问题一

总体思路:

第一步 机器人猜想鬼所有可能的位置作为知识库

第二步 输入机器人的位置和寒意值，不断更新知识库

第三步 通过知识库来做决策，前进 or 停止

具体操作步骤:

```
if __name__=='__main__':
    #初始化
    imagin_ghoust = []
    #遍历所有幽灵的可能性
    for i in range(1,7):
        for j in range(1,7):
            for v1 in [1,-1]:
                for v2 in [1,-1]:
                    imagin_ghoust.extend([[[i,4,4,j],v1,v2]])
```

1. 通过循环，遍历鬼的可能性，构建知识库

2. 初始化幽灵，机器人。

   这里的幽灵和机器人是通过面向对象的编程来实现的，因为幽灵和机器人的特征明显，而且数据和函数封闭起来调用，会十分方便。具体的 Ghoust,,Robot 函数在文件 ghoust_robot.py 中

```
#xy为幽灵坐标
#随机初始真幽灵位置x1,y1,x2,y2, v1,v2
xy = [random.randint(1, 6),4,4,random.randint(1, 6)]
v1 = random.choice([1,-1])
v2 = random.choice([1,-1])

#初始化幽灵，机器人,ab为机器人的坐标
g = Ghoust(xy,v1,v2)
r = Robot()
ab = [1,1]

#机器人的移动向量
vx = [1, 0]
vy = [0, 1]
stop = [0,0]
```

3. 只要搜集的信息越多，知识库就越接近真实的鬼的位置。

所以我让鬼走右、上、右、上这四步来到（3，3）,这样能使鬼探测的区域最大。

为了方便写，直接将这四步路放进 way 的数组中

Select_ghoust 这个函数是用来更新知识库的，具体在文件

```
    #机器人走路，右上右上
    #让机器人来到（3,3）
    way = [[1,0],[0,1],[1,0],[0,1]]
    for i in way:
        xy = g.move(xy)
        imagin_ghoust = imagine_walk(imagin_ghoust)
        ab = r.move(i)
        imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)
```

coolanddead_imagineghoust.py 中

```
def imagine_walk(imagine_ghoust):
    for i in range(len(imagine_ghoust)):
        #将前面的代码抄过来，改变变量
        if imagine_ghoust[i][0][0] + imagine_ghoust[i][1] > 6 or imagine_ghoust[i][0][0] + imagine_ghoust[i][1
            #撞墙反向
            imagine_ghoust[i][1] = -imagine_ghoust[i][1]
            imagine_ghoust[i][0][0] = imagine_ghoust[i][0][0] + imagine_ghoust[i][1]
        else :
            imagine_ghoust[i][0][0] = imagine_ghoust[i][0][0] + imagine_ghoust[i][1]

        if imagine_ghoust[i][0][3] + imagine_ghoust[i][2] > 6 or imagine_ghoust[i][0][3] + imagine_ghoust[i][2
            imagine_ghoust[i][2] = -imagine_ghoust[i][2]
            imagine_ghoust[i][0][3] = imagine_ghoust[i][0][3] + imagine_ghoust[i][2]
        else:
            imagine_ghoust[i][0][3] = imagine_ghoust[i][0][3] + imagine_ghoust[i][2]
    return imagine_ghoust

#通过计算遍历的幽灵与机器人的cool,剔除掉不符合条件的幽灵
def select_ghoust(ab, cool1,imagine_ghoust):
    a = len(imagine_ghoust)
    i = 0
    while i <= a-1 :
        if cool(imagine_ghoust[i][0], ab) != cool1:
            imagine_ghoust.pop(i)
            a = len(imagine_ghoust)
            i = i
        else:
            i = i+1

    return imagine_ghoust
```

