
while ensuring all
# deliveries are correctly made. The robot must scan the patient's ID
before delivering and
# alert nurses or doctors if needed.
import random

class Environment:
   def __init__(self):
       self.locations = ['Corridor', 'Medicine Storage', 'Nurse Station',
'room_1', 'room_2', 'room_3']
       self.patients = {
           'room_1': {'id': 'P1', 'medicine': 'panadol', 'schedule': '1:00
AM'},
           'room_2': {'id': 'P2', 'medicine': 'Arinac', 'schedule': '2:00
PM'},
           'room_3': {'id': 'P3', 'medicine': 'Softin', 'schedule': '11:00
AM'}
       }
       self.staffAvaliability = random.choice([True, False])

   def getInfo(self, roomKey):
       return self.patients.get(roomKey)
```

```
class Agent:
   def __init__(self):
       self.location = 'Medicine Storage'
       self.medicineToDeliver = None

   def move(self, destination):
       print(f"moving from {self.location.replace('_', ' ')} to
{destination.replace('_', ' ')}")
       self.location = destination

   def collectMedicine(self, medicine):
       if self.location == 'Medicine Storage':
           self.medicineToDeliver = medicine
           print(f"collected {medicine}.")
       else:
           print("medicine unavailable. go to medicine storage.")

   def verifyPatient(self, expectedId):
       print(f"scanned patient id: {expectedId}.")
       return True  # assuming successful verification for now

   def deliverMedicine(self, room, patientInfo):
       if self.location == room:
           if self.verifyPatient(patientInfo['id']):
               if self.medicineToDeliver == patientInfo['medicine']:
                   print(f"successfully delivered {self.medicineToDeliver}
to {self.location.replace('_', ' ')} ✅")
                   self.medicineToDeliver = None
               else:
                   print(f"carrying the wrong medicine. expected
{patientInfo['medicine']} ❌")
           else:
               print("id mismatch! 🚫")
       else:
           print("wrong room ⛔")

def runAgent(agent, env):
   for room, patientInfo in env.patients.items():
       agent.move('Medicine Storage')
       agent.collectMedicine(patientInfo['medicine'])
```

```
        agent.move(room)
        if not env.staffAvaliability:
            print("alert! assistance required from medical staff ⚠")
        agent.deliverMedicine(room, env.getInfo(room))

env = Environment()
agent = Agent()
runAgent(agent, env)
```

```
moving from Medicine Storage to Medicine Storage
collected panadol.
moving from Medicine Storage to room 1
scanned patient id: P1.
successfully delivered panadol to room 1 ✅
moving from room 1 to Medicine Storage
collected Arinac.
moving from Medicine Storage to room 2
scanned patient id: P2.
successfully delivered Arinac to room 2 ✅
moving from room 2 to Medicine Storage
collected Softin.
moving from Medicine Storage to room 3
scanned patient id: P3.
successfully delivered Softin to room 3 ✅
```

Q6

```
import random

class Environment:
 def __init__(self):
   self.rooms={}
   self.status={}

 def initialize(self):
   charr='a'
   print("++++++++++++++ initial state of rooms +++++++++++++\n")
   for i in range(3):
```

```
     self.rooms[i] = {}
     self.status[i]={}
     for j in range(3):
       self.rooms[i][j]= charr
       if self.rooms[i][j] in ('a', 'b', 'd', 'f', 'g', 'h'):
         self.status[i][j]='safe'
       else:
         self.status[i][j]='fire'
       print(charr + ' '+ self.status[i][j]+ '\n')

       charr=chr(ord(charr)+1)
  def displayEnvironment(self):
   print("++++++++++++++___ Displaying Environment ___++++++++++++++\n")
   for i in range(3):
     for j in range(3):
       print('\n',self.status[i][j], self.rooms[i][j] , end=' ')
     print()

 def get_percept(self):
   return self.rooms, self.status

class Agent:
 def __init__(self):
   pass
  def put_out_fire(self, env):
   for i in range(3):
     for j in range(3):
       if env.status[i][j]=='fire':
         print(f'fire in room {env.rooms[i][j]} 🔥❗ extinguishing fire
now 🧯')
         env.status[i][j]='safe'
       else:
         print(f'no fire in room {env.rooms[i][j]} ✅')
       env.displayEnvironment()

def runAgent(agent, env):
   env.initialize()
   agent.put_out_fire(env)

env= Environment()
```

```
agent=Agent()
runAgent(agent,env)
```

```
++++++++++++++ initial state of rooms ++++++++++++++

a safe

b safe

c fire

d safe

e fire

f safe

g safe

h safe

i fire

no fire in room a ✅
++++++++++++++___ Displaying Environment ___++++++++++++++

safe a,safe b,fire c,
safe d,fire e,safe f,
safe g,safe h,fire i,
no fire in room b ✅
++++++++++++++___ Displaying Environment ___++++++++++++++

safe a,safe b,fire c,
safe d,fire e,safe f,
safe g,safe h,fire i,
fire in room c 🔥❗ extinguishing fire now 🧯
++++++++++++++___ Displaying Environment ___++++++++++++++

safe a,safe b,safe c,
safe d,fire e,safe f,
safe g,safe h,fire i,
no fire in room d ✅
++++++++++++++___ Displaying Environment ___++++++++++++++
```