# -\*- coding: utf-8 -\*-

"""

Created on Tue Mar 14 17:09:37 2023

@author: mothi

"""

import heapq

import numpy as np

import math

import cv2

import pygame

import time

import itertools

import threading

import sys

import matplotlib.pyplot as plt

"""

----------------------------------------------------------------

Loading animation

----------------------------------------------------------------

"""

is\_loading\_a\_star = False

is\_loading\_backtrack = False

def animate\_A\_star():

    for c in itertools.cycle(["⢿", "⣻", "⣽", "⣾", "⣷", "⣯", "⣟", "⡿"]):

        if is\_loading\_a\_star:

            break

        sys.stdout.write('\rRunning A\* Algorithm ' + c)

        sys.stdout.flush()

        time.sleep(0.1)

def animate\_Backtrack():

    for c in itertools.cycle(["⢿", "⣻", "⣽", "⣾", "⣷", "⣯", "⣟", "⡿"]):

        if is\_loading\_backtrack:

            break

        sys.stdout.write('\rBacktracking ' + c)

        sys.stdout.flush()

        time.sleep(0.1)

"""

----------------------------------------------------------------

 Creating obstacles in the map

----------------------------------------------------------------

"""

def get\_slope\_const(p1, p2):

    try:

        slope = (p2[1]-p1[1])/(p2[0] - p1[0])

        constant = (p2[1] - slope \* p2[0])

        return slope, constant

    except:

        return p1[1], p1[1]

def getObstacleCoord(mapWidth,mapHeight):

    coords=[]

    coords\_scaled=[]

    for i in range(mapWidth ):

        for j in range(mapHeight ):

            coords.append((i,j))

    #Scaling the obstacle to make sure we include the .5 thresold

    for i in range(mapWidth\*2 ):

        for j in range(mapHeight\*2 ):

            coords\_scaled.append((round(i/2),round(j/2)))

    obstacles= []

    obstacles\_scaled=[]

    clearance = 5 +5 #Celarance+Radius

    Hexa\_pt1 = (235.05, 162.5)

    Hexa\_pt2 = (300, 200)

    Hexa\_pt3 = (364.95, 162.5)

    Hexa\_pt4 = (364.95, 87.5)

    Hexa\_pt5 = (300, 50)

    Hexa\_pt6 = (235.05, 87.5)

    slope\_1\_2,const\_1\_2 = get\_slope\_const(Hexa\_pt1,Hexa\_pt2)

    slope\_2\_3,const\_2\_3 = get\_slope\_const(Hexa\_pt2, Hexa\_pt3)

    slope\_6\_5, const\_6\_5 = get\_slope\_const(Hexa\_pt6,Hexa\_pt5)

    slope\_5\_4, const\_5\_4 = get\_slope\_const(Hexa\_pt5, Hexa\_pt4)

    tri\_pt1 = (460,225)

    tri\_pt2 = (510,125)

    tri\_pt3 = (460,25)

    tri\_slope\_1\_2,tri\_const\_1\_2 = get\_slope\_const(tri\_pt1,tri\_pt2)

    tri\_slope\_2\_3,tri\_const\_2\_3 = get\_slope\_const(tri\_pt2,tri\_pt3)

    for pts in coords\_scaled:

        x, y = pts[0], pts[1]

        if x<= clearance or y<=clearance or x>= mapWidth-clearance or  y>= mapHeight-clearance:

            obstacles\_scaled.append((x,y))

        #Rectangle 1

        if x>100 -clearance and x<150 +clearance and y >150-clearance and y <250:

            obstacles\_scaled.append((x,y))

        #Rectangle 2

        if x>100-clearance and x<150+clearance and y >=0 and y <100+clearance:

            obstacles\_scaled.append((x,y))

        # Hexagon Obstacle

        if x > 235.05 -  clearance and x < 364.95 + clearance:

            if (y - slope\_1\_2\*x < const\_1\_2  + clearance) and  (y - slope\_2\_3\*x < const\_2\_3 + clearance) and  (y - slope\_6\_5\*x > const\_6\_5 - clearance) and  (y - slope\_5\_4\*x > const\_5\_4  - clearance)  :

                obstacles\_scaled.append((x,y))