4. 之后就可以自动走了，判断下一步是否会撞鬼或者越界，如果不撞的话，就可以走

```
68    #自动走
69    while ab != [6,6]:
70        if (dead(vx,imagin_ghoust,ab) == 0) and (ab[0]<6):
71            #右
72            xy = g.move(xy)
73            imagin_ghoust = imagine_walk(imagin_ghoust)
74            ab = r.move(vx)
75            imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)
76            if (xy[0] == ab[0] and xy[1] == ab[1]) or  (xy[2] == ab[0] and xy[3] == ab[1]):
77                print("***************fail***************")
78        elif dead(vy,imagin_ghoust,ab) == 0 and ab[1]<6:
79            #上
80            xy = g.move(xy)
81            imagin_ghoust = imagine_walk(imagin_ghoust)
82            ab = r.move(vy)
83            imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)
84            if (xy[0] == ab[0] and xy[1] == ab[1]) or  (xy[2] == ab[0] and xy[3] == ab[1]):
85                print("***************fail***************")
86        else :
87            #等
88            xy = g.move(xy)
89            imagin_ghoust = imagine_walk(imagin_ghoust)
90            ab = r.move(stop)
91            imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)
92            if (xy[0] == ab[0] and xy[1] == ab[1]) or  (xy[2] == ab[0] and xy[3] == ab[1]):
93                print("***************fail***************")
```

这一步,直到机器人走到[6,6];

