成 绩 _____

北京航空航天大學

BEIHANG UNIVERSITY

使用 A*算法进行图搜索解决机器人路径规划问题

院（系）名称	高等理工学院
专 业 名 称	自动化
学 生 学 号	18376251
学 生 姓 名	乐祥立
指 导 教 师	王岩

2021 年 6 月

使用 A*算法进行图搜索解决机器人路径规划问题

一、实验目的

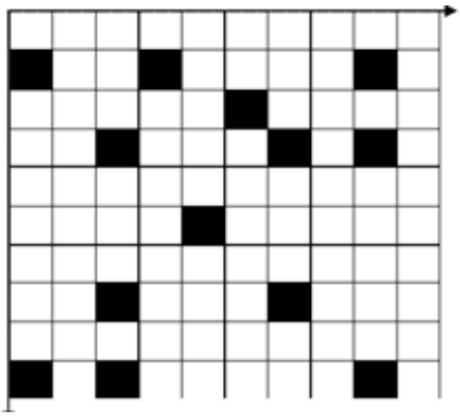
1. 使学生加深对图搜索技术的理解
2. 掌握图搜索基本编程方法
3. 运用图搜索技术解决一些应用问题

二、实验要求

1. 用启发式搜索算法实现路径规划问题。
2. 有明确的状态空间表达，规则集以及估计函数。
3. 程序运行时，应能清晰直观演示搜索过程。

三、实验内容

机器人路径规划问题：左上角为坐标原点，水平向右为 x 轴方向，竖直向下为 y 轴方向。白色为自由栅格，黑色为障碍栅格，机器人只能在自由栅格中运动，并躲避障碍。每个栅格由唯一的坐标（x, y）表示。机器人一般有八个可移动方向。给出由初始位置（3，3）到目标位置（9，9）的最佳路线。



四、实验步骤

1、设计问题的状态表示方法

状态就是问题在任一确定时刻的状况，表征了问题的特征和结构。状态一般用一组

数据表示，本实验中状态使用机器人在地图中的坐标（x, y）表示。初始状态为（3，3），也可以任意进行设定。一个问题的状态图是一个三元组(S, F, G)，S 是初始状态的集合，F 是问题的状态转换规则集合，G 是问题的目标状态集合。

对于转换规则 F，由于机器人可以沿着八个方向进行移动，因此可以制定 8 条状态转换规则：

（1）如果左边没有墙壁或障碍，机器人可以向左走一格$(x, y) \rightarrow (x-1, y)$

（2）如果右边没有墙壁或障碍，机器人可以向右走一格$(x, y) \rightarrow (x+1, y)$

（3）如果上边没有墙壁或障碍，机器人可以向上走一格$(x, y) \rightarrow (x, y-1)$

（4）如果下边没有墙壁或障碍，机器人可以向下走一格$(x, y) \rightarrow (x, y+1)$

（5）如果左上边没有墙壁或障碍，机器人可以向左上走一格$(x, y) \rightarrow (x-1, y-1)$

（6）如果左下边没有墙壁或障碍，机器人可以向左下走一格$(x, y) \rightarrow (x-1, y+1)$

（7）如果右上边没有墙壁或障碍，机器人可以向右上走一格$(x, y) \rightarrow (x+1, y-1)$

（8）如果右下边没有墙壁或障碍，机器人可以向右下走一格$(x, y) \rightarrow (x+1, y+1)$

2、启发式函数

启发式函数一般是一个估价函数 f，对当前的搜索状态进行评估，找出一个最有希望的节点进行扩展。一般定义为：

$$f(n) = g(n) + h(n)$$

$g^*(n)$是 s 到 n 的最优路径的实际代价；$h^*(n)$是 n 到 g 的最优路径的实际代价；$f^*(n) = g^*(n) + h^*(n)$是从 s 经过 n 到 g 的最优路径的实际代价。而$g(n)$、$h(n)$和$f(n)$分别是$g^*(n)$、$h^*(n)$和$f^*(n)$的估计值。

若要满足 A*算法，需要使得$h(n) \leq h^*(n)$，即对于集合$h^*(n)$，$h(n)$是$h^*(n)$的下界。根据两点之间直线最短，由于地图中存在着障碍，因此使用欧式距离则一定可以保证上述条件。

