- **Engineer: Shaun Pritchard**
- **Date: 1/27/2023**
- **Task: Reflex,goal, and reflex model agents**
- **Artifical Intellgence CAP 6635**

```
!pip install·messagebox
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: messagebox in /usr/local/lib/python3.8/dist-packages (0.1.0)
```

```
class Node:
    def __init__(self, x, y, parent=None):
        self.x = x
        self.y = y
        self.parent = parent

    def set_x(self, x):
        self.x = x

    def set_y(self, y):
        self.y = y

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y

    def set_parent(self, parent):
        self.parent = parent

    def get_parent(self):
        return self.parent
```

```
import matplotlib.pyplot as plt
import random
# from Node import *
from tkinter import messagebox
from copy import copy, deepcopy
import matplotlib.pyplot as plt
import numpy as np
import random
```

Model 1

```
# CAP 6635 Artificial Intelligence; X. Zhu; 01/13/2022
# model.py” is a reflex agent with a model to ensure agent walking through all locations
# Code adopted from https://github.com/mawippel/python-vacuum. Changes are made to reflect agent moves following predef
# Reflex Vacuum Cleaner Agent. Agent makes random move (-1 for each move, and +10 for clean a spot)

# 0 -> clean
# 1 -> wall
# 2 -> dirt
# The original matrix contains probablty values which will be used to generte the environment.
# if you want to make a spot to have dirt for sure, set the value as 1.0
# if you do NOT want to make a spot to have dirt, set the value as 0
matrix = [
    [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
    [1.0, 0.1, 0.1, 0.1, 0.4, 0.4, 0.1 ,1.0],
    [1.0, 0.1, 0.1, 0.1, 0.6, 0.5, 0.1 ,1.0],
    [1.0, 0.1, 0.4, 0.1, 0.1, 0.1, 0.1 ,1.0],
    [1.0, 0.4, 0.6, 0.4, 0.1, 0.1, 0.1 ,1.0],
    [1.0, 0.1, 0.4, 0.1, 0.1, 0.1, 0.1 ,1.0],
    [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.4 ,1.0]
```

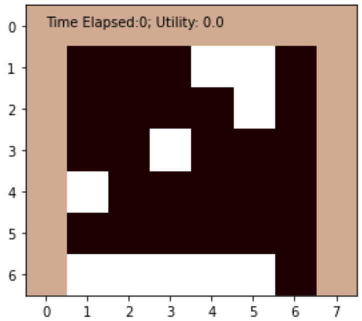
```
# Actions Matrix -> represents the action for each position
# Actions = up (0), down (1), left (2), right (3), clean(4), end (5)
actionsMatrix = [
    [9, 9, 9, 9, 9, 9, 9, 9],
    [9, 1, 3, 1, 3, 1, 0, 9],
    [9, 1, 0, 1, 0, 1, 0, 9],
    [9, 1, 0, 1, 0, 1, 0, 9],
    [9, 1, 0, 1, 0, 1, 1, 9],
    [9, 3, 0, 3, 0, 5, 0, 9],
    [9, 9, 9, 9, 9, 9, 9, 9]
]

```

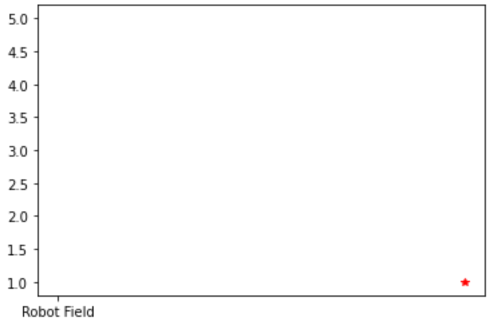
```
def renderMatrix(matrix,x,y,utility,timeElapsed):
    plt.text(0,0,"Time Elapsed:%d; Utility: %.1f"%(timeElapsed,utility))
    plt.imshow(matrix, 'pink')
    plt.show(block=False)
    plt.plot(y,x,'r:',linewidth=1)
    plt.plot(y[len(y)-1], x[len(x)-1], '*r', 'Robot Field', 5)
    plt.pause(0.5)
    plt.clf()

def createWorld(m):
    for mI in range(1, 7):
        for aI in range(1, 7):
            if (random.random()<m[mI][aI]):
                m[mI][aI] = 2
            else:
                m[mI][aI] = 0
    #renderMatrix(matrix)

def findNextAction(x, y):
  return actionsMatrix[x][y]

# decides which action will be done
# Actions = up (0), down (1), left (2), right (3), clean(4)
def modelAgentRobot(x, y):
  if (matrix[x][y] == 2): # if it's dirty, return the clean action
    return 4
  return findNextAction(x, y)

def checkDirtSpots(matrix):
  x=len(matrix)
  totalones=2*x+(x-2)*2
  sum=np.sum(matrix)-totalones*2
  return(sum)
```

```
def main():
  createWorld(matrix)
  print("Environment (beginning)\r\n")
  print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in matrix]))
  # The robot always starts at matrix[1][1]
  currLine = 1
  currCol = 1
  Lines=[]
  Cols=[]
  Lines.append(currLine)
  Cols.append(currCol)
  utility=0
  timeElapsed=0
  renderMatrix(matrix,Lines,Cols,utility,timeElapsed)
  totalDirt=checkDirtSpots(matrix)
  print(totalDirt)
  while True:
    action = modelAgentRobot(currLine, currCol)
    if (action == 0): # go up
```

