dijkstra_arshad_shaik.py

```python
# %% Import libraries
import heapq as hq
import numpy as np
import matplotlib.pyplot as plt
import math
import time

visited_x, visited_y = [], [] # list of current_node travel - initializing
anim_flag = True

# %% Function to find line passing through 2 Points
def lineFromPoints(P, Q):

    a = Q[1] - P[1]
    b = P[0] - Q[0]
    c = a*(P[0]) + b*(P[1])

    return (a, b, -c)

# %% Function to generate map
def generate_map(obstacle_space_rect, hexa, triangle, boundary_wall):
    # Creating the obstacle map
    ox,oy = [],[]
    obstacle = np.zeros((600, 250), dtype="uint8")
    # plt.imshow(obstacle)

    a = np.array([])
    b = np.array([])
    c = np.array([])

    # Equation of rectangle1
    for i in range(len(obstacle_space_rect[0])):
        if (i != (len(obstacle_space_rect[0])-1)):
            a_temp, b_temp, c_temp = lineFromPoints(obstacle_space_rect[0]
[i],obstacle_space_rect[0][i+1])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)
        else:
            a_temp, b_temp, c_temp = lineFromPoints(obstacle_space_rect[0]
[i],obstacle_space_rect[0][0])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)

    # Equation of rectangle2
    for i in range(len(obstacle_space_rect[1])):
        if (i != (len(obstacle_space_rect[1])-1)):
            a_temp, b_temp, c_temp = lineFromPoints(obstacle_space_rect[1]
[i],obstacle_space_rect[1][i+1])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)
```

```
        else:
            a_temp, b_temp, c_temp = lineFromPoints(obstacle_space_rect[1]
[i],obstacle_space_rect[1][0])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)

    # Equation of hexagon
    for i in range(len(hexa[0])):
        if (i != (len(hexa[0])-1)):
            a_temp, b_temp, c_temp = lineFromPoints(hexa[0][i],hexa[0][i+1])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)
        else:
            a_temp, b_temp, c_temp = lineFromPoints(hexa[0][i],hexa[0][0])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)

    # Equation of traingle
    for i in range(len(triangle[0])):
        if (i != (len(triangle[0])-1)):
            a_temp, b_temp, c_temp = lineFromPoints(triangle[0][i],triangle[0][i+1])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)
        else:
            a_temp, b_temp, c_temp = lineFromPoints(triangle[0][i],triangle[0][0])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)

    # Equation of boundary wall
    for i in range(len(boundary_wall[0])):
        if (i != (len(boundary_wall[0])-1)):
            a_temp, b_temp, c_temp = lineFromPoints(boundary_wall[0][i],boundary_wall[0][i+1])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)
        else:
            a_temp, b_temp, c_temp = lineFromPoints(boundary_wall[0][i],boundary_wall[0][0])
            a = np.append(a, a_temp)
            b = np.append(b, b_temp)
            c = np.append(c, c_temp)

    # print(" The line coefficeints of obstacle shapes (in the form - ax+by+c):\n", "Co-efficient
a:\n", a, "\n Co-efficient b:\n", b,  "\n Co-efficient c:\n", c)

    # Find out if each point is within the obstacle spacce or not
    for x in range(0, 600): # row
        for y in range(0, 250): # column

            f1 =  a[0]*x + b[0]*y + c[0]
            f2 =  a[1]*x + b[1]*y + c[1]
            f3 =  a[2]*x + b[2]*y + c[2]
            f4 =  a[3]*x + b[3]*y + c[3]
```

```
            f5 =  a[4]*x + b[4]*y + c[4]
            f6 =  a[5]*x + b[5]*y + c[5]
            f7 =  a[6]*x + b[6]*y + c[6]
            f8 =  a[7]*x + b[7]*y + c[7]

            f9 =   a[8]*x + b[8]*y + c[8]
            f10 =  a[9]*x + b[9]*y + c[9]
            f11 =  a[10]*x + b[10]*y + c[10]
            f12 =  a[11]*x + b[11]*y + c[11]
            f13 =  a[12]*x + b[12]*y + c[12]
            f14 =  a[13]*x + b[13]*y + c[13]

            f15 =  a[14]*x + b[14]*y + c[14]
            f16 =  a[15]*x + b[15]*y + c[15]
            f17 =  a[16]*x + b[16]*y + c[16]

            f18 =  a[17]*x + b[17]*y + c[17]
            f19 =  a[18]*x + b[18]*y + c[18]
            f20 =  a[19]*x + b[19]*y + c[19]
            f21 =  a[20]*x + b[20]*y + c[20]

            obs_space_rec1 = (f1 <= 0 and f3 <= 0 and f2 <=0 and f4<=0)
            obs_space_rec2 = (f5 <= 0 and f6 <= 0 and f7 <=0 and f8<=0)
            obs_space_hex = (f9 <= 0 and f10 <= 0 and f11 <=0 and f12<=0 and f13<=0 and f14<=0)
            obs_space_tri = (f15 >= 0 and f16 >= 0 and f17 >=0)
            obs_space_bndwl = (f18 >= 0 or f20 >=0 or f19 >= 0 or f21 >= 0 )

            # If a point is within the obstacle space, changee the color of that pixel
            if(obs_space_rec1 or obs_space_rec2 or obs_space_hex or obs_space_tri or 
obs_space_bndwl):
                obstacle[x][y] = 1
                ox.append(x)
                oy.append(y)

    return obstacle, ox, oy
# %% Define obstacle space
obstacle_space_rect =np.array([  [[95,250], [95,145], [155,145], [155, 250]],
                            [[95,105], [95, 0], [155, 0], [155, 105]],