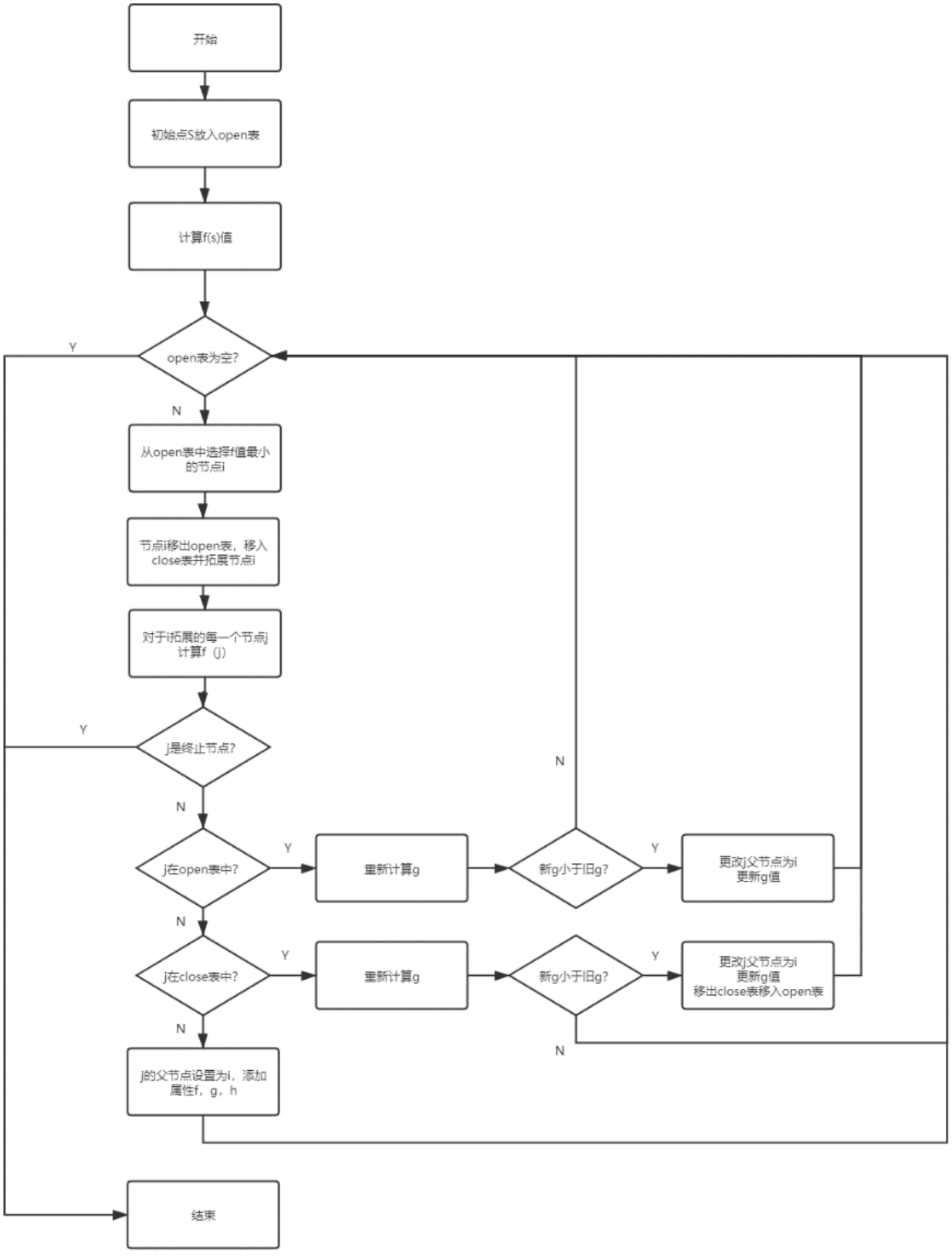
$$h(n) = \sqrt{(x_1 - x_g)^2 + (y_1 - y_g)^2}$$

