TEL280 Assignment 5

A\* path planer with the Manhattan distance heuristic, moving the robot just up, down, left, right. Green is cells that are explored and blue is the backtracking path:

Chart

Description automatically generated

A\* path planer with the Manhattan distance heuristic, moving the robot in the eight neighboring cell. Green is cells that are explored and blue is the backtracking path:

Chart

Description automatically generated

A\* path planer with the Diagonal distance heuristic, moving the robot in the eight neighboring cell. Green is cells that are explored and blue is the backtracking path:

Chart

Description automatically generated

Code used to calculate A\* with given image.

import skimage  
import matplotlib.pyplot as plt  
import numpy as np  
  
  
class Squared(object):  
 def \_\_init\_\_(self, x, y, x\_start, x\_stop, y\_start, y\_stop):  
 self.x = x  
 self.y = y  
 self.x\_start = x\_start  
 self.x\_stop = x\_stop  
 self.y\_start = y\_start  
 self.y\_stop = y\_stop  
 self.free = True  
 self.goal = False  
 self.start = False  
 self.G = 0  
 self.H = 0  
 self.F = self.G + self.H  
 self.parent = None  
  
 def check\_box(self, image):  
 for k in np.arange(self.x\_start, self.x\_stop, 1):  
 for o in np.arange(self.y\_start, self.y\_stop, 1):  
 red = image[int(o), int(k), 0]  
 green = image[int(o), int(k), 1]  
 blue = image[int(o), int(k), 2]  
 if red > 200 and green < 1 and blue < 1:  
 self.free = False  
 break  
  
 elif green > 200 and red < 200 and blue < 200:  
 self.goal = True  
 break  
  
 elif blue > 200 and red < 200 and green < 200:  
 self.start = True  
 break  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 # %% Initialising  
  
 im = skimage.io.imread('Assignment5\_tel280.png')  
 xstart = x\_\_start = 0  
 xstop = x\_\_stop = 1169  
 ystart = 0  
 ystop = y\_\_stop = 797  
 grid\_sizex = 0  
 stepx = (xstop-xstart)/22  
 grid\_sizey = 0  
 stepy = (ystop-ystart)/15  
  
 list\_of\_nodes = []  
 x\_\_stop = x\_\_start + stepx  
 for i in range(21):  
 x\_\_start += stepx  
 x\_\_stop += stepx  
 y\_\_start = ystart  
 y\_\_stop = y\_\_start + stepy  
 for j in range(14):  
 y\_\_start += stepy  
 y\_\_stop += stepy  
 list\_of\_nodes.append(Squared(i, j, x\_\_start, x\_\_stop, y\_\_start, y\_\_stop))  
 im[int(y\_\_start):int(y\_\_stop), int(x\_\_start)] = 0  
 im[int(y\_\_start):int(y\_\_stop), int(x\_\_stop)] = 0  
 im[int(y\_\_start), int(x\_\_start):int(x\_\_stop)] = 0  
 im[int(y\_\_start), int(x\_\_stop):int(x\_\_stop)] = 0  
  
 for element, node in enumerate(list\_of\_nodes):  
 node.check\_box(im)  
  
 # %% A\* mapping  
 plt.imshow(im)  
 plt.show()  
 im2 = im  
  
 open\_list = []  
 closed\_list = []  
 for i in range(len(list\_of\_nodes)):  
 if list\_of\_nodes[i].start:  
 start\_square = list\_of\_nodes[i]  
 open\_list.append(list\_of\_nodes[i])  
  
 for i in range(len(list\_of\_nodes)):  
 if list\_of\_nodes[i].goal:  
 goal\_square = list\_of\_nodes[i]  
  
 run = True  
 while run:  
 open\_list.sort(key=lambda x: x.F)  
 q = open\_list[0]  
  
 # find neighbour\_cell  
 neighbour\_cell = []  
 for i in range(len(list\_of\_nodes)):  
 if list\_of\_nodes[i].y == q.y - 1 \  
 and list\_of\_nodes[i].x == q.x \  
 and list\_of\_nodes[i].free:  
 upper = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].y == q.y - 1 \  
 and list\_of\_nodes[i].x == q.x + 1 \  
 and list\_of\_nodes[i].free:  
 upper\_right = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].x == q.x + 1 \  
 and list\_of\_nodes[i].y == q.y \  
 and list\_of\_nodes[i].free:  
 right = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].x == q.x + 1 \  
 and list\_of\_nodes[i].y == q.y + 1 \  
 and list\_of\_nodes[i].free:  
 lower\_right = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].y == q.y + 1 \  
 and list\_of\_nodes[i].x == q.x \  
 and list\_of\_nodes[i].free:  
 lower = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].y == q.y + 1 \  
 and list\_of\_nodes[i].x == q.x - 1 \  
 and list\_of\_nodes[i].free:  
 lower\_left = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].x == q.x - 1 \  
 and list\_of\_nodes[i].y == q.y \  
 and list\_of\_nodes[i].free:  
 left = list\_of\_nodes[i]  
  
