八皇后问题程序报告

学号： 2111460 姓名： 张洋

一、问题重述

八皇后问题：如何能在 8*8 的国际象棋棋盘上放置八个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？为了到达此目的，任两个皇后都不能处于同一条横行、纵行或斜线上。

自由定义函数，以序列格式存储结果输出，例如 [0,6,4,7,1,3,5,2] 指代在第 0-7 行依次在指定列落子，并给出尽可能多的求解，通过 self.solves.append([0,6,4,7,1,3,5,2]) 添加求解结果，最后的结果以解的 list 格式输出。

二、设计思想

回溯算法：
1.定义状态：用一个长度为 8 的数组 list 存储每个皇后的列数，其中 list[i]表示第 i 行的皇后所在的列数。
2.递归函数：定义一个递归函数 queen(self , list , row)，其中 row 表示当前处理的行数。
3.终止条件：当处理到最后一行时，说明找到了一组解，将当前的 list 数组通过 self.solves.append 加入到结果列表中。
4.回溯过程：
a. 遍历当前行的每一列，对于每一列，判断当前位置是否可以放置皇后。
b. 判断当前位置是否可以放置皇后调用 put(self, list, row)，其中 row 表示当前处理的行数。题目中要求任两个皇后都不能处于同一条横行、纵行或斜线上，通过编写条件判断语句判断当前位置是否可以放置皇后。
c. 如果当前位置可以放置皇后，则放置皇后之后继续处理下一行，即调用 queen(self , list , row+1)。
d. 如果处理下一行后无法找到解，则回溯到当前行，尝试将皇后放置在当前行的下一列。
e. 如果处理下一行后找到解，则返回结果。
5.返回结果：将所有解存储在一个列表中，返回该列表即可。

三、代码内容

```
1.  import numpy as np            # 提供维度数组与矩阵运算
2.  import copy                   # 从 copy 模块导入深度拷贝方法
3.  from board import Chessboard
4.
5.  # 基于棋盘类，设计搜索策略
6.  class Game:
7.      def __init__(self, show = True):
8.          """
```

```
        初始化游戏状态.
        """

        self.chessBoard = Chessboard(show)
        self.solves = []
        self.gameInit()

    # 重置游戏
    def gameInit(self, show = True):
        """
        重置棋盘.
        """

        self.Queen_setRow = [-1] * 8
        self.chessBoard.boardInit(False)

    ##########################################################################
    ####             请在以下区域中作答(可自由添加自定义函数)              ####
    ####          输出: self.solves = 八皇后所有序列解的 list           ####
    ####          如:[[0,6,4,7,1,3,5,2],]代表八皇后的一个解为          ####
    ####      (0,0),(1,6),(2,4),(3,7),(4,1),(5,3),(6,5),(7,2)        ####
    ##########################################################################
    #                                                                        #
    def put(self, list, row): #list 为存储结果的列表，row 为当前行
        for i in range(row):
            #判断当前列是不是只对应当前行并且判断当前位置的斜线方向有无棋子
            if list[i] == list[row] or abs(list[row] - list[i]) == abs(row -
    i):
                return False
        return True

    def queen(self, list, row):
        x = [] #存放结果
        if row == 8:
            x.extend(list)
            self.solves.append(x)
            return
        for i in range(8): #i 表示所在列
            list[row] = i
            if self.put(list, row): #判断当前位置是否能够放棋子
                self.queen(list, row + 1) #上一行的棋子放完后执行递归进入下一
    行

    def run(self, row=0):
```

```
        list = [0 for i in range(8)]
        self.queen(list, row)

    #                                                                      #
    ########################################################################
    #################         完成后请记得提交作业          #################
    ########################################################################

    def showResults(self, result):
        """
        结果展示.
        """

        self.chessBoard.boardInit(False)
        for i,item in enumerate(result):
            if item >= 0:
                self.chessBoard.setQueen(i,item,False)

        self.chessBoard.printChessboard(False)

    def get_results(self):
        """
        输出结果(请勿修改此函数).
        return: 八皇后的序列解的 list.
        """

        self.run()
        return self.solves
================================================================================
game = Game()
solutions = game.get_results()
print('There are {} results.'.format(len(solutions)))
game.showResults(solutions[0])

```

四、实验结果

运行输出结果如下，可以通过 game.showResults(solutions[i])来可视化输出所有解，这里只输出了第一个解。

```
  0 1 2 3 4 5 6 7
0 - - - - - - - -
1 - - - - - - - -
2 - - - - - - - -
3 - - - - - - - -
4 - - - - - - - -
5 - - - - - - - -
6 - - - - - - - -
7 - - - - - - - -
There are 92 results.
  0 1 2 3 4 5 6 7
0 o - - - - - - -
1 - - - - o - - -
2 - - - - - - - o
3 - - - - - o - -
4 - - o - - - - -
5 - - - - - - o -
6 - o - - - - - -
7 - - - o - - - -
```

五、总结

本次实验已经达到了预期目标，通过了测试用例，给出了正确结果。

可能改进的方向：可以在掌握更多知识后自己定义逻辑语法，提高性能。通过更深入的学习和搜索，也可以通过其他算法来完成八皇后问题的求解：

（1）排列组合

在上面的基础上，可以稍微转化一下思路，放置不同的皇后位置就是将不同的八个元素的作排列组合问题，然后只要再保证"不在同一斜线"这个条件就可以了。下面是代码：

```
list1=[0,1,2,3,4,5,6,7]
result=[]
final_result=[]
from itertools import  permutations  #permutations 函数：可以对集合或字符串进行排序或排列的所有可能的组合
for i in permutations(list1,8):
    result.append(i)                #转化为排列组合问题，result 存储部分可能的结果

#下面只需要从 result 中找出满足‘任一两个皇后不在同一斜线上’的结果
for i in result:
    n=0                    #n 用以标记那些满足条件的结果；
    for j in enumerate(i):
        for m in enumerate(i):
            if  m!=j and abs(m[0]-j[0])==abs(m[1]-j[1]):  #m!=j 是避免对同一个坐标位置进行判断
```

```
                n+=1                                    #只要有两个坐标的连线斜率为1，n就不为0
    if n==0:
        final_result.append(i)

print(len(final_result))
print(final_result)
```

（2）遗传算法

遗传算法将要解决的问题模拟成一个生物进化的过程，通过复制、交叉、突变等操作产生下一代的解，并逐步淘汰掉适应度函数值低的解，增加适应度函数值高的解。这样进化 N 代后就很有可能会进化出适应度函数值很高的个体。

其计算步骤如下：

1） 编码：将问题空间转换为遗传空间；

2）创建初始种群：随机生成 P 个染色体作为初始种群；

3）适应度计算：染色体评价，计算各个染色体的适应度；

4）选择操作：根据染色体适应度，按选择算子进行染色体的选择；

5）交叉操作：按交叉概率进行交叉操作；

6）变异操作：按变异概率进行变异操作；

7）返回（4）形成新的种群，继续迭代，直到满足终止条件。

实验过程中遇到的困难：

（1） 编写函数的过程中有些代码的逻辑不太正确，用了很长时间查找错误。

（2） 不了解 board 包中的类和函数定义，本题中使用的 Chessboard 类就来自 board.py，其主要功能有:

1. 八皇后棋盘绘制
2. 胜负条件判定
3. 合法落子点判定
4. 玩家互动接口