```python
# -*- 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_scal
ed)
        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('Dijkstra 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,clos
ed_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 ")
```