在该定义下，满足 A*算法。

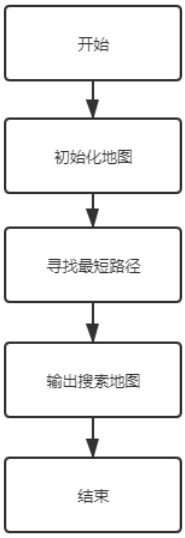
$g(n)$则定义为从起点运动到当前位置的实际代价。

3、过程分析

使用 A*算法进行搜索的流程图：

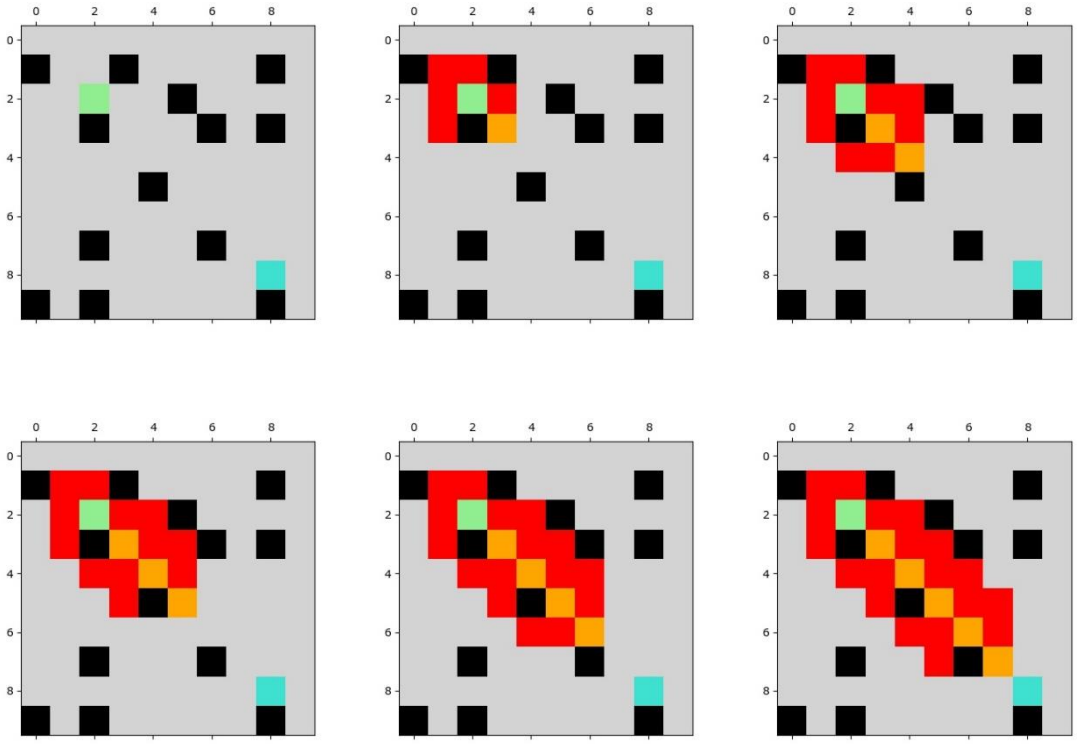

开始
初始点S放入open表
计算f(s)值
open表为空?
Y
N
从open表中选择f值最小的节点i
节点i移出open表，移入close表并拓展节点i
对于i拓展的每一个节点j计算f（j）
j是终止节点?
Y
N
j在open表中?
Y
重新计算g
新g小于旧g?
Y
更改j父节点为i
更新g值
N
j在close表中?
Y
重新计算g
新g小于旧g?
Y
更改j父节点为i
更新g值
移出close表移入open表
N
N
j的父节点设置为i，添加属性f，g，h
结束

