程序报告

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。

在给出的chessboard的基础上，设计算法，求解八皇后问题。

二、设计思路

利用回溯算法，寻找所有的摆放方式。

采用的主算法及其注释：

```
A=[0]*(8+1)          # 本行的这一列是否已经落子
B=[0]*(3*8+1)        # /对角线标记
C=[0]*(2*8+1)        # \对角线标记
k=[[0]*9 for _ in range(9)]   # chessboard

def lay(self,i) :
       j = 1
       while (j <= 8):
              # 如果这一列、/对角线、\对角线 都没有落子
              if A[j] + B[j - i+8] + C[j + i] == 0 :
                     # 落子，并且禁用斜对角线
                     k[i][j] =1
                     A[j] =1
                     B[j - i+8] =1
                     C[j + i] =1
                     # 如果还不是最后一行，继续递归
                     if i < 8:
                            self.lay(i + 1)
                     else :
                            # 如果已经是最后一行了
                            # tmp是最后会被append到solutions之中的
                            tmp=[]
                            for u in range(1,9):
                                   for t in range(1,9):
                                          if k[u][t]==1:
                                                 tmp.append(t-1)
                            self.solves.append(tmp)
                     # 从递归处回来或者append完，恢复为0
                     A[j] =0
                     B[j - i+8] =0
                     C[j + i] = 0
                     k[i][j] = 0
              j=j+1
```

三、代码内容

```
import numpy as np            # 提供维度数组与矩阵运算
import copy                   # 从copy模块导入深度拷贝方法
from board import Chessboard

A=[0]*(8+1)
B=[0]*(3*8+1)
C=[0]*(3*8+1)
k=[[0]*9 for _ in range(9)]

# 基于棋盘类，设计搜索策略
class Game:
    def __init__(self, show = True):
        """
        初始化游戏状态.
        """

        self.chessBoard = Chessboard(show)          # a chessBoard
        self.solves = []                            # solves is a list
        self.gameInit()

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

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

    def run(self, row=0):
        self.lay(1)

    # 回溯算法
    def lay(self,i) :
       j = 1
       while (j <= 8):
              if A[j] + B[j - i+8] + C[j + i] == 0 :
                     k[i][j] =1
                     A[j] =1
                     B[j - i+8] =1
                     C[j + i] =1
                     if i < 8:
                            self.lay(i + 1)
                     else :
                            tmp=[]
                            for u in range(1,9):
                                   for t in range(1,9):
                                          if k[u][t]==1:
                                                 tmp.append(t-1)
                            self.solves.append(tmp)
                     A[j] =0
                     B[j - i+8] =0
                     C[j + i] = 0
                     k[i][j] = 0
              j=j+1
```

```
    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)  # draw the chessboard

    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)))
```

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

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

五、总结

是否达到目标预期：是

可能改进的方向：提高算法性能

实验过程中遇到的困难：对于python语言的不熟悉，对于回溯算法的不了解。