南開大學
Nankai University

人工智能学院

人工智能技术实验报告

实验一　八皇后问题求解

姓名：石若川

学号：2111381

专业：智能科学与技术

2023 年 9 月 19 日

1 问题简述

八皇后问题，是一个古老而著名的问题：如何能够在 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？为了达到此目的，任两个皇后都不能处于同一条横行、纵行或斜线上。

最早是由国际西洋棋棋手马克斯 · 贝瑟尔于 1848 年提出，之后陆续有数学家对其进行研究，其中包括高斯和康托。

八皇后问题可以推广为更一般的 N 皇后摆放问题：这时棋盘的大小变为 N×N，而皇后个数也变成 N。

2 实验目的

1. 通过求解皇后问题，熟悉深度优先搜索法技术；
2. 理解递归回溯算法思想，进而推广到 n 皇后问题；（选做）
3. 对实验进行图形化界面设计，实现按步或按解的展示。（选做）

3 实验内容

1. 实现八皇后问题的解法统计；
2. 将八皇后问题推广到 N 皇后；（选做）
3. 图形化界面设计。（选做）

4 编译环境

编译环境为 Windows 11 下 Python3.6，图形化界面利用了 Pyqt5 进行可视化。

5 实验步骤

5.1 回溯算法求解

从第一行开始，尝试在每一列中放置一个皇后。在每一步中，检查当前位置是否能够避免被攻击，即是否和之前摆放的皇后在同一行、同一列或同一对角线上。如果可以放置，就继续到下一行；如果无法放置，就回溯到上一行，尝试将皇后摆放在下一个列。终止条件为：当所有皇后都被成功放置在棋盘上，即当第 N 行成功摆放上皇后时，就找到了一个解。

以三皇后问题为例，流程图如图5.1所示：

- 初始状态皇后摆放在（1,1）位置
- 在第二行中摆放皇后，（2,1）（2,2）位置均不满足条件，将第二枚皇后摆放在（2,3）位置
- 在第三行中摆放皇后，（3,1）（3,2）（3,3）位置均不满足条件
- 第一枚皇后摆放在（1,1）位置的情况无解

- 回溯到第一枚皇后的摆放，将皇后摆放到（1,2）位置
- 在第二行中摆放皇后，（2,1）（2,2）（2,3）位置均不满足条件
- 第一枚皇后摆放在（1,2）位置的情况无解
- 回溯到第一枚皇后的摆放，将皇后摆放在（1,3）位置
- 在第二行中摆放皇后，（2,2）（2,3）位置均不满足条件，将第二枚皇后摆放在（2,1）位置
- 在第三行中摆放皇后，（3,1）（3,2）（3,3）位置均不满足条件
- 第一枚皇后摆放在（1,2）位置的情况无解
- 三皇后问题无解