5. 让机器人只能向右或者向上走是符合 A 星算法最优的,A 星算法实现起来不太容易，

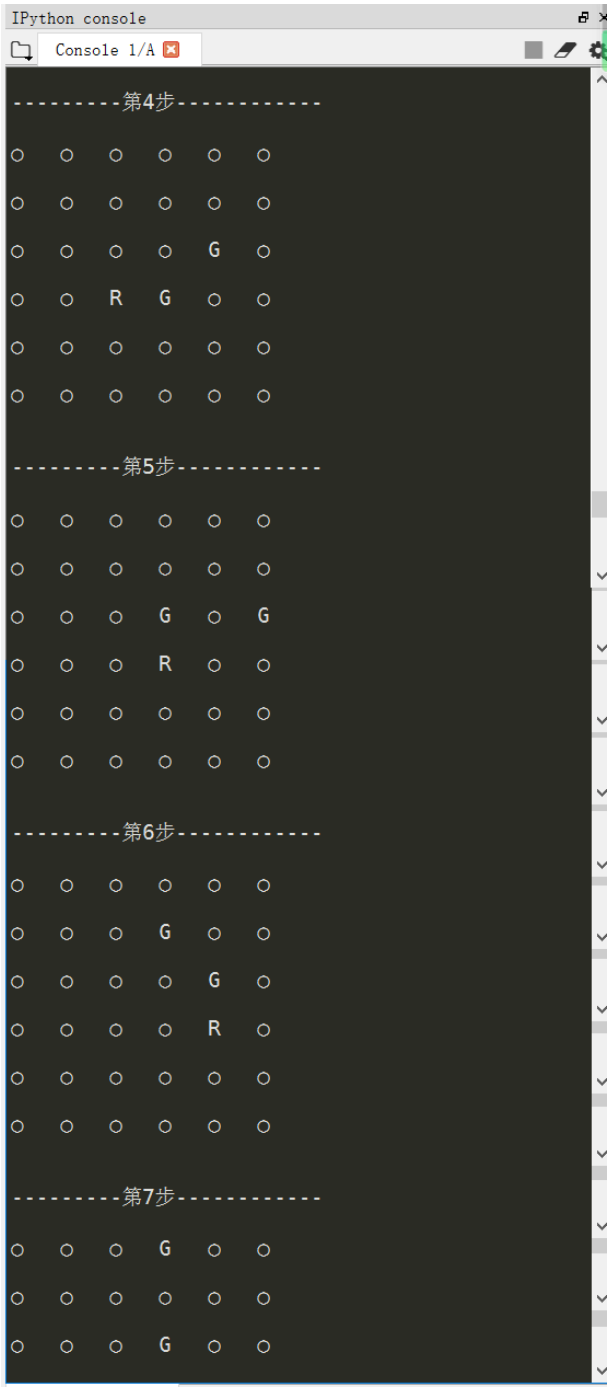
所以就直接让机器人只能向上或者向右走

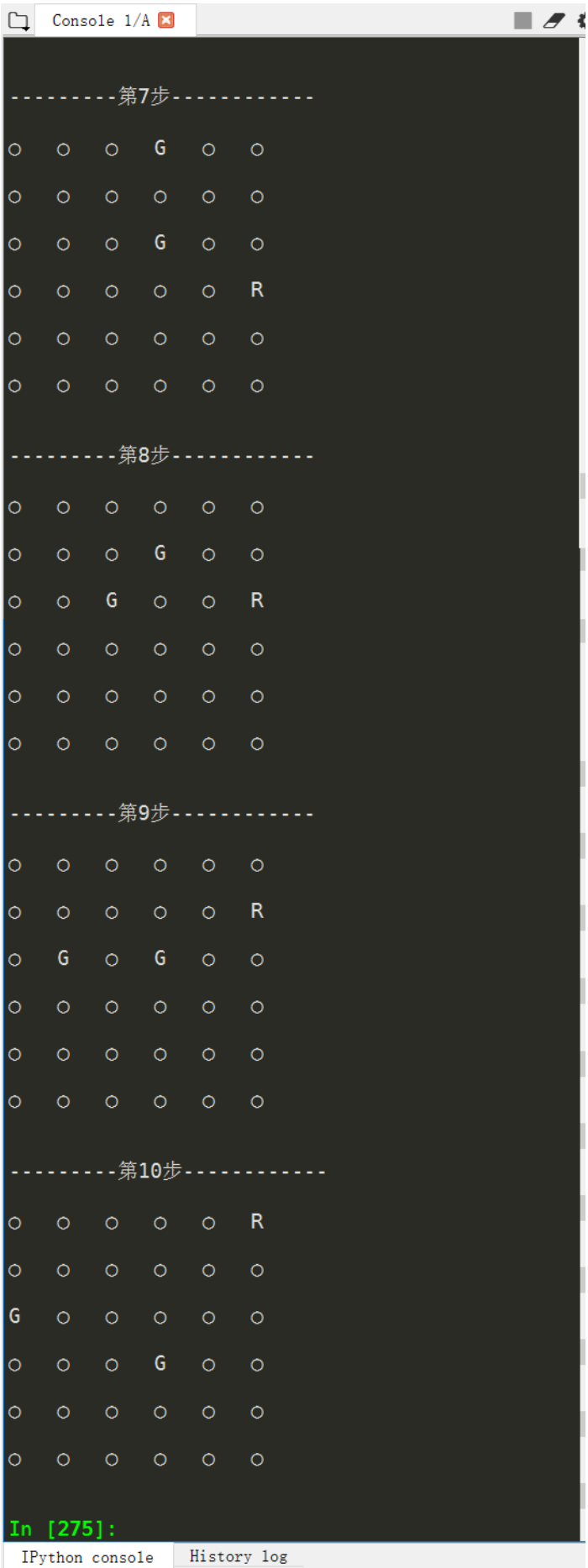
```
97    #打印路径
98    def printway():
99        for n in range(len(r.robot_way)):
100           print("\n---------第{}步------------\n".format(n+1))
101           for i in range(5,-1,-1):
102               for j in range(6):
103                   if i == r.robot_way[n][1]-1 and j == r.robot_way[n][0]-1:
104                       print("R   ",end='')
105                   elif (i == g.ghoust_way[n][1]-1 and j == g.ghoust_way[n][0]-1 ) or (i == g.ghoust_way[n
106                       print("G   ",end='')
107                   else :
108                       print("o   ",end='')
109                   if j == 5:
110                       print("\n")
111
112   printway()
113   #print("输入printway()查看路径")
```

6. 最后写个函数打印路径即可

最终结果输出路径（图比较长，所以分了三次截图）

```
In [274]: runfile('C:/Users/70882/Desktop/代码/
人工智能作业/wenti1_main.py', wdir='C:/Users/
70882/Desktop/代码/人工智能作业')
Reloaded modules: ghoust_robot,
coolanddead_imagineghoust

---------第1步------------

○   ○   ○   ○   ○   ○

○   ○   ○   ○   ○   ○

○   G   ○   ○   ○   ○

○   ○   ○   ○   ○   ○

○   ○   ○   G   ○   ○

○   R   ○   ○   ○   ○


---------第2步------------

○   ○   ○   ○   ○   ○

○   ○   ○   ○   ○   ○

○   ○   G   ○   ○   ○

○   ○   ○   ○   ○   ○

○   R   ○   ○   ○   ○

○   ○   ○   G   ○   ○


---------第3步------------

○   ○   ○   ○   ○   ○

○   ○   ○   ○   ○   ○

○   ○   ○   G   ○   ○

○   ○   ○   ○   ○   ○

○   ○   R   G   ○   ○

○   ○   ○   ○   ○   ○


---------第4步------------

○   ○   ○   ○   ○   ○

○   ○   ○   ○   ○   ○

○   ○   ○   ○   G   ○
```


