

```
# -*- coding: utf-8 -*-
```

```
''''
```

```
Created on Sun Mar 5 18:39:16 2023
```

```
@author: mothi
```

```
''''
```

```
import numpy as np
```

```
import heapq
```

```
import time
```

```
import pygame
```

```
from tqdm import tqdm
```

```
''''
```

```
-----
```

```
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=[]

    for i in range(mapWidth + 1):
        for j in range(mapHeight + 1):
            coords.append((i,j))

    obstacles= []
    clearance = 5

    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:

```

```

x, y = pts[0], pts[1]

#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

```

```

"""

```

```

-----

```

Action Set

```

-----

```

```

"""

```

```

def up(coords,map_width,map_height):

```

```
if coords[1]<map_height-1:
    coords= (coords[0] , coords[1]+1)
    return(coords,True)
```

```
else:
    return(coords,False)
```

```
def upRight(coords,map_width,map_height): #DONE
```

```
if coords[0]<map_width-1 and coords[1]<map_height-1:
    coords= (coords[0]+1 , coords[1]+1)
    return(coords,True)
```

```
else:
    return(coords,False)
```

```
def right(coords,map_width,map_height):
```

```
if coords[0]<map_width-1:
    coords= (coords[0]+1 , coords[1])
    return(coords,True)
```

```
else:
    return(coords,False)
```

```
def downRight(coords,map_width,map_height): #DONE
```

```
if coords[0]<map_width-1 and coords[1]>0:
    coords= (coords[0]+1 , coords[1]-1)
    return(coords,True)
```

else:

return(coords,False)

def down(coords,map_width,map_height):

if coords[1]>0:

coords= (coords[0] , coords[1]-1)

return(coords,True)

else:

return(coords,False)

def downLeft(coords,map_width,map_height): #DONE

if coords[0]>0 and coords[1]>0:

coords= (coords[0]-1 , coords[1]-1)

return(coords,True)

else:

return(coords,False)

def left(coords,map_width,map_height):

if coords[0]>0:

coords= (coords[0]-1 , coords[1])

return(coords,True)

else:

return(coords,False)

def upLeft(coords,map_width,map_height):

```
if coords[0]>0 and coords[1]<map_height-1:
```

```
    coords= (coords[0]-1 , coords[1]+1)
```

```
    return(coords,True)
```

```
else:
```

```
    return(coords,False)
```

```
"""
```

```
-----
```

Get Cost Map for the map width and height

```
-----
```

```
"""
```

```
def get_Map_Cost (map_width,map_height,ObstacleList):
```

```
    graph_map ={}
```

```
    cost_map={}
```

```
    obs_set = set(ObstacleList)
```

```
    #Creating the Graph with nodes
```

```
    for i in tqdm(range(map_width-1,-1,-1)):
```

```
        for j in range(map_height-1,-1,-1):
```

```
            if (i,j) not in ObstacleList:
```

```
                possible_action=[]
```

```
                up_action, is_up = up((i,j),map_width,map_height)
```

```
                down_action, is_down = down((i,j),map_width,map_height)
```

```
                left_action, is_left = left((i,j),map_width,map_height)
```

```
                right_action, is_right = right((i,j),map_width,map_height)
```

```
up_right_action, is_up_right = upRight((i,j),map_width,map_height)
up_left_action, is_up_left = upLeft((i,j),map_width,map_height)
down_left_action, is_down_left = downLeft((i,j),map_width,map_height)
down_right_action, is_down_right = downRight((i,j),map_width,map_height)
```

```
if(is_up) and up_action not in obs_set :
    possible_action.append(up_action)
if(is_down) and down_action not in obs_set:
    possible_action.append(down_action)
if(is_left) and left_action not in obs_set:
    possible_action.append(left_action)
if(is_right) and right_action not in obs_set:
    possible_action.append(right_action)
if(is_up_right) and up_right_action not in obs_set:
    possible_action.append(up_right_action)
if(is_up_left) and up_left_action not in obs_set:
    possible_action.append(up_left_action)
if(is_down_left) and down_right_action not in obs_set:
    possible_action.append(down_right_action)
if(is_down_right) and down_left_action not in obs_set:
    possible_action.append(down_left_action)
if(len(possible_action)>1):
    graph_map[(i,j)]= possible_action[:]
```