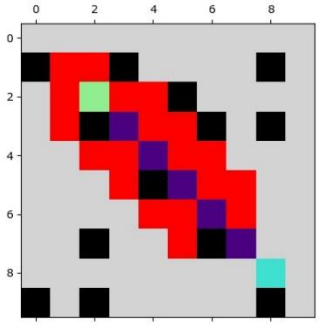
代码的总流程图为：



五、实验结果

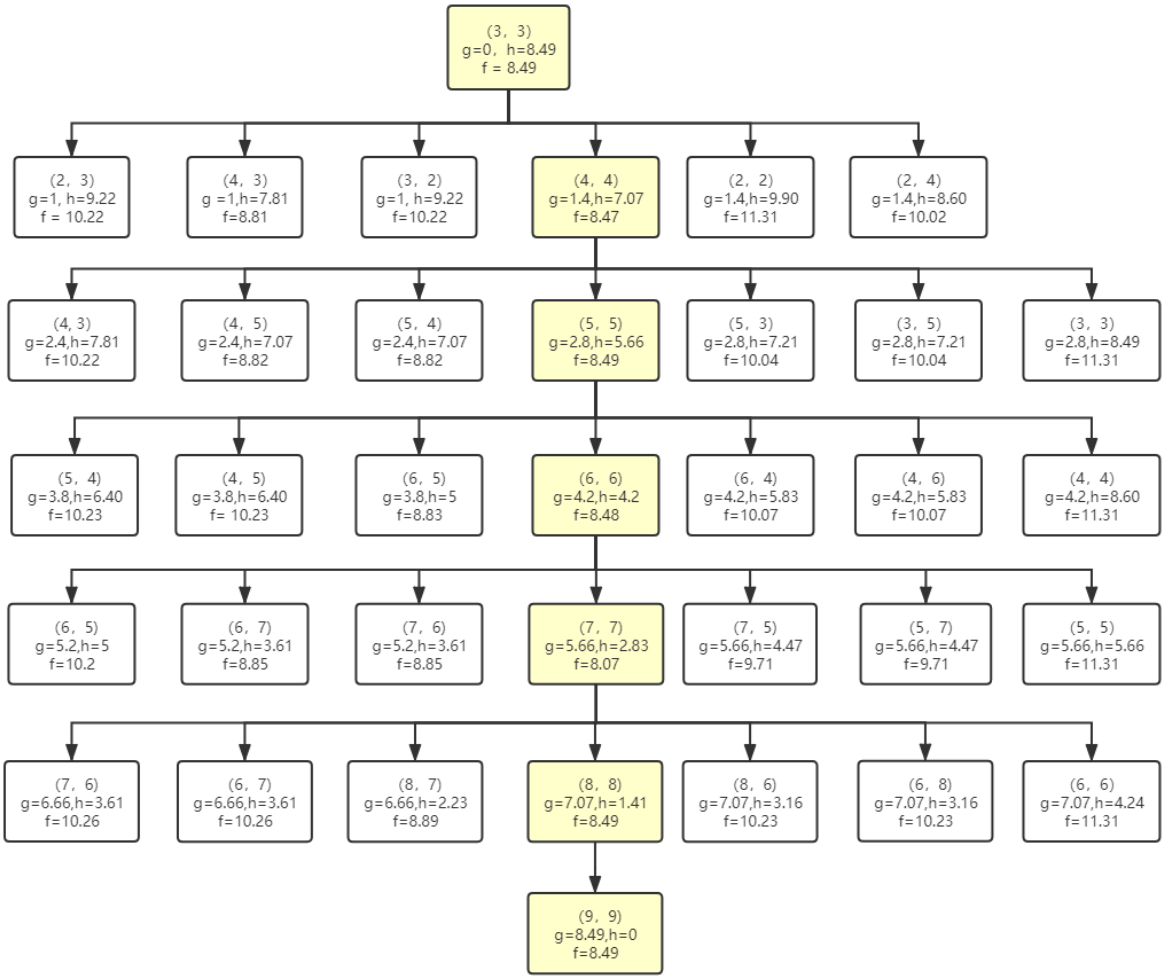
根据上述代码流程图可以知道，算法会根据当前的状态选择最优的节点进行拓展。根据启发函数的定义，拓展的方向总是一开始沿着从起始节点指向终止节点的。首先给出搜过过程的动态变化图。





图中，灰色的区域为机器人可以移动的范围；黑色的区域为地图上的障碍；浅绿色的点为机器人出发的位置；蓝色的点为目标位置；红色的区域为 open 表涵盖的区域；橙色的区域为 close 表涵盖的区域；最后出现的紫色区域是机器人的最优路径。