 elif list\_of\_nodes[i].x == q.x - 1 \  
 and list\_of\_nodes[i].y == q.y - 1 \  
 and list\_of\_nodes[i].free:  
 upper\_left = list\_of\_nodes[i]  
  
 neighbour\_cell = [upper, upper\_left, left, lower\_left,  
 lower, lower\_right, right, upper\_right]  
  
 for i in range(len(neighbour\_cell)):  
  
 dx = abs(goal\_square.x - neighbour\_cell[i].x)  
 dy = abs(goal\_square.y - neighbour\_cell[i].y)  
 neighbour\_cell[i].H = (dx + dy) # + (np.sqrt(2) - 2) \* min(dx, dy) # Only used for Diagonal distance  
  
 dx = abs(start\_square.x - neighbour\_cell[i].x)  
 dy = abs(start\_square.y - neighbour\_cell[i].y)  
 neighbour\_cell[i].G = (dx + dy) # + (np.sqrt(2) - 2) \* min(dx, dy) # Only used for Diagonal distance   
  
 neighbour\_cell[i].F = neighbour\_cell[i].G + neighbour\_cell[i].H  
  
 im[int(neighbour\_cell[i].y\_start):int(neighbour\_cell[i].y\_stop),  
 int(neighbour\_cell[i].x\_start):int(neighbour\_cell[i].x\_stop), 2] = 0  
 im[int(neighbour\_cell[i].y\_start):int(neighbour\_cell[i].y\_stop),  
 int(neighbour\_cell[i].x\_start):int(neighbour\_cell[i].x\_stop), 0] = 0  
 im[int(neighbour\_cell[i].y\_start):int(neighbour\_cell[i].y\_stop),  
 int(neighbour\_cell[i].x\_start):int(neighbour\_cell[i].x\_stop), 1] = 255  
  
 if neighbour\_cell[i].parent is None:  
 neighbour\_cell[i].parent = q  
  
 temp = []  
 not\_add = []  
 for i in range(len(neighbour\_cell)):  
 for j in range(len(open\_list)):  
 """if a node with the same position as successor is in the OPEN list which   
 has a lower f than successor, skip this successor"""  
 if (neighbour\_cell[i].y == open\_list[j].y  
 and neighbour\_cell[i].x == open\_list[j].x  
 and neighbour\_cell[i].F >= open\_list[j].F):  
 not\_add.append(neighbour\_cell[i])  
  
 elif (neighbour\_cell[i].y == open\_list[j].y  
 and neighbour\_cell[i].x == open\_list[j].x  
 and neighbour\_cell[i].F < open\_list[j].F):  
 open\_list.remove(open\_list[j])  
  
 for j in range(len(closed\_list)):  
 """if a node with the same position as successor is in the CLOSED list which  
 has a lower f than successor, skip this successor"""  
 if (neighbour\_cell[i].y == closed\_list[j].y  
 and neighbour\_cell[i].x == closed\_list[j].x  
 and neighbour\_cell[i].F >= closed\_list[j].F):  
 not\_add.append(neighbour\_cell[i])  
  
 elif (neighbour\_cell[i].y == closed\_list[j].y  
 and neighbour\_cell[i].x == closed\_list[j].x  
 and neighbour\_cell[i].F < closed\_list[j].F):  
 open\_list.remove(closed\_list[j])  
  
 temp.append(neighbour\_cell[i])  
  
 for i in range(len(not\_add)):  
 temp.remove(not\_add[i])  
  
 for i in range(len(temp)):  
 open\_list.append(temp[i])  
  
 open\_list.remove(q)  
 closed\_list.append(q)  
  
 for i in range(len(neighbour\_cell)):  
 if neighbour\_cell[i].goal:  
 closed\_list.append(neighbour\_cell[i])  
 run = False  
  
 plt.imshow(im)  
 plt.show()  
  
 def callback(cell, start\_squar, parentlist, im2):  
 if start\_squar.x == cell.x and start\_squar.y == cell.y:  
 return parentlist  
 parent = cell.parent  
 im2[int(cell.y\_start):int(cell.y\_stop), int(cell.x\_start):int(cell.x\_stop), 2] = 255  
 im2[int(cell.y\_start):int(cell.y\_stop), int(cell.x\_start):int(cell.x\_stop), 0] = 0  
 im2[int(cell.y\_start):int(cell.y\_stop), int(cell.x\_start):int(cell.x\_stop), 1] = 0  
 parentlist.append(parent)  
 callback(parent, start\_squar, parentlist, im2)  
  
  
 closed\_list.reverse()  
 parentliste = []  
 ans = callback(goal\_square, start\_square, parentliste, im2)  
 plt.imshow(im2)  
 plt.show()