IPython console
Console 1/A
---------第4步------------
O O O O O O
O O O O O O
O O O O G O
O O R G O O
O O O O O O
O O O O O O
---------第5步------------
O O O O O O
O O O O O O
O O O G O G
O O O R O O
O O O O O O
O O O O O O
---------第6步------------
O O O O O O
O O O G O O
O O O O G O
O O O O R O
O O O O O O
O O O O O O
---------第7步------------
O O O G O O
O O O O O O
O O O G O O


Console 1/A
---------第7步------------
O O O G O O
O O O O O O
O O O G O O
O O O O O R
O O O O O O
O O O O O O
---------第8步------------
O O O O O O
O O O G O O
O O G O O R
O O O O O O
O O O O O O
O O O O O O
---------第9步------------
O O O O O O
O O O O O R
O G O G O O
O O O O O O
O O O O O O
O O O O O O
---------第10步------------
O O O O O R
O O O O O O
G O O O O O
O O O G O O
O O O O O O
O O O O O O
In [275]:
IPython console
History log

问题二

总体思路:

总体思路与第一步类似，只不过要多判断一下是否撞墙、是否陷入死角、是否进入老路

具体操作步骤:

1. 将图中的墙壁读入列表，为了方便调用与代码整洁，墙的初始化函数 wall()定义在 wall.py 文件中，方便调用。墙在两个房间之间，所以用 0.5 来确定墙壁的位置，比如[2.5,1]

```
25 if __name__=='__main__':
26
27     #初始化墙的位置，边界和内部的墙
28     wall = wall()
29
```

2. 墙壁定义完之后，初始化鬼与幽灵，与第一问相同

3. 仍然是使用 A*搜索的原则，即只进行向右、向上或者停止动作，(向右或者向上优

先级相同）（但没有真正实现 A*搜索）

```
while ab != [6,6]:
    #定义走位向量
    v1 = random.choice([[1,0],[0,1]])
    v2 = v1[::-1]
    v3 = [-1,0]
    v4 = [0,-1]

    if (dead(v1,imagin_ghoust,ab) == 0)  and (iswall(ab,v1) == 0) and (isold(ab,v1,r.robot_way) == 
        #右或上
        xy = g.move(xy)
        imagin_ghoust = imagine_walk(imagin_ghoust)
        ab = r.move(v1)
        imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)

        if (xy[0] == ab[0] and xy[1] == ab[1]) or (xy[2] == ab[0] and xy[3] == ab[1]):

            fail = fail + 1
            print("1")
            print(r.robot_way)

            break

    elif (dead(v2,imagin_ghoust,ab) == 0) and iswall(ab,v2) == 0 and isold(ab,v2,r.robot_way) == 0:

        xy = g.move(xy)
        imagin_ghoust = imagine_walk(imagin_ghoust)
        ab = r.move(v2)
        imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)

        if (xy[0] == ab[0] and xy[1] == ab[1]) or (xy[2] == ab[0] and xy[3] == ab[1]):

            fail = fail + 1
            print("2")
            print(r.robot_way)

            break
```

4. 因为我的知识库是非常聪明的，不会出现被鬼卡两步的情况；因为鬼只有一种可能吃掉机器人，最坏的情况下，机器人停留一步就知道这个鬼是什么情况了，所以停留两步就一定是卡进了死胡同，那就让他退回去叭

```
        #如果等了两步
        if len(r.robot_way)>3:
            if r.robot_way[-1] == r.robot_way[-3]:

                #如果走到死胡同，就往回走一步
                xy = g.move(xy)
                imagin_ghoust = imagine_walk(imagin_ghoust)
                v = list(np.array(r.robot_way[-4])-np.array(r.robot_way[-1]))
                ab = r.move(v)
                imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)


    #循环结束，记录步数
    wholeway.append(r.robot_way)
    wholetime.append(time)
```