                        ])
hexa = np.array([[[235,165],[235,85],[300,45],[365,85],[365, 165], [300, 205]]], np.int32)
triangle = np.array([[[455, 245],[515, 125],[455, 5]]])
boundary_wall = np.array([[(5, 5), (595, 5), (595, 245), (5, 245)]])

# Unfinflated obstacle space
obstacle_space_rect1 =np.array([  [[100,250], [100,140], [150,140], [150, 250]],
                            [[100,100], [100, 0], [150, 0], [150, 100]],
                        ])
hexa1 = np.array([[[240,87],[240,163],[300,200],[360,163],[360, 87], [300, 50]]], np.int32)
triangle1 = np.array([[[460, 225],[510, 125],[460, 25]]])
boundary_wall1 = np.array([[(5, 5), (595, 5), (595, 245), (5, 245)]])

# %% Generate map
map, ox, oy = generate_map(obstacle_space_rect, hexa, triangle, boundary_wall)
# map1, ox1, oy1 = generate_map(obstacle_space_rect1, hexa1, triangle1, boundary_wall1)
```

```python
# %% Function for Action set
def actions_set():
    steps = [[1,0,1],
             [0,1,1],
             [-1,0,1],
             [0,-1,1],
             [1,1,math.sqrt(2)],
             [1,-1,math.sqrt(2)],
             [-1,-1,math.sqrt(2)],
             [-1,1,math.sqrt(2)]]

    # print(steps)
    return steps
# %% Function for Dijkstra algorithm
def dijkstra(start,goal, obstacle):

    start_node = (0,start,None)
    goal_node = (0,goal,None)

    motion = actions_set()

    open_list = []
    closed_list = []

    hq.heappush(open_list,(start_node))
    obstacle[start_node[1][0]][start_node[1][1]] = 1

    while len(open_list)>0:
        current_node = hq.heappop(open_list)
        hq.heappush(closed_list,current_node)
        visited_x.append(current_node[1][0])
        visited_y.append(current_node[1][1])

        if (len(visited_x))%500 == 0:
            if anim_flag:
                    plt.plot(visited_x,visited_y, "vg")
                    plt.pause(0.00001)

        if current_node[1] == goal_node[1] :
            print('Goal reached')
            path = []
            length = len(closed_list)
            current_pos = closed_list[length-1][1]
            path.append(current_pos)
            parent = closed_list[length-1][2]
            while parent != None:
                for i in range(length):
                    X = closed_list[i]
                    if X[1] == parent:
                        parent = X[2]
                        current_pos = X[1]
                        path.append(current_pos)
            return path[::-1]

        neighbors = []
        for new_position in motion:
```

```
            # Fetch the current position
            node_position = (current_node[1][0] + new_position[0],
                             current_node[1][1] + new_position[1])
            node_position_cost = current_node[0] + new_position[2]

            node_parent = current_node[1]

            # Check if the node is in obstacle space
            if node_position[0] > (len(obstacle) - 5) or node_position[0] < 5 \
                or node_position[1] > (len(obstacle[0]) -5) or node_position[1] < 5:
                continue

            # Check free space
            if obstacle[node_position[0]][node_position[1]] != 0.0:
                continue

            #Creating cost_map
            obstacle[node_position[0]][node_position[1]] = 1

            # Creating a new node and also assigning a parent
            new_node = (node_position_cost,node_position,node_parent)
            neighbors.append(new_node)
            hq.heappush(open_list,(new_node))
# %% Taking user inputs
sx = int(input('Enter the Start Point (x coordinate) : '))
sy = int(input('Enter the Start Point (y coordinate): '))
gx = int(input('Enter the Goal Point (x coordinate): '))
gy = int(input('Enter the Goal Point (y coordinate): '))

start_time = time.time()
#plotting the obstacle map
plt.plot(ox,oy,".k")
plt.ylim((0,250))
plt.xlim((0,600))

start = (sx, sy)
goal = (gx, gy)

if start in zip(ox,oy):
    print('Start node is in obstacle space.Please select another node.')
elif goal in zip(ox,oy) :
    print('Goal node is in obstacle space .Please select another node.')
else:
    path = dijkstra(start,goal, map)
    if path == None:
        print('Goal node is in obstacle space.Please select another node.')
    else:
        pathx = [path[i][0] for i in range(len(path))]
        pathy = [path[i][1] for i in range(len(path))]

        plt.plot(goal[0], goal[1], "v")
        plt.plot(visited_x,visited_y, "vg")
        plt.plot(pathx,pathy,"-r", linewidth=3)
        plt.text(start[0], start[1], 'Starting Point', color='red', fontsize=15)
        plt.plot(start[0], start[1], marker="^", markerfacecolor='blue',markersize=10)
```

```
        plt.text(goal[0], goal[1], 'Goal Reached with Shortest Path!!', color='red', fontsize=15)
        plt.plot(goal[0], goal[1], marker="^", markerfacecolor='cyan',markersize=10)
        # manager = plt.get_current_fig_manager()
        # manager.full_screen_toggle()
        plt.show()
        end_time = time.time()
        print('time elapsed:',abs(end_time - start_time))
```