

成 绩

****

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

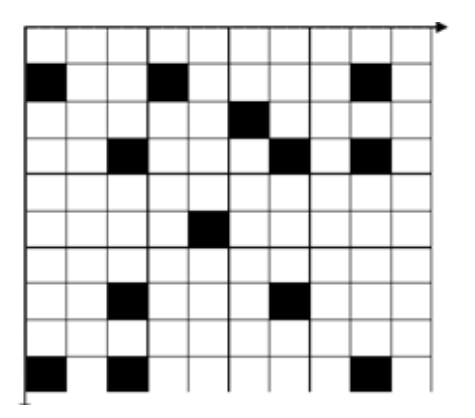
1. 用启发式搜索算法实现路径规划问题。

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

（2）如果右边没有墙壁或障碍，机器人可以向右走一格

（3）如果上边没有墙壁或障碍，机器人可以向上走一格

（4）如果下边没有墙壁或障碍，机器人可以向下走一格

（5）如果左上边没有墙壁或障碍，机器人可以向左上走一格

（6）如果左下边没有墙壁或障碍，机器人可以向左下走一格

（7）如果右上边没有墙壁或障碍，机器人可以向右上走一格

（8）如果右下边没有墙壁或障碍，机器人可以向右下走一格

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：

是s到n的最优路径的实际代价；是n到g的最优路径的实际代价；是从s经过n到g的最优路径的实际代价。而、和分别是、和的估计值。

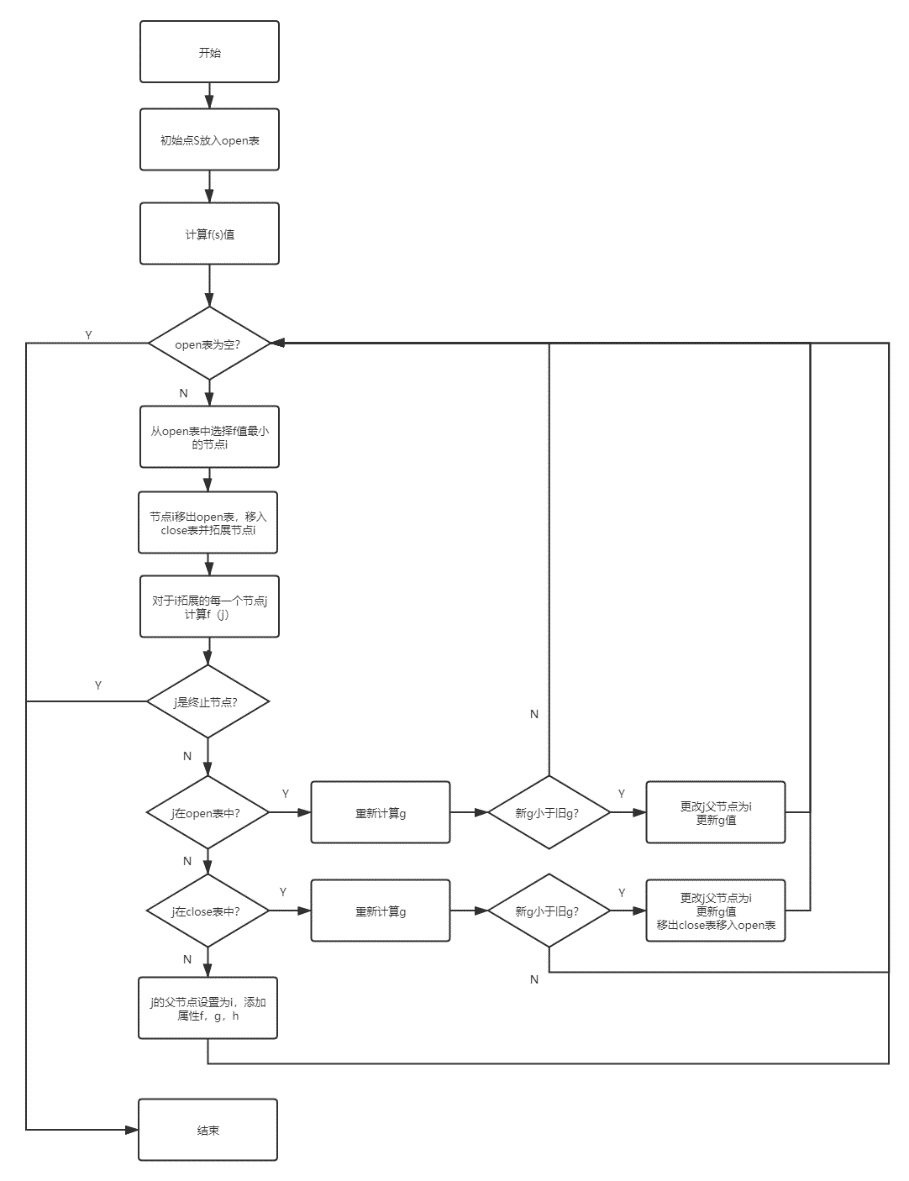
若要满足A\*算法，需要使得，即对于集合，是的下界。根据两点之间直线最短，由于地图中存在着障碍，因此使用欧式距离则一定可以保证上述条件。