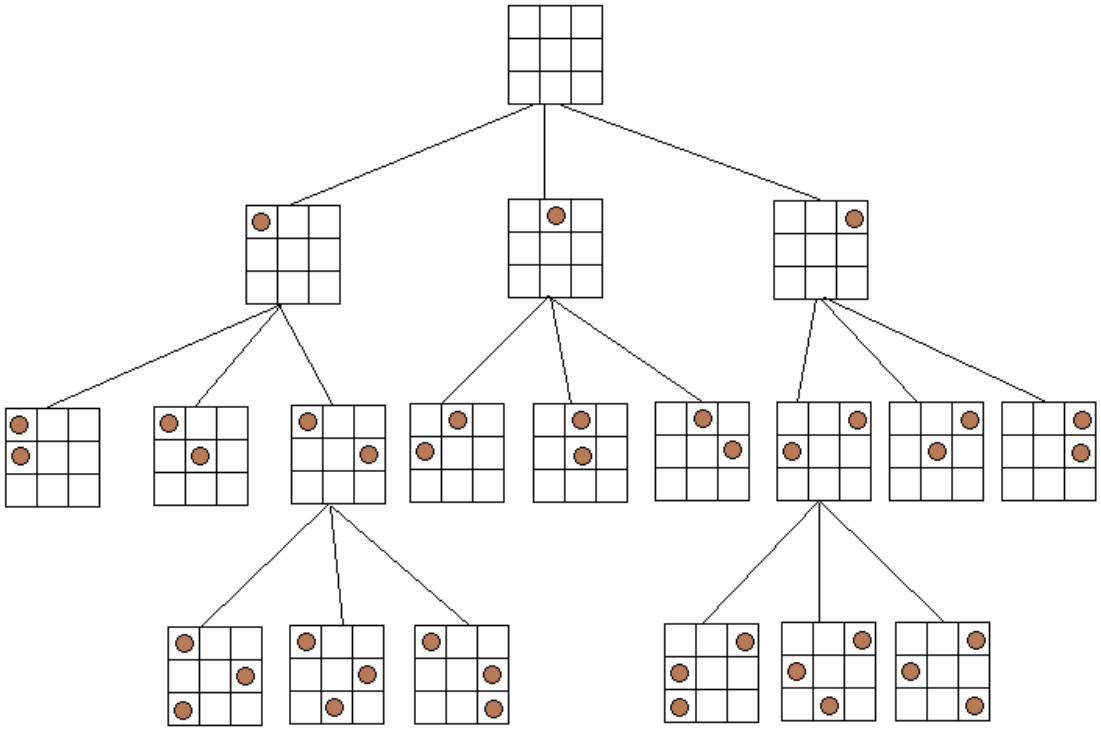


图 5.1: 三皇后问题流程图

5.2 可视化界面的设计

5.2.1 基本功能按钮

利用 QlineEdit 函数生成皇后数量 N 的输入框，利用 QPushButton 函数生成 “开始搜索” “退出” 和 “输出所有解” 的按钮并利用 connect 函数与对应的自定义函数相关联，利用 move 和 setGeometry 函数调整各组件和窗口的位置。

```
def initUI(self):
    self.label_numberbox = QLabel('皇后数量：', self)
    self.label_numberbox.move(440, 20)
    self.label_numberbox.resize(60, 20)
```

```
    # 皇后的数量N
    self.number_input = QLineEdit(self)
    self.number_input.move(500, 20)
    self.number_input.resize(40, 20)

    # 耗时
    self.time_text = QLabel('搜索耗时: ', self)
    self.time_used = QLabel(self)
    self.time_text.move(440, 100)
    self.time_used.move(530, 100)

    # 开始与退出
    Start_Button = QPushButton('开始搜索',self)
    Exit_Button = QPushButton('退出',self)
    Start_Button.clicked.connect(self.Start)
    Exit_Button.clicked.connect(self.Stop)
    Start_Button.move(475, 140)
    Exit_Button.move(475, 200)

    # 输出所有解
    OutputButton = QPushButton('输出所有解', self)
    OutputButton.pressed.connect(self.Output_Solution)
    OutputButton.move(475, 260)

    self.setGeometry(300, 300, 620, 410)
    self.setWindowTitle('N-皇后问题')
    self.show()
```

5.2.2 棋盘的绘制

已知皇后个数 N，利用 drawRect 函数绘制每个棋盘格矩形，利用 setBrush 函数设置棋盘格颜色。设置棋盘的奇数行奇数列和偶数行偶数列为灰色（128,128,128），奇数行偶数列和偶数行奇数列为白色（255,255,255）。

```
def Draw_Board(self):
    qp = QPainter()
    qp.begin(self)

    pen = QPen(Qt.black, 2, Qt.SolidLine)
    qp.setPen(pen)
    qp.drawLine(5, 5, 405, 5)
```

```
    qp.drawLine(5, 5, 5, 405)
    qp.drawLine(5, 405, 405, 405)
    qp.drawLine(405, 5, 405, 405)

    temp = int(400 / self.Num_of_Queens)

    for i in range(self.Num_of_Queens):
        for j in range(self.Num_of_Queens):
            if (i % 2 == 0 and j % 2 == 0) or (i % 2 == 1 and j % 2 == 1):
                qp.setBrush(QColor(128,128,128))
                qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)
            else:
                qp.setBrush(QColor(255, 255, 255))
                qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)

qp.end()
```