#Adding cost to the different actions or states

```
for key,value in graph_map.items():
    cost_map[key]={}
```

```

    up_node,down_node,right_node,left_node =
up(key,map_width,map_height),down(key,map_width,map_height), left(key,map_width,map_height),
right(key,map_width,map_height)

    up_left_node, up_right_node, down_left_node, down_right_node =
upLeft(key,map_width,map_height),upRight(key,map_width,map_height),
downLeft(key,map_width,map_height), downRight(key,map_width,map_height)

    for coord in value:

        if coord in [up_node[0],left_node[0],right_node[0],down_node[0]]:

            cost_map[key][coord] = 1

        elif coord in [up_left_node[0],up_right_node[0],down_left_node[0],down_right_node[0]]:

            cost_map[key][coord]= 1.4

    return cost_map

```

"""

Dijkstra Algorithm

"""

```

def getDijkstra(cost_graph,start,goal):

    cost_list = {}

    closed_list = []

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

    parent_index = {}

    cost_list[start]=0

    closed_list.append(start)

    open_list = [(0,start)]

```



```

print("Running Dijkstra Algorithm.....")

for parent,child in cost_graph.items():
    cost_list[parent]= float('inf')

Goal_Reached = False

while len(open_list)>0 and Goal_Reached == False:

    total_cost,coord = heapq.heappop(open_list)

    for child_coord,current_cost in cost_graph[coord].items():
        cost2Come = total_cost + current_cost
        if cost2Come < cost_list[child_coord]:
            parent_index[child_coord] = {}
            #Assigning parent coord to each child coord
            parent_index[child_coord][cost2Come] = coord
            cost_list[child_coord]= cost2Come
            heapq.heappush(open_list, (cost2Come, child_coord))
            if child_coord not in closed_list:
                closed_list.append(child_coord)
                if child_coord == goal:
                    Goal_Reached = True
                    print("Goal Identified")
                    break

print("Total Cost Explored: ",cost2Come)
print("Backtracking.....")

```

```
return closed_list,parent_index
```

```
"""
```

```
-----
```

```
Backtrack
```

```
-----
```

```
"""
```

```
def get_Backtrack(parent_index,goal,start):
```

```
    back_track = []
```

```
    current= start
```

```
    back_track.append(current)
```

```
    is_goal_reached = False
```

```
    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
```

```
                        break
```

```
    return back_track
```

```
"""
```

```
-----  
Get all the list  
-----
```

```
"""
```

```
def get_closed_parentindex(map_width,map_height,start,goal,obstacle_list):
```

```
    cost_graph = get_Map_Cost(map_width,map_height,obstacle_list)
```

```
    parent_index = {}
```

```
    closed_list = []
```

```
    closed_list,parent_index= getDijkstra(cost_graph,start,goal)
```

```
    return(closed_list,parent_index)
```

```
"""
```

```
-----  
Visualization  
-----
```

```
"""
```

```
def visualize_map(map_width,map_height,obstacle_list,closed_list,back_track_coord):
```

```
    obstacle_map = np.zeros((map_width,map_height),np.uint8)
```

```
    obstacle_map[obstacle_list]=255
```

```
    pygame.init()
```

```
    gameDisplay = pygame.display.set_mode((map_width,map_height))
```

```
    pygame.surfarray.make_surface(obstacle_map)
```

```
    pygame.display.set_caption('Dijkstra Algorithm')
```

```

gameDisplay.fill((0,0,0))

#Adding explored region/ visited nodes

for coords in closed_list:

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

    pygame.display.flip()

#Addinf back track path

for coords in back_track_coord:

    pygame.time.wait(5)

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

    pygame.display.flip()


pygame.quit()

```

```

def main():

    map_width = 600

    map_height = 250

    obstacle_list = getObstacleCoord(map_width,map_height)


    start = (0,0)

    goal = (599,249)

    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 : "))

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

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

```

```
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 between 0-599 and y between 0-249")
```

```
    continue
```

```
elif((x_s,y_s) in obstacle_list ) or ((x_g,y_g) in obstacle_list ):
```

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

```
    continue
```

```
else:
```

```
    start = (x_s,y_s)
```

```
    goal = (x_g,y_g)
```

```
    break
```

```
start_time = time.time()
```

```
closed_list,parent_index= get_closed_parentindex(map_width,map_height,start,goal,obstacle_list)
```

```
back_track_coord={}
```

```
back_track_coord = get_Backtrack(parent_index,start,goal)
```

```
print("Time to Find Path: ",time.time() - start_time, "seconds")
```

```
visualize_map(map_width,map_height,obstacle_list,closed_list,back_track_coord)
```

```
# break
```

```
except:
```

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

main()