在该定义下，满足A\*算法。

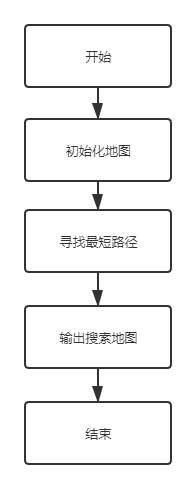
则定义为从起点运动到当前位置的实际代价。

3、过程分析

使用A\*算法进行搜索的流程图：

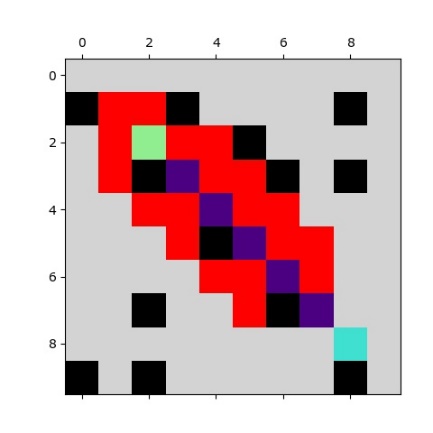
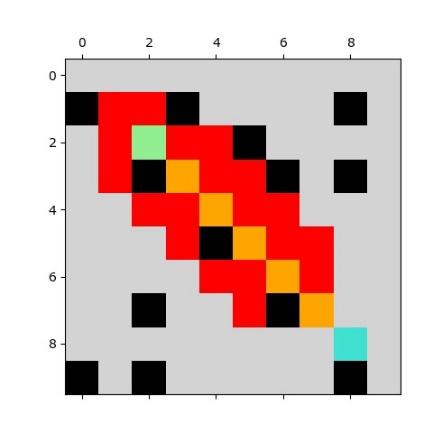
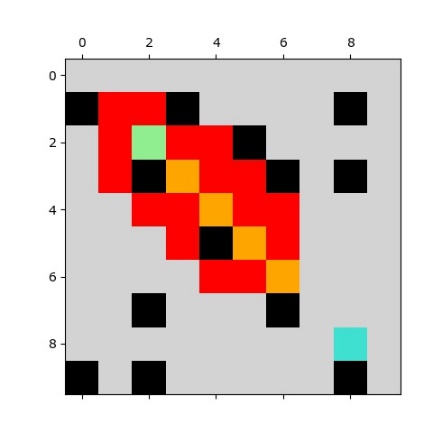
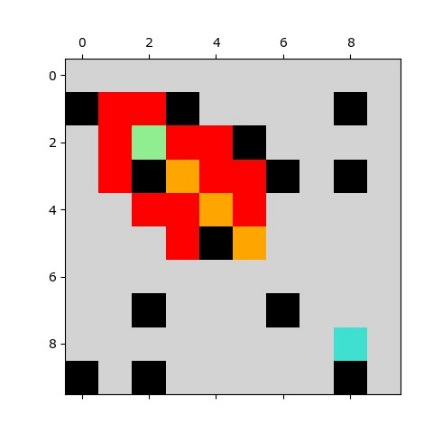
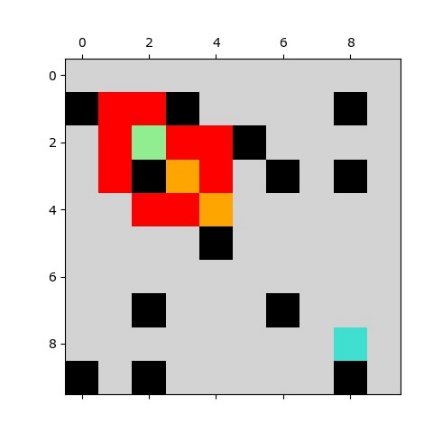
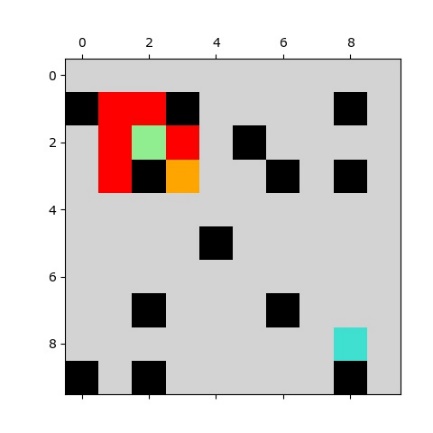
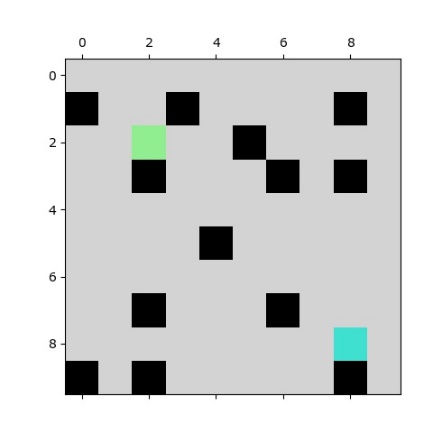