        #Triangle

        if x>460-clearance and x<510 + clearance:

            if  (y - tri\_slope\_1\_2\*x < tri\_const\_1\_2 + clearance) and (y - tri\_slope\_2\_3\*x > tri\_const\_2\_3 - clearance)  :

                obstacles\_scaled.append((x,y))

    for pts in coords:

        x, y = pts[0], pts[1]

        if x<= clearance or y<=clearance or x>= mapWidth-clearance or  y>= mapHeight-clearance:

            obstacles.append((x,y))

        #Rectangle 1

        if x>100 -clearance and x<150 +clearance and y >150-clearance and y <250:

            obstacles.append((x,y))

        #Rectangle 2

        if x>100-clearance and x<150+clearance and y >=0 and y <100+clearance:

            obstacles.append((x,y))

        # Hexagon Obstacle

        if x > 235.05 -  clearance and x < 364.95 + clearance:

            if (y - slope\_1\_2\*x < const\_1\_2  + clearance) and  (y - slope\_2\_3\*x < const\_2\_3 + clearance) and  (y - slope\_6\_5\*x > const\_6\_5 - clearance) and  (y - slope\_5\_4\*x > const\_5\_4  - clearance)  :

                obstacles.append((x,y))

        #Triangle

        if x>460-clearance and x<510 + clearance:

            if  (y - tri\_slope\_1\_2\*x < tri\_const\_1\_2 + clearance) and (y - tri\_slope\_2\_3\*x > tri\_const\_2\_3 - clearance)  :

                obstacles.append((x,y))

    return obstacles\_scaled,obstacles

"""

----------------------------------------------------------------

Action set

----------------------------------------------------------------

"""

# We have created a funstion that can accomodate any angles

def action\_set(step\_size,coord, orientation,map\_width,map\_height):

    x = round((step\_size)\*np.cos(np.deg2rad(orientation)) + coord[0],2)

    y = round((step\_size)\*np.sin(np.deg2rad(orientation)) + coord[1],2)

    if x>=0 and x<=map\_width and y>=0 and y<= map\_height:

        return((x,y),True)

    else:

        return(coord,False)

'''

------------------------------------------------------------------

Get Neighbours

------------------------------------------------------------------

'''

def getGraph(coord,orientation,map\_width,map\_height,step\_size,ObstacleList):

    obs\_set = set(ObstacleList)

    costs = {}

    minusSixty,is\_minusSixty = action\_set(step\_size,coord, orientation - 60,map\_width,map\_height)#-60

    if is\_minusSixty and (minusSixty[0],minusSixty[1]) not in obs\_set:

        costs[(minusSixty[0],minusSixty[1],orientation-60)] = step\_size

    minusThirty,is\_minusThirty = action\_set(step\_size,coord, orientation - 30,map\_width,map\_height)#-30

    if is\_minusThirty and (minusThirty[0],minusThirty[1]) not in obs\_set:

        costs[(minusThirty[0],minusThirty[1],orientation-30)] = step\_size

    zero,is\_zero = action\_set(step\_size,coord, orientation + 0,map\_width,map\_height)

    if is\_zero and (zero[0],zero[1]) not in obs\_set:

        costs[(zero[0],zero[1],orientation)] = step\_size

    thirty,is\_thirty = action\_set(step\_size,coord, orientation + 30,map\_width,map\_height)#30

    if is\_thirty and (thirty[0],thirty[1]) not in obs\_set:

        costs[(thirty[0],thirty[1],orientation+30)] = step\_size

    sixty,is\_sixty = action\_set(step\_size,coord, orientation + 60,map\_width,map\_height)#60

    if is\_sixty and (sixty[0],sixty[1]) not in obs\_set:

        costs[(sixty[0],sixty[1],orientation+60)] = step\_size

    return costs

def A\_Star(start,goal,map\_width,map\_height,step\_size,ObstacleList):

    cost\_list = {}

    closed\_list = []

    #Contains the parent node and the cost taken to reach the current node

    parent\_index = {}

    print("start Point :",start)

    print("Goal Point : ",goal)

    print("Step Size : ", step\_size)

    cost\_list[start]=0

    open\_list = [(0,start)]

