```python
from queue import PriorityQueue
from matplotlib.patches import Polygon

x_max = 600
y_max = 250

hexagon1 = [300, 50]
hexagon2 = [364, 87]
hexagon3 = [364, 162]
hexagon4 = [300, 200]
hexagon5 = [235, 162]
hexagon6 = [235, 87]

hexagon1_clearance = [300, 45]
hexagon2_clearance = [370, 85]
hexagon3_clearance = [370, 165]
hexagon4_clearance = [300, 205]
hexagon5_clearance = [230, 165]
hexagon6_clearance = [230, 85]

rectangle_1_1 = [100, 150]
rectangle_1_2 = [150, 150]
rectangle_1_3 = [150, 250]
rectangle_1_4 = [100, 250]

rectangle_1_1_clearance = [95,145]
rectangle_1_2_clearance = [155,145]
rectangle_1_3_clearance = [155, 250]
rectangle_1_4_clearance = [95, 250]

rectangle_2_1 = [100, 0]
rectangle_2_2 = [150, 0]
rectangle_2_3 = [150, 100]
rectangle_2_4 = [100, 100]

rectangle_2_1_clearance = [95,0]
rectangle_2_2_clearance = [155,0]
rectangle_2_3_clearance = [155, 105]
rectangle_2_4_clearance = [95, 105]

triangle1 = [460,25]
triangle2 = [510, 125]
triangle3 = [460, 225]

triangle1_clearance = [455, 5]
triangle2_clearance = [515, 125]
triangle3_clearance = [455, 245]

def get_line_equation(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    if x2==x1:
        return [x1]
    slope =(y2 - y1)/(x2 - x1)
    intercept = y1-(slope*x1)
    return [slope, intercept]

def inside_rectangle1(point):
    x, y = point
    bl = [95,145]
    ur = [155,250]
    if((x>=bl[0] and x<=ur[0]) and (y>=bl[1] and y<=ur[1])):
        return True
    return False

def inside_rectangle(point):
    x, y = point
    bl = [95,0]
    ur = [155,105]
    if((x>=bl[0] and x<=ur[0]) and (y>=bl[1] and y<=ur[1])):
        return True
    return False

def inside_hexagon(point):
    x, y = point
    equation1 = get_line_equation(hexagon1_clearance, hexagon2_clearance)
    l1_flag = y - equation1[0]*x - equation1[1]<=0
    l2_flag = x<=hexagon2_clearance[0]
    equation3 = get_line_equation(hexagon3_clearance, hexagon4_clearance)
    l3_flag = y - equation3[0]*x - equation3[1]>=0
    equation4 = get_line_equation(hexagon4_clearance, hexagon5_clearance)
    l4_flag = y - equation4[0]*x - equation4[1]>=0
    equation5 = get_line_equation(hexagon5_clearance, hexagon6_clearance)
    l5_flag = x>=hexagon5_clearance[0]
    equation6 = get_line_equation(hexagon6_clearance, hexagon1_clearance)
    l6_flag = y - equation6[0]*x - equation6[1]<=0

    flag = l1_flag and l2_flag and l3_flag and l4_flag and l5_flag and l6_flag
    return flag

def inside_triangle(point):
    x, y = point
    l1_flag = x >=455
    equation2 = get_line_equation(triangle3_clearance, triangle2_clearance)
    l2_flag = y - equation2[0]*x - equation2[1]<=0
    equation3 = get_line_equation(triangle2_clearance, triangle1_clearance)
    l3_flag = y - equation3[0]*x - equation3[1]>=0
    flag = l1_flag and l2_flag and l3_flag
    return flag

def check_obstacle(current_node):
    x, y = current_node
    if (x > x_max) or (y > y_max) or (x < 0) or (y < 0):
        return False
    if(inside_hexagon(current_node) or inside_rectangle1(current_node) or inside_rectangle(current_node) or inside_triangle(current_node)):
        return False
    return True

def get_shapes():
    hexagon_clearance = Polygon([hexagon1_clearance, hexagon2_clearance, hexagon3_clearance, hexagon4_clearance, hexagon5_clearance, hexagon6_clearance], facecolor = 'r')
    hexagon = Polygon([hexagon1, hexagon2, hexagon3, hexagon4, hexagon5, hexagon6], facecolor = 'b')
```

```python
    rectangle_1_clearance = Polygon([rectangle_1_1_clearance, rectangle_1_2_clearance, rectangle_1_3_clearance, rectangle_1_4_clearance], facecolor = 'r')
    rectangle_1 = Polygon([rectangle_1_1, rectangle_1_2, rectangle_1_3, rectangle_1_4], facecolor = 'b')
    rectangle_2_clearance = Polygon([rectangle_2_1_clearance, rectangle_2_2_clearance, rectangle_2_3_clearance, rectangle_2_4_clearance], facecolor = 'r')
    rectangle_2 = Polygon([rectangle_2_1, rectangle_2_2, rectangle_2_3, rectangle_2_4], facecolor = 'b')
    triangle_clearance = Polygon([triangle1_clearance, triangle2_clearance, triangle3_clearance], facecolor = 'r')
    triangle = Polygon([triangle1, triangle2, triangle3], facecolor = 'b')