代码的总流程图为：



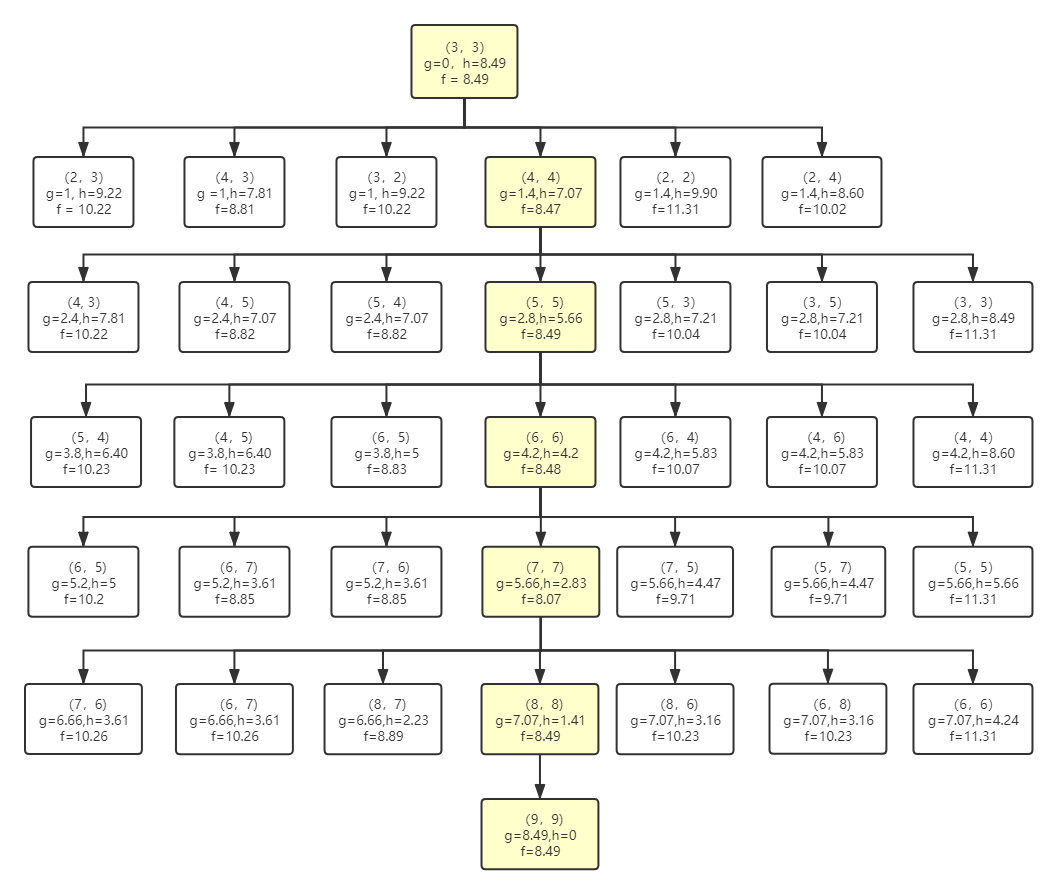
五、实验结果

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。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。

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，

搜索的效率较低，但是能找到最优解

找到最优解，兼具效率

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

1. **import** random
2. **import** numpy **as** np
3. **import** matplotlib.pyplot **as** plt
4. **from** matplotlib.colors **import** ListedColormap
6. robot\_map = []

9. **class** Block:
10. **def** \_\_init\_\_(self, x, y, status):
11. self.x = x
12. self.y = y
13. self.status = status
14. self.bl\_g = -1
15. self.bl\_h = -1
16. self.bl\_f = -1
17. self.is\_path = 0
18. self.parent\_block = None

21. BLOCK\_STATE\_NORMAL = 0
22. BLOCK\_STATE\_OBSTACLE = 1
23. BLOCK\_STATE\_START = 2
24. BLOCK\_STATE\_END = 3

27. **def** init\_map(length, width, opoint):
28. **global** map\_length, map\_width, start\_point, end\_point
29. **if** (width == -1) **and** (length == -1):
30. map\_length = 10
31. map\_width = 10
32. **for** i **in** range(map\_width):
33. **for** j **in** range(map\_length):
34. robot\_map.append(Block(j, i, BLOCK\_STATE\_NORMAL))
35. obstacle = [10, 13, 18, 25, 32, 36, 38, 54, 72, 76, 90, 92, 98]
36. **for** i **in** range(map\_width \* map\_length):
37. **if** i **in** obstacle:
38. robot\_map[i].status = BLOCK\_STATE\_OBSTACLE
39. **elif** robot\_map[i].x == 2 **and** robot\_map[i].y == 2:
40. 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**

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:
84. map\_line.append('s')
85. **else**:
86. map\_line.append('e')
87. **if** (i + 1) % map\_length == 0:
88. map\_array.append(map\_line)
89. map\_line = []