5. 然后记录所有的路径和时间耗散，排序，计算成功率，得到最短的两条路径

```
成功率是：0.758
成功路径的平均时间是11.3125
两条最短的成功路径是：
[[1, 2], [2, 2], [2, 3], [3, 3], [4, 3], [5, 3],
[6, 3], [6, 4], [6, 5], [6, 6]]
[[2, 1], [2, 2], [2, 3], [3, 3], [4, 3], [5, 3],
[6, 3], [6, 4], [6, 5], [6, 6]]
```

6. 结果发现，这种方法成功率比较低，调试发现，这种方法容易停在沟沟角角，从而被鬼撞死，于是加入一个判断，就是如果停一步会死，那就保命要紧，哪个方向能走就往哪个方向跑

```
        #如果等着会死，那么保命要紧
        else:

            if (dead(v1,imagin_ghoust,ab) == 0)  and (iswall(ab,v1) == 0) :

                xy = g.move(xy)
                imagin_ghoust = imagine_walk(imagin_ghoust)
                ab = r.move(v1)
                imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)

                if (xy[0] == ab[0] and xy[1] == ab[1]) or  (xy[2] == ab[0] and xy[3] == ab[1]):
    #                print("***************fail***************")
                    print(r.robot_way)
                    print("3")
                    fail = fail + 1
                    break

            elif (dead(v2,imagin_ghoust,ab) == 0) and iswall(ab,v2) == 0:

                xy = g.move(xy)
                imagin_ghoust = imagine_walk(imagin_ghoust)
                ab = r.move(v2)
                imagin_ghoust = select_ghoust(ab, cool(xy,ab),imagin_ghoust)

                if (xy[0] == ab[0] and xy[1] == ab[1]) or (xy[2] == ab[0] and xy[3] == ab[1]):

                    fail = fail + 1
                    print("2")
```

7. 最终结果成功率变为了 1，但是平均路径耗散也大约提高了 1，

```
In [36]: runfile('C:/Users/70882/Desktop/代码/人工
智能作业/wenti2_main.py', wdir='C:/Users/70882/
Desktop/代码/人工智能作业')
Reloaded modules: ghoust_robot,
coolanddead_imagineghoust, iswallandisold, wall
成功率是：1.0
成功路径的平均时间是12.522
两条最短的成功路径是：
[[2, 1], [2, 2], [2, 3], [3, 3], [4, 3], [5, 3],
[6, 3], [6, 4], [6, 5], [6, 6]]
[[2, 1], [2, 2], [3, 2], [4, 2], [4, 3], [5, 3],
[6, 3], [6, 4], [6, 5], [6, 6]]
```

8. 也许这就是要在时间与准确率之间权衡叭

问题三

问题三的总体思路：

1. 让机器人上下左右随机走路
2. 遍历足够多次
3. 然后将机器人的路径与鬼行走路径比对，如果有一个路径相同，那就是一条失败的路径
4. 将成功的路径添加进数组，然后排序
5. 得到路径耗散最低的一条路，打印出来

   这个方法非常笨拙，但是有了问题一问题二的基础，这种方法是最容易实现的

具体实现方法：

1. 初始化鬼和机器人的方法和问题一问题二相同

```
choseible_way = [[1,0],[0,1],[-1,0],[0,-1]]
wholeway = []

for i in range(10000):

    robot_orgin = [1,1]
    path = []

    for j in range(20):

        rado = random.choice(choseible_way)
        fakewalk = list(np.array(robot_orgin) + np.array(rado))

        if (iswall(robot_orgin,rado) is 0) and fakewalk[0]<=6 and fakewalk[1]<=6 \
            and fakewalk[0]> 0 and fakewalk[1]> 0 and isold(robot_orgin,rado,path) == 0:

            robot_orgin = list(np.array(robot_orgin) + np.array(rado))
            path.append(robot_orgin)

        if  robot_orgin == [6,6]:

            wholeway.append(path)

            break
```