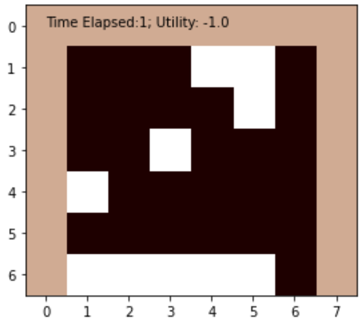
```
      print("up")
      currLine = currLine - 1 # remove 1 line
      utility=utility-1
    elif (action == 1): # go down
      print("down")
      currLine = currLine + 1
      utility=utility-1
    elif (action == 2): # go left
      print("left")
      currCol = currCol - 1
      utility=utility-1
    elif (action == 3): # go right
      print("right")
      currCol = currCol + 1
      utility=utility-1
    elif (action == 4): # clean
      print("clean")
      matrix[currLine][currCol] = 0
      utility=utility+10
    else:
      print("end")
      break
    Lines.append(currLine)
    Cols.append(currCol)
    timeElapsed=timeElapsed+1
    renderMatrix(matrix,Lines,Cols,utility,timeElapsed)
  print("Environment (ending): %f\r\n"%utility)
  print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in matrix]))




if __name__ == "__main__":
  main()
```

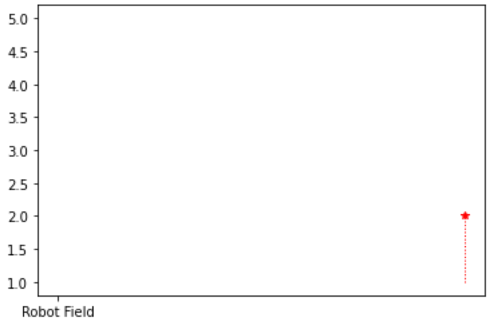
```
Environment (beginning)

1.0     1.0     1.0     1.0     1.0     1.0     1.0     1.0
1.0     0       0       0       2       2       0       1.0
1.0     0       0       0       0       2       0       1.0
1.0     0       0       2       0       0       0       1.0
1.0     2       0       0       0       0       0       1.0
1.0     0       0       0       0       0       0       1.0
1.0     2       2       2       2       2       0       1.0
```



