~/Documents/GitHub/Astar-algorithm-implementation-for-a-non-holonomic-mobile-robot/a_star_DarshitMiteshkumar_Shivam.py

```python
# import the pygame module
import pygame as pyg
import numpy as np
#import queue module
from queue import PriorityQueue
import math
import time

# Function to find the points between 2 points
def bresenham_line(x0, y0, x1, y1):
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    sx = 1 if x0 < x1 else -1
    sy = 1 if y0 < y1 else -1
    err = dx - dy

    points = []
    while x0 != x1 or y0 != y1:
        points.append((x0, y0))
        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x0 += sx
        if e2 < dx:
            err += dx
            y0 += sy

    points.append((x1, y1))
    return points

# Function that appends all the exploration nodes to a list new_nodes
def move_robot(robot,curr_theta, costtocome):
    x,y=robot
    new_nodes = []
    for t in range(-60,61,30):
        x_t,y_t, t_t, c2g, c2c=  actions(x,y,t,curr_theta,costtocome)
        robot_position=(x_t,y_t)
        points=bresenham_line(x,y,x_t,y_t)

        new_nodes.append([c2g+c2c,c2c,c2g,robot_position,t_t,points])
    return new_nodes

# Function to calculate distance between two points
def euclidean(x,y,xg,yg):
    return math.dist((x,y),(xg,yg))

# Function that generated new postions and cost for robot exploration
def actions(x,y,t,ct,c2c):
    xr,yr=(round(x+step_size*np.cos(np.pi*(t+ct)/180)),round(y+step_size*np.sin(np.pi*(t
    c2g = euclidean(xr,yr,goal_x,goal_y)
```

```
        c2c = c2c+step_size
        return xr,yr,t+ct,c2g,c2c

# Function that appends the generated nodes to the open list if node not in open list
# If the list is in open list it updates the open list
def new_node(new_node_list):
    total_cost=new_node_list[0]
    cost_to_come=new_node_list[1]
    cost_to_goal=new_node_list[2]
    new_pos=new_node_list[3]
    t=new_node_list[4]
    x,y=new_pos
    points=new_node_list[5]
    if (((x>0 and x<600) and (y>0 and y<250))==True):
        if ( not(any(screen.get_at((a,b))!=white for a,b in points)) and screen.get_at((
white and not (new_pos in check_closed_list)):
            if not (new_pos in global_dict):
                global node_index
                node_index += 1
                global_dict[new_pos]=[total_cost,cost_to_come,cost_to_goal,node_index,in
                open_list.put(global_dict[new_pos])
                dict_vector[info[5]].append(new_pos)
            else:
                if (global_dict[new_pos][1]>cost_to_come):
                    global_dict[new_pos][4]=info[3]
                    global_dict[new_pos][1]=cost_to_come
                    global_dict[new_pos][0]=total_cost
                    global_dict[new_pos][-1]=t

# To input the step size from the user and validate the step size
print("ROBOT PARAMETERS")
print("***STEP SIZE OF THE ROBOT***")
while True:
    step_size=int(input("Enter the step size \n"))
    if (step_size<0):
        print("Invalid step size try again. Step size should be greater than zero")
        continue
    elif (step_size>20):
        print ("Invalid step size try again. Step size should be less than 20")
        continue
    else:
        break
# Taking input from user for clearance and radius of robot and defining the canvas
print("ROBOT CLEARANCE DIMENSIONS AND RADIUS. Enter valid dimensions between 0 to 50")
while (True):
    clr = int(input("Enter the clearance of the robot: "))
    radii = int(input("Enter the radius of the robot: "))
    if ((clr>0 and clr<50) and (radii>0 and radii<50)):
        print("Valid coordinates received")
        #Define the Surface Map
        screen = pyg.Surface((600, 250))
        #Define the rectangles which make the base map
        rect_color = (255, 255, 255)
        #Define the rectangle which makes the outer border
        rectangle1 = pyg.Rect(clr+radii, clr+radii, 600-2*(clr+radii), 250-2*(clr+radii)
```

```
        screen.fill((255,0,0))
        pyg.draw.rect(screen, rect_color, rectangle1)
        #Define the rectangle which makes the 2 rectangles
        bottom_rect_dim = [(150+radii+clr,150-radii-clr),(150+radii+clr,250),(100-radii-
radii-clr,150-radii-clr)]
        pyg.draw.polygon(screen, (255,0,0),bottom_rect_dim)
        top_rect_dim = [(100-radii-clr,0),(150+radii+clr,0),(150+radii+clr,100+radii+clr
clr,100+radii+clr)]
        pyg.draw.polygon(screen,(255,0,0),top_rect_dim)
        #Define the hexagon in the center with original dimensions
        hexagon_dim = [(300,50-radii-clr),(364.95190528+radii+clr,87.5-
((radii+clr)*np.tan(np.pi*30/180))),(364.95190528+radii+clr,162.5+((radii+clr)*np.tan(np
(300,200+radii+clr),(235.04809472-radii-clr,162.5+((radii+clr)*np.tan(np.pi*30/180))),(2
radii-clr,87.5-((radii+clr)*np.tan(np.pi*30/180)))]
        # pyg.draw.polygon(screen,(255,0,0),hexagon_dim)
        pyg.draw.polygon(screen,(255,0,0),hexagon_dim)
        #Define the triangle with the original dimensions
        triangle_dim = [(460-radii-clr,25-((radii+clr)/np.tan(np.pi*13.28/180))),(460.00
((radii+clr)/np.tan(np.pi*13.28/180))),(510+((radii+clr)/np.cos(np.pi*26.5650518/180)),1
        # pyg.draw.polygon(screen,(255,0,0),triangle_dim)
        pyg.draw.polygon(screen,(255,0,0), triangle_dim)
        white = (255,255,255)