5.2.3 棋子的绘制

在绘制棋盘的基础上，利用 QPixmap 函数加载皇后棋子图片 queen.jpg，利用 scaled 函数调整图片大小以适应棋盘格的大小，再通过 setPixmap 和 setGeometry 函数将图片添加到解的位置（x, y）。

```
def Draw_Chess(self):
    temp = int(400 / self.Num_of_Queens) # 棋盘格大小
    qp = QPainter()
    qp.begin(self)

    pen = QPen(Qt.black, 2, Qt.SolidLine)
    qp.setPen(pen)
    qp.drawLine(5, 5, 405, 5)
    qp.drawLine(5, 5, 5, 405)
    qp.drawLine(5, 405, 405, 405)
    qp.drawLine(405, 5, 405, 405)

    for i in range(self.Num_of_Queens):
        for j in range(self.Num_of_Queens):
            if (i % 2 == 0 and j % 2 == 0) or (i % 2 == 1 and j % 2 == 1):
                qp.setBrush(QColor(128, 128, 128))
                qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)
            else:
                qp.setBrush(QColor(255, 255, 255))
                qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)
```

```
21
22      for i in range(self.Num_of_Queens):
23          x = 5 + temp * i
24          y = 5 + temp * (self.current_result[i])
25
26          queen = QPixmap('queen.jpg') # 加载皇后图片
27          queen = queen.scaled(temp, temp) # 调整图片大小以适应格子
28          label = QLabel(self)
29          label.setPixmap(queen)
30          label.setGeometry(x, y, temp, temp)
31
32      qp.end()
```

5.2.4 异常警告窗口

当检测到输入 N 值后，先通过 int() 进行转换。若无法转换或转换值小于等于 0，则说明输入值不是正整数，利用 QMessageBox.warning() 函数弹出警告信息。

```
1  def Start(self):
2      queen_input = self.number_input.text()
3      try:
4          Num = int(queen_input)
5          if Num <= 0:
6              # 输入不是正整数，弹出警告窗口
7              QMessageBox.warning(self, '警告',
                   '您输入的皇后数量非法，请输入一个正整数！')
8              self.number_input.clear() # 清空文本框内容
9              return
10         else:
11             self.Draw_Flag = 1
12             self.Num_of_Queens = Num
13             def Get_All_Results(num):
14                 ...
15
16             def Get_All_Results_Current(current_board, all_results):
17                 ...
18
19     except ValueError:
20         # 无法转换为整数，弹出警告窗口
21         QMessageBox.warning(self, '警告', '您输入的皇后数量非法，请输入一个正整数！')
```

6 实验结果

以 N=4 的情况为例：

1. 初始页面如图6.2所示。

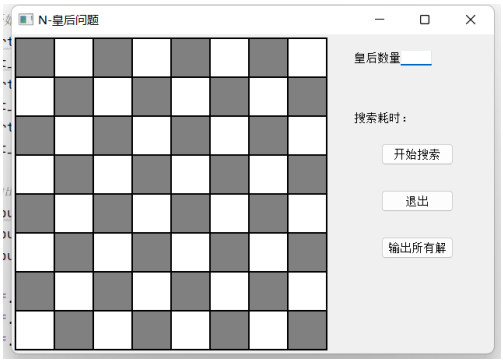
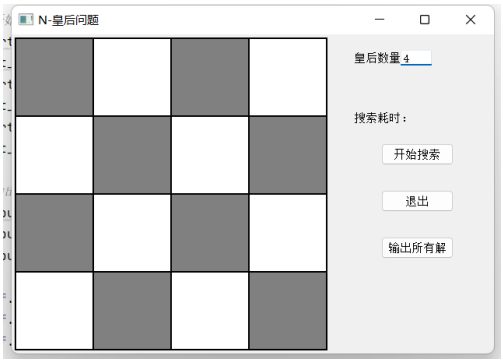