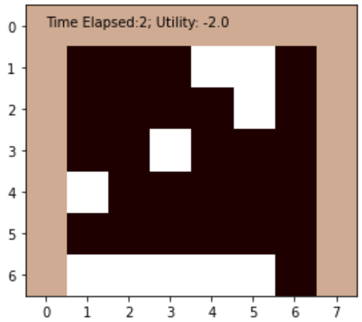


```
-8.0
down
```





```
down
```

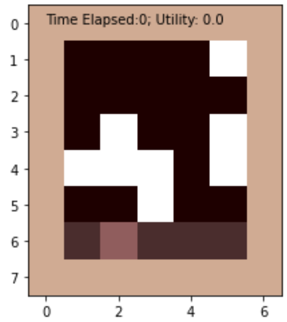


GOAL

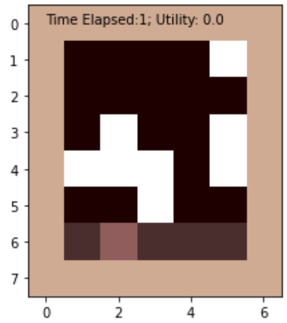
```
# CAP 6635 Artificial Intelligence; X. Zhu; 01/13/2022

# Code adopted from https://github.com/mawippel/python-vacuum. Changes are made to reflect agent moves following searc
# Goal Based Vacuum Cleaner Agent. Agent repetitively searches closest dirt, and walks to clean the dirt. After that, s
# dirt again. -1 for each move, and +10 for clean a dirt.
# -1 -> clean
# 0 -> wall
# 1 -> dirt
# The original matrix contains probablty values which will be used to generte the environment.
# if you want to make a spot to have dirt for sure, set the value as 0.0
# if you do NOT want to make a spot to have dirt, set the value as -1
matrix = [
    [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
    [1.0, 0.1, 0.1, 0.1, 0.4, 0.4, 1.0],
    [1.0, 0.1, 0.1, 0.1, 0.6, 0.1, 1.0],
    [1.0, 0.1, 0.4, 0.1, 0.1, 0.1, 1.0],
    [1.0, 0.4, 0.6, 0.4, 0.1, 0.1, 1.0],
    [1.0, 0.1, 0.4, 0.1, 0.1, 0.1, 1.0],
    [1.0, 0.1, 0.4, 0.1, 0.1, 0.1, 1.0],
    [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
]
presentationMatrix = [
    [1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1],
]

# The robot always starts at matrix[1][1]
currLine = 1
currCol = 1
stack = [Node(1, 1)]
solution = [Node(1, 1)]
process_map = []


def mapNotClean():
    for i in range(1, len(matrix) - 1):
        for j in range(1, len(matrix[i]) - 1):
            if (matrix[i][j] == 2):
                return True
    return False


#def renderMatrix(matrix):
#    plt.imshow(matrix, 'pink')
#    plt.show(block=False)
#    plt.plot(currCol, currLine, '*r', 'LineWidth', 5)
#    plt.pause(0.5)
#    plt.clf()
def renderMatrix(matrix,x,y,utility,timeElapsed):
    plt.text(0,0,"Time Elapsed:%d; Utility: %.1f"%(timeElapsed,utility))
    plt.imshow(matrix, 'pink')
    plt.show(block=False)
    plt.plot(y,x,'r:',linewidth=1)
    plt.plot(y[len(y)-1], x[len(x)-1], '*r', 'Robot Field', 5)
    plt.pause(0.5)
    plt.clf()
```

```



def createWorld(m):
    for mI in range(1, 6):
        for aI in range(1, 6):
#            if (mI == 1 and aI == 1):
#                continue
#            number = random.randint(0, 3)
#            m[mI][aI] = 2 if number == 1 else 0
            if (random.random()<m[mI][aI]):
                m[mI][aI] = 2
            else:
                m[mI][aI] = 0
    #renderMatrix(matrix)
    global process_map
    global presentationMatrix
    process_map = deepcopy(matrix)
    presentationMatrix = deepcopy(matrix)


def hasPosition(x, y):
    if (matrix[x][y] == 1):
        return False
    return True


def lookLeft(x, y, node):
    if (hasPosition(x - 1, y)):
        new_node = Node(x - 1, y, node)
        if (process_map[x - 1][y] == 2):
            return new_node
        if (process_map[x - 1][y] != 4):
            stack.append(new_node)
            process_map[x - 1][y] = 4


def lookRight(x, y, node):
    if (hasPosition(x + 1, y)):
        new_node = Node(x + 1, y, node)
        if (process_map[x + 1][y] == 2):
            return new_node
        if (process_map[x + 1][y] != 4):
            stack.append(new_node)
            process_map[x + 1][y] = 4


def lookAbove(x, y, node):
    if (hasPosition(x, y - 1)):
        new_node = Node(x, y - 1, node)
        if (process_map[x][y - 1] == 2):
            return new_node
        if (process_map[x][y - 1] != 4):
            stack.append(new_node)
            process_map[x][y - 1] = 4


def lookDown(x, y, node):
    if (hasPosition(x, y + 1)):
        new_node = Node(x, y + 1, node)
        if (process_map[x][y + 1] == 2):
            return new_node
        if (process_map[x][y + 1] != 4):
            stack.append(new_node)
            process_map[x][y + 1] = 4


def discoverPath():
    while (len(stack) != 0):
        node = stack.pop(0)
        x = node.get_x()
        y = node.get_y()

        auxNode = lookLeft(x, y, node)
        if (auxNode):
            return auxNode

        auxNode = lookAbove(x, y, node)
        if (auxNode):
            return auxNode

        auxNode = lookRight(x, y, node)
        if (auxNode):
            return auxNode

        auxNode = lookDown(x, y, node)
        if (auxNode):
            return auxNode


def main():
    global matrix
    global process_map
    global stack
    global currCol
    global currLine
    createWorld(matrix)

    print("Environment (beginning)\r\n")
    print('\n'.join(['\t'.join([str(cell) for cell in row]) for row in matrix]))

    # The robot always starts at matrix[1][1]
    currLine = 1
    currCol = 1
    utility=0
    Lines=[]
    Cols=[]
    Lines.append(currLine)
    Cols.append(currCol)
    timeElapsed=0
    renderMatrix(matrix,Lines,Cols,utility,timeElapsed)

    while (mapNotClean()):
        path = discoverPath()
        x = path.get_x()
        y = path.get_y()

        aux_list = []
        while (path.get_parent() is not None):
            process_map[path.get_x()][path.get_y()] = 3
            aux_list.append(path)
            path = path.get_parent()
        aux_list.reverse()
        solution.extend(aux_list)

        matrix[x][y] = 0
        stack = [Node(x, y)]
        process_map = deepcopy(matrix)

    for path in solution:
        currCol = path.get_y()
        currLine = path.get_x()
        Lines.append(currLine)
        Cols.append(currCol)
        timeElapsed=timeElapsed+1
        renderMatrix(presentationMatrix,Lines,Cols,utility,timeElapsed)
        if (presentationMatrix[currLine][currCol] == 2):
            presentationMatrix[currLine][currCol] = 0
            utility=utility+10
        else:
            utility=utility-1
    timeElapsed=timeElapsed+1
    renderMatrix(presentationMatrix,Lines,Cols,utility,timeElapsed)
    messagebox.showinfo(
        "Summary", "Total traveled %s steps" % (len(solution) - 1))


if __name__ == "__main__":
    main()
```