可以给出搜索图如下：



程序每次从当前所在的位置出发，寻找周围可以移动的区域，加入拓展表中，若新

的位置已经存在于 open 表或者 close 表中，则重新计算代价进行评估；若不在，则直接加入 open 表。从 open 表中寻找到下一个最优的节点，并将其移入 close 表开始拓展。动态过程图展示的情况和算法一致。

从搜索图中可以看出，一共进行了 6 次拓展，每次按照最优的启发函数寻找节点进行拓展，过程为$(3,3) \rightarrow (4,4) \rightarrow (5,5) \rightarrow (6,6) \rightarrow (7,7) \rightarrow (8,8) \rightarrow (9,9)$。从搜索过程的动态变化图和搜索树中可以发现，程序找到的确实是最优路径，每次的启发函数估计值最小。由于地图中有障碍的存在，因此始终满足$h(\mathrm{n}) \leq h^*(n)$。

六、总结分析

通过实验可以发现，启发式搜索具有较高的搜索效率，但是其搜索的效率和所定义的估价函数有关，特别是h(n)的定义。当使用欧式距离时，从搜索图中可以看出，每个状态之间的估价函数差距不大，当搜索的地图更加复杂时，虽然可以找到最优的路径，但是会耗费较长的时间。如果使用曼哈顿距离，可以让搜索变得更加高效，但是有可能找到的并不是最优解。也就是说，

$h(\mathrm{n}) < h^*(n)$搜索的效率较低，但是能找到最优解

$h(\mathrm{n}) = h^*(n)$找到最优解，兼具效率

$h(\mathrm{n}) \geq h^*(n)$搜索的效率较高，可有能找到的不是最优解

此外，在编写代码的运行的过程中，还出现了找到目标位置却继续进行搜索和没找到目标，但最后显示了路径的错误。前一个问题是在 open 表中选择 f 值最小的节点之后没有判断是否包含目标节点导致的持续进行搜索，最坏的情况是将整个图进行了遍历，在增添了判断之后解决了这个问题。后一个问题是最后没有找到路径时（原因是障碍过多），绘图的函数判断出错，导致的颜色错误，解决的办法是在绘图的函数中增添判断是否没有找到路径的情况。