90. np\_map = np.array(map\_array)
91. **print**(np\_map)

94. **def** find\_path():
95. **global** map\_length, map\_width, start\_point, end\_point
96. count = 0
97. *# Open and Close 表*
98. open\_list = []
99. close\_list = []
101. open\_list.append(robot\_map[start\_point])
102. cal\_function\_F(robot\_map[start\_point], robot\_map[end\_point])
104. **while** len(open\_list) > 0:
105. minFpoint = find\_minF\_point(open\_list)
106. open\_list.remove(minFpoint)
107. close\_list.append(minFpoint)
109. display(open\_list, close\_list, count)
110. count = count + 1
112. expand\_list = find\_surround\_way(minFpoint, close\_list)
113. **if** robot\_map[end\_point] **in** expand\_list:
114. robot\_map[end\_point].parent\_block = minFpoint
115. **print**('yes!')
116. **break**
117. **for** expoint **in** expand\_list:
119. **if** expoint **in** open\_list:
120. new\_gain = cal\_alter\_g(expoint, minFpoint)
121. **if** new\_gain < expoint.bl\_g:
122. expoint.parent\_block = minFpoint
123. expoint.bl\_g = new\_gain
124. **elif** expoint **in** close\_list:
125. new\_gain = cal\_alter\_g(expoint, minFpoint)
126. **if** new\_gain < expoint.bl\_g:
127. expoint.parent\_block = minFpoint
128. expoint.bl\_g = new\_gain
129. close\_list.remove(expoint)
130. open\_list.append(expoint)
131. **else**:
132. expoint.parent\_block = minFpoint
133. cal\_function\_F(expoint, robot\_map[end\_point])
134. open\_list.append(expoint)
135. End\_game = robot\_map[end\_point]
136. **while** True:
137. End\_game.is\_path = 1
138. End\_game = End\_game.parent\_block
139. **if** End\_game **is** None:
140. **break**
141. **print**('结果：')
142. print\_map()
143. display(open\_list, close\_list, count)