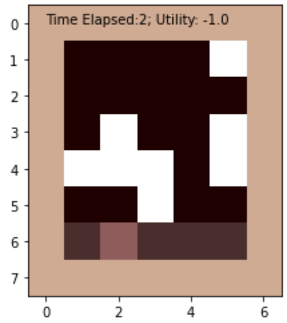
```
Environment (beginning)

1.0     1.0     1.0     1.0     1.0     1.0     1.0
1.0     0       0       0       0       2       1.0
1.0     0       0       0       0       0       1.0
1.0     0       2       0       0       2       1.0
1.0     2       2       2       0       2       1.0
1.0     0       0       2       0       0       1.0
1.0     0.1     0.4     0.1     0.1     0.1     1.0
1.0     1.0     1.0     1.0     1.0     1.0     1.0
```

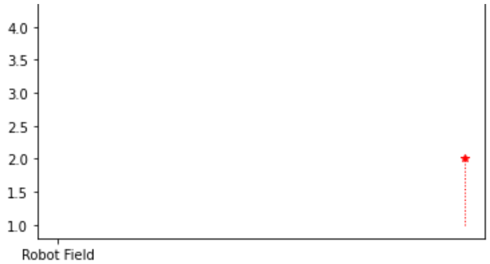


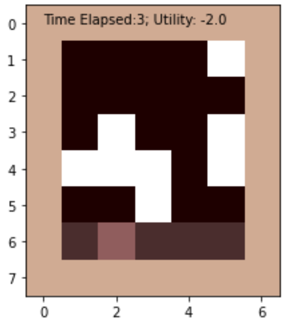
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field

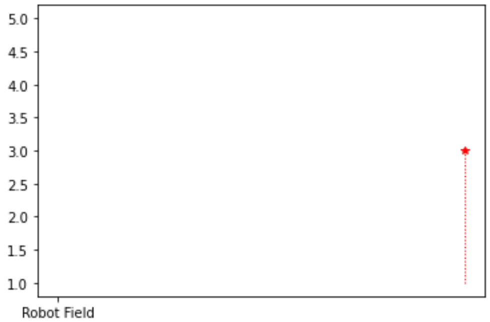


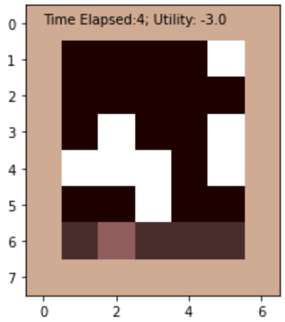
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field