    Goal\_Reached = False

    count=0

    loading = threading.Thread(target=animate\_A\_star)

    loading.start()

    global is\_loading\_a\_star

    #Converting to set for faster checking

    obs\_set = set(ObstacleList)

    while len(open\_list)>0 and Goal\_Reached == False:

        count = count+1

        totalC, parent\_coord = heapq.heappop(open\_list)

        parent\_position = (parent\_coord[0],parent\_coord[1])

        orientation = parent\_coord[2]

        neighbours = getGraph(parent\_position,orientation,map\_width,map\_height,step\_size,obstacle\_scaled)

        if parent\_position not in obs\_set:

            for key, cost in neighbours.items():

                cost\_list[key]=math.inf

            for coord, cost in neighbours.items():

                if(coord not in  closed\_list) and (coord not in ObstacleList):

                    coord\_round = (round(coord[0]),round(coord[1]),coord[2])

                    closed\_list.append(coord\_round)

                    Cost2Come = cost

                    Cost2Go = math. dist((coord[0],coord[1]),(goal[0],goal[1]))  # h(n)

                    TotalCost = Cost2Come + Cost2Go   # f(n)

                    if TotalCost < cost\_list[coord] or coord not in open\_list :

                        parent\_index[coord\_round]={}

                        parent\_index[coord\_round][TotalCost] = parent\_coord

                        cost\_list[coord\_round]=TotalCost

                        heapq.heappush(open\_list, (TotalCost, coord\_round))

                    #The thersold is set according to the step size to reach closest to goal with few steps

                    if ((coord\_round[0]-goal[0])\*\*2 + (coord\_round[1]-goal[1])\*\*2 <= (step\_size)\*\*2) and coord\_round[2]==goal[2] :

                        print("\nFinal Node :",coord\_round)

                        print('GOAL  Reached !!')

                        print("Total Cost :  ",TotalCost)

                        Goal\_Reached = True

                        time.sleep(0)

                        is\_loading\_a\_star = True

                        return parent\_index,closed\_list,coord\_round,True

    return parent\_index,closed\_list,False

"""

 ----------------------------------------------------------------

Backtracking

 ----------------------------------------------------------------

 """

def get\_Backtrack(parent\_index,goal,start):

    back\_track = []

    current= start

    back\_track.append(current)

    is\_goal\_reached = False

    #For loading icon

    global is\_loading\_backtrack

    loading = threading.Thread(target=animate\_Backtrack)

    loading.start()

    while is\_goal\_reached == False:

        for coord,parent\_cost in parent\_index.items():

            for cost,parent in parent\_cost.items():

                if coord==current:

                    if parent not in back\_track:

                        back\_track.append(current)

                    current = parent

                    if parent == goal:

                        is\_goal\_reached = True

                        time.sleep(1)

                        is\_loading\_backtrack = True

                        break

    back\_track.append(goal)

    return back\_track

"""

----------------------------------------------------------------

Visualization

----------------------------------------------------------------

"""

def visualize\_map(map\_width,map\_height,obstacle\_scaled,obstacle\_cord,closed\_list,back\_track\_coord):

    obstacle\_map = np.zeros((map\_width\*2+1,map\_height\*2+1,3),np.uint8)

    obstacle\_map[obstacle\_scaled\*2]=255

    obs\_set = set(obstacle\_scaled)

    pygame.init()

    gameDisplay = pygame.display.set\_mode((map\_width\*2,map\_height\*2))

    pygame.surfarray.make\_surface(obstacle\_map)

    pygame.display.set\_caption('A Star Algorithm')

    gameDisplay.fill((0,0,0))

    #Adding obstacle to animation

    for coords in obs\_set:

        pygame.draw.rect(gameDisplay, (255,0,255), [coords[0]\*2 ,abs(250-coords[1])\*2,1,1])

        pygame.display.flip()

    #Adding explored region/ visited nodes

    for coords in closed\_list:

        pygame.time.wait(10)

        pygame.draw.rect(gameDisplay, (255,255,0), [coords[0]\*2 ,abs(250-coords[1])\*2,1,1])

        pygame.display.flip()