    return (hexagon_clearance, hexagon, rectangle_1_clearance, rectangle_1, rectangle_2_clearance, rectangle_2, triangle_clearance, triangle)

def look_for_neighbors(current_node):
    neighbors = []
    x, y = current_node
    actions = [[1,0], [-1,0], [0,1], [0,-1], [1,1], [-1,1], [1,-1], [-1,-1]]
    for action in actions:
        if check_obstacle((x+action[0], y+action[1])):
            if action[0]!=0 and action[1]!=0:
                neighbors.append([(x+action[0], y+action[1]), 1.4])
            else:
                neighbors.append([(x+action[0], y+action[1]), 1])
    return neighbors

def dijkstra(source, destination):
    OpenList={}
    ClosedList={}

    OpenList[source] = 0
    visited=[]

    queue = PriorityQueue()
    queue.put((OpenList[source], source))
    path = []
    while queue:
        current_node = queue.get()[1]
        print("Current node : ", current_node)
        if current_node == destination:
            print("Found!!")
            temp = destination
            while(ClosedList[temp]!=source):
                path.append(ClosedList[temp])
                temp = ClosedList[temp]
            break
        if current_node not in visited:
            if(check_obstacle(current_node)):
                visited.append(current_node)
            neighbors = look_for_neighbors(current_node)
            for neighbor, cost in neighbors:
                node_cost = OpenList[current_node] + cost
                if (not OpenList.get(neighbor)):
                    OpenList[neighbor] = node_cost
                    ClosedList[neighbor] = current_node
                else:
                    if(node_cost<OpenList[current_node]):
                        OpenList[neighbor] = node_cost
                        ClosedList[neighbor] = current_node
                queue.put((OpenList[neighbor], neighbor))
    return path, visited
```

```python
# Youtube Video link - https://youtu.be/q70aM7p1eg4
# Github link - https://github.com/whosthemaan/Dijkstra_Path_Planning

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
from matplotlib.patches import Polygon
from Func import  check_obstacle, dijkstra, get_shapes

def init():
    scatter_plot.set_offsets([])
    return scatter_plot

def animation_func(index, last_visited):
    if(index<=last_visited):
        line.set_data(path_x[0], path_y[0])
        data = np.hstack((visited_x[:index,np.newaxis], visited_y[:index, np.newaxis]))
        scatter_plot.set_offsets(data)
    else:
        line.set_data(path_x[:(index-last_visited)*30], path_y[:(index-last_visited)*30])
    return scatter_plot, line

if __name__ == "__main__":
    source_x, source_y, destination_x, destination_y = 0, 0, 0, 0

    while True:
        print("Enter source points as X,Y :")
        source = input()
        source_x = int(source.split(",")[0])
        source_y = int(source.split(",")[1])

        print("Enter destination points as X,Y :")
        destination = input()
        destination_x = int(destination.split(",")[0])
        destination_y = int(destination.split(",")[1])

        if(check_obstacle([source_x, source_y]) and check_obstacle([destination_x, destination_y])):
            break
        else:
            print("Please enter valid input :")
            continue

    source = (source_x, source_y)
    destination = (destination_x, destination_y)
    print("Source is : ", source)
    print("Destination is : ", destination)

    fig = plt.figure()
    axis = plt.axes(xlim=(0, 600), ylim=(0, 250))
    axis.set_facecolor('k')

    path, visited = dijkstra(source, destination)
    path.append(source)
    path.insert(0, destination)

    visited_x, visited_y = (np.array(visited)[:,0], np.array(visited)[:,1])

    path_x, path_y = (np.array(path)[:,0], np.array(path)[:,1])

    line, = axis.plot(path_x, path_y, color = 'g')
    scatter_plot = axis.scatter([], [], s=2, color='w')

    shapes = get_shapes()
    for index in shapes:
        axis.add_patch(index)

    animator = FuncAnimation(fig, animation_func, frames = (len(visited)+len(path)) ,fargs=[len(visited)], interval=0.01, repeat=False, blit=True)
    plt.show()
```