图 6.2: 初始页面

2. 输入皇后数量 N，并点击“开始搜索”，搜索完成后会将有弹窗显示解的数量，如图6.3。

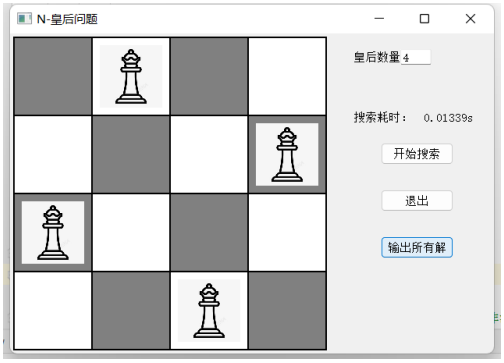


(a) 输入皇后数量 N

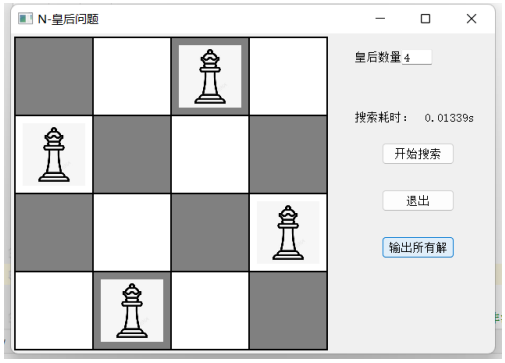
(b) 开始搜索

图 6.3: 输入 N 值开始搜索

3. 点击“输出所有解”，将以 0.5s 展示一种解的频率，打印出所有的解，如图6.4。



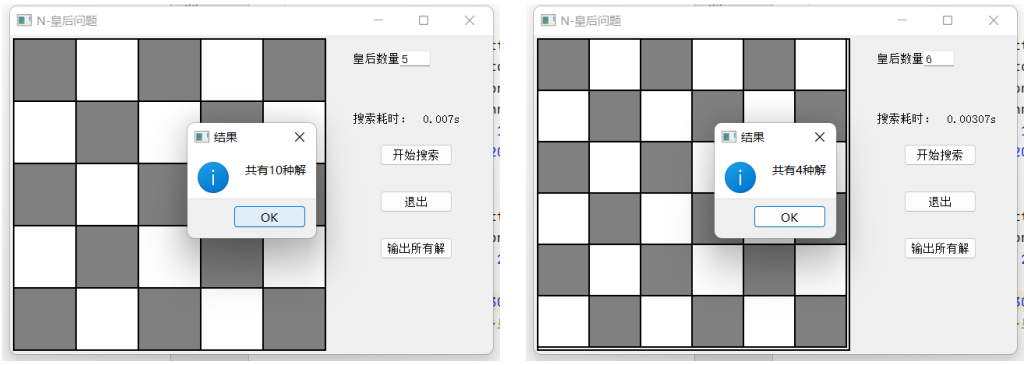
(a) 第 1 种解



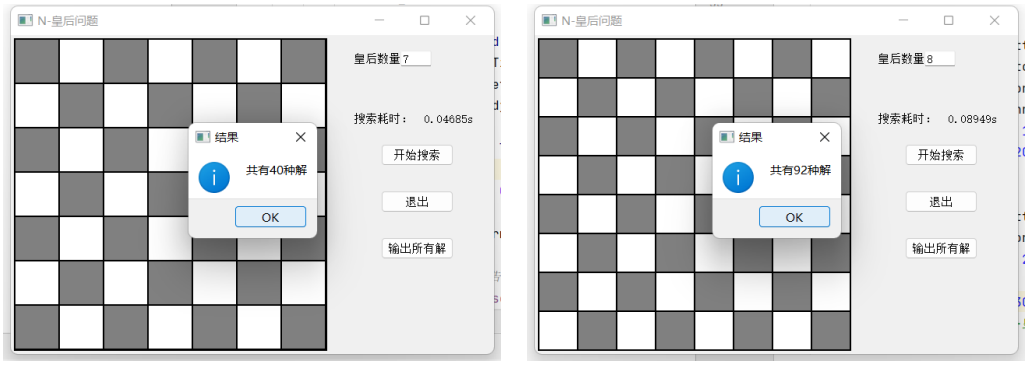
(b) 第 2 种解

图 6.4: 输出所有解

4. 以 N = 5, 6, 7, 8 进行验证，解的个数均正确，如图6.5所示。



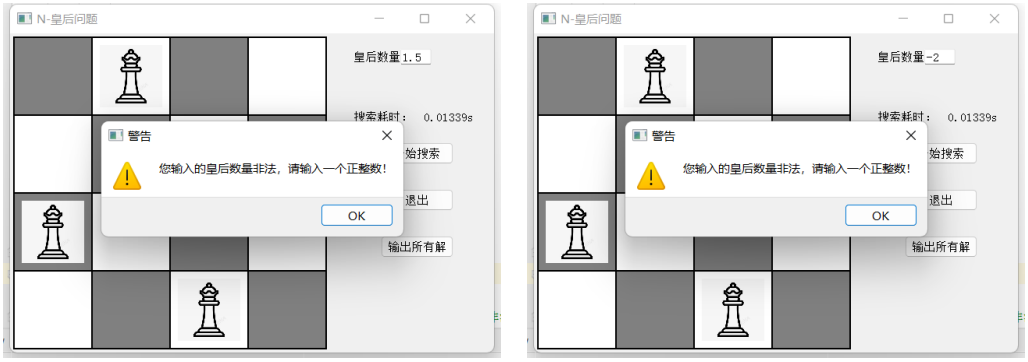
(a) N=5 (b) N=6



(c) N=7 (d) N=8