    #Adding back track path

    for coords in back\_track\_coord:

        pygame.time.wait(10)

        pygame.draw.rect(gameDisplay, (255,0,0), [coords[0]\*2,abs(250-coords[1])\*2,1,1])

        pygame.display.flip()

    pygame.quit()

    obstacle\_map\_3d =  np.zeros((map\_height+1,map\_width+1,3),np.uint8)

    for x,y in obstacle\_cord:

        obstacle\_map\_3d[(y,x)]=[255,0,255]

    for x,y,d in back\_track\_coord:

        obstacle\_map\_3d[(250-y,x)]=[0,0,255]

    backtrack\_map = cv2.resize(obstacle\_map\_3d,(map\_width\*3,map\_height\*3))

    cv2.imshow('Backtrack Path',backtrack\_map)

    cv2.waitKey(0)

    cv2.destroyAllWindows()

"""

----------------------------------------------------------------

Main Function

----------------------------------------------------------------

"""

obstacle\_scaled,obstacle\_cord =getObstacleCoord(600,250)

map\_width=600

map\_height=250

step\_size=1

start = (11,11,0)

goal = (400,100,60)

#Showing the map

x, y = [], []

for i in obstacle\_scaled:

    x.append(i[0])

    y.append(i[1])

plt.scatter(x,y,s=0.1,c='red')

plt.axis([0,600,0,250])

plt.title('Obstacles Map')

plt.grid(which='both')

plt.show()

try:

    while True:

        x\_s = int(input("Please enter the x coordinate of start  : "))

        y\_s = int(input("Please enter the y coordinate of start  : "))

        orientation\_s =int(input("Please enter the orientation of start  : "))

        x\_g = int(input("Please enter the x coordinate of goal  : "))

        y\_g = int(input("Please enter the y coordinate of goal  : "))

        orientation\_g =int(input("Please enter the orientation of goal  : "))

        step\_size = int(input("Please enter the step size between 1-10 (Warning!!! The goal threshold might increase when step size is high to get the closest solution quickly): "))

        radius = int(input("Please enter the radius of robot (sum of clearance + radius should be 10): "))

        clearance = int(input("Please enter the clearance of robot (sum of clearance + radius should be 10): "))

        if radius+ clearance !=10:

            print("Please enter clearance and radius summing up to 10  ")

        if step\_size< 1 and step\_size>10:

            print("Please enter a valid step size")

            continue

        if(x\_s>=map\_width or x\_g>=map\_width or y\_g>=map\_height or y\_g>=map\_height or x\_s<0 or x\_g<0 or y\_g<0 or y\_g<0):

            print("Please enter a value for x betweenn 0-599 and y between 0-249")

            continue

        elif((x\_s,y\_s)  in obstacle\_cord ) or ((x\_g,y\_g) in obstacle\_cord ):

            print("The input entered is on the obstacle point, Please enter a valid input")

            continue

        elif orientation\_s not in (0,30,60,-30,-60) or orientation\_g not in (0,30,60,-30,-60):

            print("The input entered for orientation is not 30,60,-30,-60,0 , Please enter a valid input")

            continue

        else:

            start = (x\_s,y\_s,orientation\_s)

            goal =  (x\_g,y\_g,orientation\_g)

            break

    #since the triangle is obstructing the path to reach we are keeping a threshold for x (Please refer the map )

    if x\_s <460 :

        start\_time = time.time()

        parent\_index,closed\_list,goal\_new,isGoal = A\_Star(start,goal,map\_width,map\_height,3,obstacle\_scaled)

        if(isGoal):

            back\_track\_coord =get\_Backtrack(parent\_index,start,goal\_new)

            print("\nTime to Find Path: ",time.time() - start\_time, "seconds (Warning : Actual time can be less since the loading animation can add more time" )

            visualize\_map(map\_width,map\_height,obstacle\_scaled,obstacle\_cord,closed\_list,back\_track\_coord)

        else:

            print("Bactracking cannot be done")

    else :

        print("Goal cannot be reached")

except:

    print("You have entered an invalid output please Run the program again")

print("Program Executed ")