        break
    else:
        print("Invalid coordinates received, Try again")
        continue

# Taking start position and goal position for the robot from the user and validating the
while True:
    try:
        print("Enter Robot start and goal coordinates. Ensure that theta values are mult
deg")
        start_x=int(input("Enter the starting x coordinate: "))
        start_y=int(input("Enter the starting y coordinate: "))
        start_theta=int(input("Enter the start theta orientation: "))
        if (start_theta%30!=0):
            print("Invalid Theta value try entering the coordinates again")
            continue
        goal_x=int(input("Enter the goal x coordinate:"))
        goal_y=int(input("Enter the goal y coordinate: "))
        goal_theta=int(input("Enter the goal theta orientation: \n"))
        if (goal_theta%30!=0):
            print("Invalid Theta value try entering the coordinates again")
            continue
        start_y=250-start_y
        goal_y=250-goal_y
        robot_start_position=(start_x,start_y)
        robot_goal_position=(goal_x,goal_y)
        if screen.get_at(robot_start_position) != white and screen.get_at(robot_goal_pos
            raise ValueError
        break
    except ValueError:
        print("Wrong input entered. Please enter an integer in correct range x(0,599) ar


ctc_node=0  # cost to come for start node
ctc_goal=math.dist((start_x,start_y),(goal_x,goal_y)) # cost to goal for the start node
```

```
    parent_node_index=None # Index for the parent node
    node_index=0 # Index of the current node
    closed_list={} # dictionary to store information about the current node
    check_closed_list={} # dictionary to store the nodes to check if nodes present in closed
    open_list=PriorityQueue() # list the store nodes and pop them according to priority
    info=[ctc_goal+ctc_node,ctc_node,ctc_goal,node_index,parent_node_index,robot_start_posit
# list to save all info of a node
    open_list.put(info)

    global_dict={} # global dictionary to reference all the information for nodes in the ope
update the information
    global_dict[robot_start_position]=[ctc_g
oal+ctc_node,ctc_node,ctc_goal,node_index,parent_node_index,robot_start_position,start_t


    start_time=time.time()# to store start time of the algorithm
    end_loop=0 # variable to break out of the loop
    dict_vector = {} # to save the node as key and it childs as values to draw nodes as vect

    # loop to explore the nodes and find the goal
    while True and end_loop!=1:
        # if the open list is empty means that no solution could be found
        if(open_list.empty()):
            print("No solution")
            goal_node=None
            break

        info=open_list.get()
        dict_vector[info[5]]=[]
        new_nodes=move_robot(info[5],info[6],info[1])
        for i in range(0,5):
            if(new_nodes[i][2]<=0.5):
                print("goal reached")
                closed_list[node_index+i+1]=[new_nodes[i][0]+info[1],new_nodes[i][1]+info[1]
[2],info[4],new_nodes[i][3],new_nodes[i][4]]
                goal_node=node_index+i+1
                end_loop=1
                break
            new_node(new_nodes[i])


        # append the node to node list
        closed_list[info[3]]=[info[0],info[1],info[2],info[4],info[5],info[6]]
        check_closed_list[info[5]]=None


    green=(0,255,0) # color for backtracking path line
    end_time=time.time() # to store end time for algorithm
    print("Total time taken for search:",end_time-start_time) # to check the total time take
algorithm

    screen_display = pyg.display.set_mode((600, 250)) # Create a screen
    screen_display.blit(screen, (0, 0))
    pyg.display.update()
```

```
    # Find the path form start to goal
    path = []
    if goal_node!=None:
        st_time = time.time()
        print("The final goal node is given by: ",goal_node)
        while goal_node is not None:
            goal_node_parent = closed_list[goal_node][3]
            path.append(closed_list[goal_node][4])
            goal_node=goal_node_parent


        # reverse the path list to get the correct order of nodes
        path.reverse()
        et_time = time.time()
        print("Total time taken for backtracking:",et_time-st_time)
        print("****** The optimum path is ****",path)

    # To draw the graph of exploration nodes on the canvas
    for key in dict_vector.keys():
        for i in range (0,len(dict_vector[key])):
            pyg.draw.aaline(screen_display,(0,0,0),key,dict_vector[key][i])
            pyg.display.update()

    print("Length of closed nodes=",len(closed_list))

    # To draw the path taken by the robot from start node to goal node
    for i in range(0,len(path)):
        if(i+1>len(path)-1):
            break
        pyg.draw.line(screen_display,green,path[i],path[i+1],width=1)
        pyg.display.update()



    # Set the caption of the screen
    pyg.display.set_caption('A* Visualization Map')
    pyg.display.update()
    pyg.time.wait(1)
    running=True
    while running:
        # for loop through the event queue
        for event in pyg.event.get():
            # Check for QUIT event
            if event.type == pyg.QUIT:
                running = False
```