2. 通过，控制 i 和 j 的大小，控制计算次数，j 要大于 10，因为最短的路径就是 10

3. 清理数据，把重复的路径、遇到鬼的路径去掉，排序得到前三短的路径（但是问题只要求了最短的）

```
#去掉重复的路
a = []
for i in wholeway:
    if i not in a:

        a.append(i)

wholeway = a

#得到ghoust的路线
whole_ghoust = []

for i in range(20):

    xy = g.move(xy)
    a = []
    a = copy.deepcopy(xy)
    whole_ghoust.append(a)

#去掉遇到ghoust的路
i,j = 0,0
while i < len(wholeway):
    while j < len(wholeway[i]):
        if (wholeway[i][j][0] == whole_ghoust[j][0] and wholeway[i][j][1] == 4) \
            or (wholeway[i][j][1] == whole_ghoust[j][3] and wholeway[i][j][0] == 4):

            wholeway.pop(i)
            i = i
            j = 0

        j = j +1

    i = i +1

#找到路线最短的3条路
lenth = []

for i in range(len(wholeway)):

    lenth.append(len(wholeway[i]))

index = np.argsort(lenth)

top3 = []
#取前3条路
for i in range(3):

    top3.append(wholeway[index[i]])


top = top3[0]

#打印
def printway():

    for n in range(len(top)):
```

4. 最后打印出路径即可，打印方法与问题一一致

```
---------第1步------------
O   O   O   O   O   O
O   O   O   O   O   O
G   O   O   O   O   O
O   O   O   O   O   O
O   O   O   G   O   O
O   R   O   O   O   O

---------第2步------------
O   O   O   O   O   O
O   O   O   O   O   O
O   G   O   O   O   O
O   O   O   G   O   O
O   R   O   O   O   O
O   O   O   O   O   O

---------第3步------------
O   O   O   O   O   O
O   O   O   O   O   O
O   O   G   G   O   O
O   O   O   O   O   O
O   O   R   O   O   O
O   O   O   O   O   O

---------第4步------------
O   O   O   O   O   O
O   O   O   G   O   O
O   O   O   G   O   O
O   O   R   O   O   O
O   O   O   O   O   O
O   O   O   O   O   O

---------第5步------------
```

```
O   O   O   O   O   O

---------第5步------------

O   O   O   G   O   O

O   O   O   O   O   O

O   O   R   O   G   O

O   O   O   O   O   O

O   O   O   O   O   O

O   O   O   O   O   O

---------第6步------------

O   O   O   O   O   O

O   O   R   G   O   O

O   O   O   O   O   G

O   O   O   O   O   O

O   O   O   O   O   O

O   O   O   O   O   O

---------第7步------------

O   O   R   O   O   O

O   O   O   O   O   O

O   O   O   G   G   O

O   O   O   O   O   O

O   O   O   O   O   O

O   O   O   O   O   O

---------第8步------------

O   O   O   R   O   O

O   O   O   O   O   O

O   O   O   G   O   O

O   O   O   G   O   O

O   O   O   O   O   O

O   O   O   O   O   O
```

```
O   O   O   O   O   O
O   O   O   G   G   O
O   O   O   O   O   O
O   O   O   O   O   O
O   O   O   O   O   O

---------第8步------------
O   O   O   R   O   O
O   O   O   O   O   O
O   O   O   G   O   O
O   O   O   G   O   O
O   O   O   O   O   O
O   O   O   O   O   O

---------第9步------------
O   O   O   O   R   O
O   O   O   O   O   O
O   O   G   O   O   O
O   O   O   O   O   O
O   O   O   G   O   O
O   O   O   O   O   O

---------第10步------------
O   O   O   O   O   R
O   O   O   O   O   O
O   G   O   O   O   O
O   O   O   O   O   O
O   O   O   O   O   O
O   O   O   G   O   O

In [38]:
```

总结：

1. 我将函数打包成 py 文件，在写代码时直接 import 文件，这样使主函数行数非常少，而且更加简洁；但是打包时就要注意函数的编写，尽量让函数独立，这样可移植性才强
2. 问题三解决的非常笨拙，使用的是运算时间空间换取求解效果的方法，A*搜索实现起来效果应该非常好，但是我尝试了并没有写成，所以还是采用第一版的暴力穷举法；以后有时间将 A*搜索开发出来。
3. 问题一与问题二解决的较好，主要是因为使用了知识库的推理。其中在编写代码过程中，最需要注意的是深层拷贝的理解，我就曾因为这个知识没能理解，浪费了 1 个小时的调试时间。