146. **def** cal\_alter\_g(point, minpoint):
147. new\_g = np.sqrt((point.x - minpoint.x) \*\* 2 + (
148. point.y - minpoint.y) \*\* 2) + minpoint.bl\_g
149. **return** new\_g

152. **def** cal\_function\_F(point, epoint):
153. **if** point.parent\_block **is** None:
154. g = 0
155. **else**:
156. g = np.sqrt((point.x - point.parent\_block.x) \*\* 2 + (
157. point.y - point.parent\_block.y) \*\* 2) + point.parent\_block.bl\_g
158. h = np.sqrt((point.x - epoint.x) \*\* 2 + (point.y - epoint.y) \*\* 2)
159. point.bl\_g = g
160. point.bl\_h = h
161. point.bl\_f = g + h

164. **def** find\_minF\_point(op\_list):
165. min = np.inf
166. temp = 0
167. **for** point **in** op\_list:
168. **if** point.bl\_f < min:
169. min = point.bl\_f
170. temp = point
171. **return** temp

174. **def** find\_surround\_way(mpoint, exist\_list):
175. **global** map\_length, map\_width, start\_point, end\_point
176. expand\_list = []
177. *# 左边的点*
178. **if** mpoint.x > 0:
179. left = robot\_map[mpoint.y \* map\_length + mpoint.x - 1]
180. **if** check(left, exist\_list):
181. expand\_list.append(left)
182. *# 右边的点*
183. **if** mpoint.x < map\_length - 1:
184. right = robot\_map[mpoint.y \* map\_length + mpoint.x + 1]
185. **if** check(right, exist\_list):
186. expand\_list.append(right)
187. *# 上边的点*
188. **if** mpoint.y > 0:
189. up = robot\_map[(mpoint.y - 1) \* map\_length + mpoint.x]
190. **if** check(up, exist\_list):
191. expand\_list.append(up)
192. *# 下边的点*
193. **if** mpoint.y < map\_width - 1:
194. down = robot\_map[(mpoint.y + 1) \* map\_length + mpoint.x]
195. **if** check(down, exist\_list):
196. expand\_list.append(down)
197. *# 左上方的点*
198. **if** mpoint.x > 0 **and** mpoint.y > 0:
199. left\_up = robot\_map[(mpoint.y - 1) \* map\_length + mpoint.x - 1]
200. **if** check(left\_up, exist\_list):
201. expand\_list.append(left\_up)
202. *# 左下方的点*
203. **if** mpoint.x > 0 **and** mpoint.y < map\_width - 1:
204. left\_down = robot\_map[(mpoint.y + 1) \* map\_length + mpoint.x - 1]
205. **if** check(left\_down, exist\_list):
206. expand\_list.append(left\_down)
207. *# 右上方的点*
208. **if** mpoint.x < map\_length - 1 **and** mpoint.y > 0:
209. right\_up = robot\_map[(mpoint.y - 1) \* map\_length + mpoint.x + 1]
210. **if** check(right\_up, exist\_list):
211. expand\_list.append(right\_up)
212. *# 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

220. **def** check(point, exist):
221. **if** point.status == BLOCK\_STATE\_OBSTACLE:
222. **return** 0
223. **else**:
224. **return** 1

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()

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()