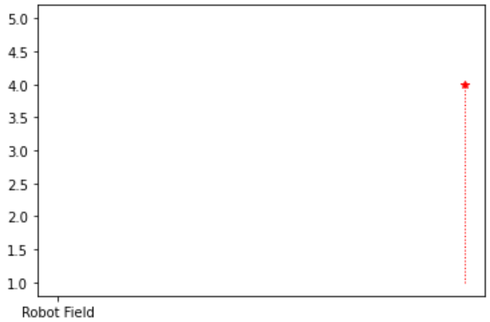


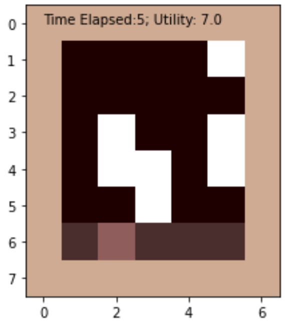

4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field


Time Elapsed:3; Utility: -2.0
0
1
2
3
4
5
6
7
0
2
4
6

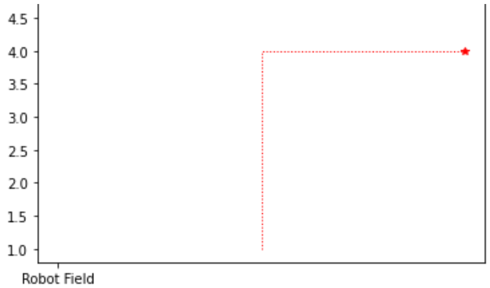

5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field

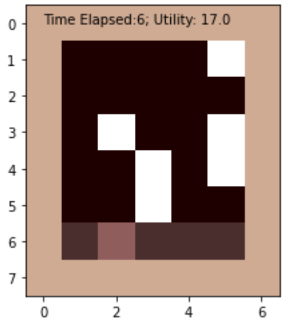

Time Elapsed:4; Utility: -3.0
0
1
2
3
4
5
6
7
0
2
4
6

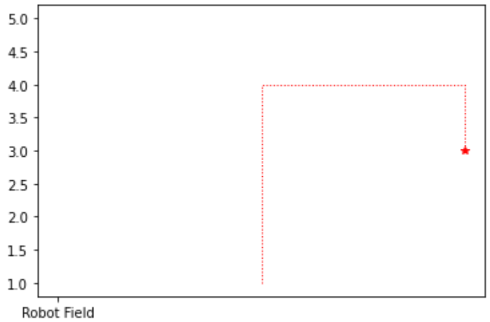

5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field

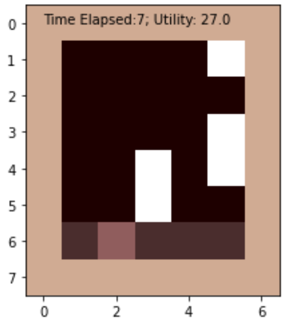

Time Elapsed:5; Utility: 7.0
0
1
2
3
4
5
6
7
0
2
4
6

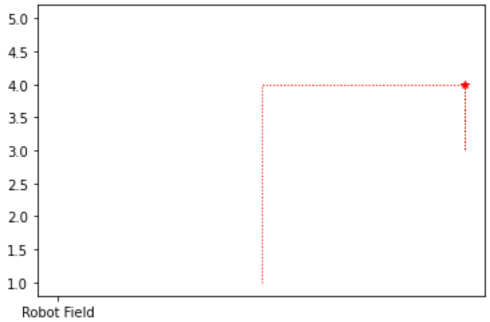
5.0

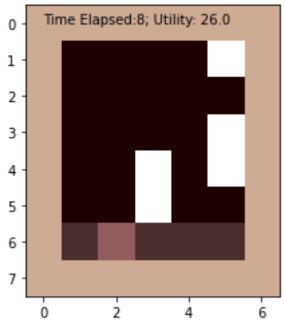

4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field


Time Elapsed:6; Utility: 17.0
0
1
2
3
4
5
6
7
0
2
4
6


5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field


Time Elapsed:7; Utility: 27.0
0
1
2
3
4
5
6
7
0
2
4
6


5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Robot Field


Time Elapsed:8; Utility: 26.0
0
1
2
3
4
5
6
7
0
2
4
6