附件：

实验代码以及数据链接：

https://github.com/lerlis/robot_path_planning_by_Astar

实验代码：

main.py

```
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

robot_map = []


class Block:
    def __init__(self, x, y, status):
        self.x = x
        self.y = y
        self.status = status
        self.bl_g = -1
        self.bl_h = -1
        self.bl_f = -1
        self.is_path = 0
        self.parent_block = None


BLOCK_STATE_NORMAL = 0
BLOCK_STATE_OBSTACLE = 1
BLOCK_STATE_START = 2
BLOCK_STATE_END = 3


def init_map(length, width, opoint):
    global map_length, map_width, start_point, end_point
    if (width == -1) and (length == -1):
        map_length = 10
        map_width = 10
        for i in range(map_width):
            for j in range(map_length):
                robot_map.append(Block(j, i, BLOCK_STATE_NORMAL))
        obstacle = [10, 13, 18, 25, 32, 36, 38, 54, 72, 76, 90, 92, 98]
        for i in range(map_width * map_length):
            if i in obstacle:
                robot_map[i].status = BLOCK_STATE_OBSTACLE
            elif robot_map[i].x == 2 and robot_map[i].y == 2:
                robot_map[i].status = BLOCK_STATE_START
```

```
41.                 start_point = i
42.             elif robot_map[i].x == 8 and robot_map[i].y == 8:
43.                 robot_map[i].status = BLOCK_STATE_END
44.                 end_point = i
45.     else:
46.         for i in range(map_width):
47.             for j in range(map_length):
48.                 robot_map.append(Block(j, i, BLOCK_STATE_NORMAL))
49.         count = 0
50.         while True:
51.             ob = random.randint(0, map_width * map_length - 1)
52.             if robot_map[ob].status == BLOCK_STATE_NORMAL:
53.                 count = count + 1
54.                 robot_map[ob].status = BLOCK_STATE_OBSTACLE
55.             if count == opoint:
56.                 break
57.         while True:
58.             spoint = random.randint(0, map_width * map_length - 1)
59.             if robot_map[spoint].status == BLOCK_STATE_NORMAL:
60.                 robot_map[spoint].status = BLOCK_STATE_START
61.                 start_point = spoint
62.                 break
63.         while True:
64.             epoint = random.randint(0, map_width * map_length - 1)
65.             if robot_map[epoint].status == BLOCK_STATE_NORMAL:
66.                 robot_map[epoint].status = BLOCK_STATE_END
67.                 end_point = epoint
68.                 break
69.
70.
71. def print_map():
72.     len_map = len(robot_map)
73.     map_array = []
74.     map_line = []
75.     for i in range(len_map):
76.         if robot_map[i].status == BLOCK_STATE_NORMAL:
77.             if robot_map[i].is_path == 1:
78.                 map_line.append('P')
79.             else:
80.                 map_line.append(0)
81.         elif robot_map[i].status == BLOCK_STATE_OBSTACLE:
82.             map_line.append(1)
83.         elif robot_map[i].status == BLOCK_STATE_START:
```

```
            map_line.append('s')
        else:
            map_line.append('e')
        if (i + 1) % map_length == 0:
            map_array.append(map_line)
            map_line = []
    np_map = np.array(map_array)
    print(np_map)


def find_path():
    global map_length, map_width, start_point, end_point
    count = 0
    # Open and Close 表
    open_list = []
    close_list = []

    open_list.append(robot_map[start_point])
    cal_function_F(robot_map[start_point], robot_map[end_point])

    while len(open_list) > 0:
        minFpoint = find_minF_point(open_list)
        open_list.remove(minFpoint)
        close_list.append(minFpoint)

        display(open_list, close_list, count)
        count = count + 1

        expand_list = find_surround_way(minFpoint, close_list)
        if robot_map[end_point] in expand_list:
            robot_map[end_point].parent_block = minFpoint
            print('yes!')
            break
        for expoint in expand_list:

            if expoint in open_list:
                new_gain = cal_alter_g(expoint, minFpoint)
                if new_gain < expoint.bl_g:
                    expoint.parent_block = minFpoint
                    expoint.bl_g = new_gain
            elif expoint in close_list:
                new_gain = cal_alter_g(expoint, minFpoint)
                if new_gain < expoint.bl_g:
```

```
                    expoint.parent_block = minFpoint
                    expoint.bl_g = new_gain
                    close_list.remove(expoint)
                    open_list.append(expoint)
            else:
                expoint.parent_block = minFpoint
                cal_function_F(expoint, robot_map[end_point])
                open_list.append(expoint)
    End_game = robot_map[end_point]
    while True:
        End_game.is_path = 1
        End_game = End_game.parent_block
        if End_game is None:
            break
    print('结果: ')
    print_map()
    display(open_list, close_list, count)


def cal_alter_g(point, minpoint):
    new_g = np.sqrt((point.x - minpoint.x) ** 2 + (
            point.y - minpoint.y) ** 2) + minpoint.bl_g
    return new_g


def cal_function_F(point, epoint):
    if point.parent_block is None:
        g = 0
    else:
        g = np.sqrt((point.x - point.parent_block.x) ** 2 + (
            point.y - point.parent_block.y) ** 2) + point.parent_block.bl_g
    h = np.sqrt((point.x - epoint.x) ** 2 + (point.y - epoint.y) ** 2)
    point.bl_g = g
    point.bl_h = h
    point.bl_f = g + h


def find_minF_point(op_list):
    min = np.inf
    temp = 0
    for point in op_list:
        if point.bl_f < min:
            min = point.bl_f
```

```
            temp = point
    return temp


def find_surround_way(mpoint, exist_list):
    global map_length, map_width, start_point, end_point
    expand_list = []
    # 左边的点
    if mpoint.x > 0:
        left = robot_map[mpoint.y * map_length + mpoint.x - 1]
        if check(left, exist_list):
            expand_list.append(left)
    # 右边的点
    if mpoint.x < map_length - 1:
        right = robot_map[mpoint.y * map_length + mpoint.x + 1]
        if check(right, exist_list):
            expand_list.append(right)
    # 上边的点
    if mpoint.y > 0:
        up = robot_map[(mpoint.y - 1) * map_length + mpoint.x]
        if check(up, exist_list):
            expand_list.append(up)
    # 下边的点
    if mpoint.y < map_width - 1:
        down = robot_map[(mpoint.y + 1) * map_length + mpoint.x]
        if check(down, exist_list):
            expand_list.append(down)
    # 左上方的点
    if mpoint.x > 0 and mpoint.y > 0:
        left_up = robot_map[(mpoint.y - 1) * map_length + mpoint.x - 1]
        if check(left_up, exist_list):
            expand_list.append(left_up)
    # 左下方的点
    if mpoint.x > 0 and mpoint.y < map_width - 1:
        left_down = robot_map[(mpoint.y + 1) * map_length + mpoint.x - 1]
        if check(left_down, exist_list):
            expand_list.append(left_down)
    # 右上方的点
    if mpoint.x < map_length - 1 and mpoint.y > 0:
        right_up = robot_map[(mpoint.y - 1) * map_length + mpoint.x + 1]
        if check(right_up, exist_list):
            expand_list.append(right_up)
    # 右下方的点
```

```
213.     if mpoint.x < map_length - 1 and mpoint.y < map_width - 1:
214.         right_down = robot_map[(mpoint.y + 1) * map_length + mpoint.x + 1]
215.         if check(right_down, exist_list):
216.             expand_list.append(right_down)
217.     return expand_list
218.
219.
220. def check(point, exist):
221.     if point.status == BLOCK_STATE_OBSTACLE:
222.         return 0
223.     else:
224.         return 1
225.
226.
227. def display(op, cl, count):
228.     mat = np.zeros([map_length, map_width])
229.     chang = len(mat)
230.     kuan = len(mat[1])
231.     t = 0
232.     judge = [0, 0, 0]
233.     c = 0
234.     for i in range(kuan):
235.         for j in range(chang):
236.             if robot_map[t] in op:
237.                 judge[0] = 1
238.                 mat[i][j] = 4
239.             if robot_map[t] in cl:
240.                 if len(cl) == 1:
241.                     judge[1] = 0
242.                 else:
243.                     judge[1] = 1
244.                 mat[i][j] = 5
245.             if robot_map[t].is_path == 1:
246.                 c = c + 1
247.                 if c == 1:
248.                     judge[2] = 0
249.                 else:
250.                     judge[2] = 1
251.                 mat[i][j] = 6
252.             if robot_map[t].status == BLOCK_STATE_OBSTACLE:
253.                 mat[i][j] = 1
254.             elif robot_map[t].status == BLOCK_STATE_START:
255.                 mat[i][j] = 2
```

```
256.             elif robot_map[t].status == BLOCK_STATE_END:
257.                 mat[i][j] = 3
258.             t = t + 1
259.     cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise', '
    Red', 'Orange', 'Indigo'])
260.     if judge == [0, 0, 0]:
261.         cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise
    '])
262.     elif judge == [1, 0, 0]:
263.         cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise
    ', 'Red'])
264.     elif judge == [0, 1, 0]:
265.         cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise
    ', 'Orange'])
266.     elif judge == [1, 1, 1]:
267.         cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise
    ', 'Red', 'Orange', 'Indigo'])
268.     elif judge == [1, 1, 0]:
269.         cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise
    ', 'Red', 'Orange'])
270.     elif judge == [0, 1, 1]:
271.         cmap = ListedColormap(['LightGray', 'Black', 'Lightgreen', 'Turquoise
    ', 'Orange', 'Indigo'])
272.     plt.figure(count)
273.     plt.matshow(mat, cmap=cmap)
274.     plt.savefig('./random/%s.jpg' % count)
275.     plt.show()
276.
277.
278. if __name__ == '__main__':
279.     start_point = -1
280.     end_point = -1
281.     map_length = -1
282.     map_width = -1
283.     obstacle_point = 2000
284.     init_map(map_length, map_width, obstacle_point)
285.     print_map()
286.     find_path()
```