图 6.5: 验证正确性

5. 由于 N 值只能为正整数，所以当输入值为小数或负值时，将弹出警告，如图6.6所示。



(a) 输入小数 (b) 输入负数

图 6.6: 输入值为异常值

6. 当使用结束后，点击“退出”即可退出页面，如图6.7所示。

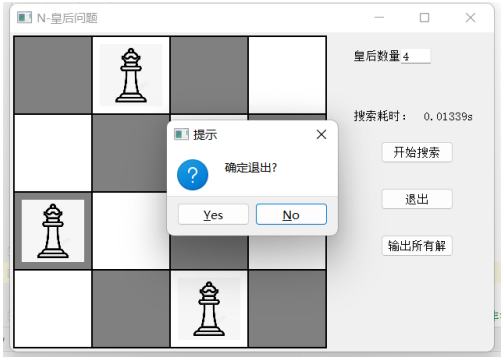


图 6.7: 退出可视化界面

7. 求解结果如表1所示，求解耗时与 N 值的关系图如图6.8所示。分析可得，随着 N 值增加，以回溯算法求解 N 皇后问题的耗时成指数增加，出现组合爆炸问题。因此，在 N 值较大时进行求解需要用到启发式算法，如 A* 算法等。

表 1: 求解结果

N 值	解的个数	求解耗时（s）
4	2	0.002
5	10	0.006
6	4	0.003
7	40	0.03237
8	92	0.08172
9	352	0.36499
10	724	0.87764
11	2680	3.89179
12	14200	21.07787
13	73712	131.74665

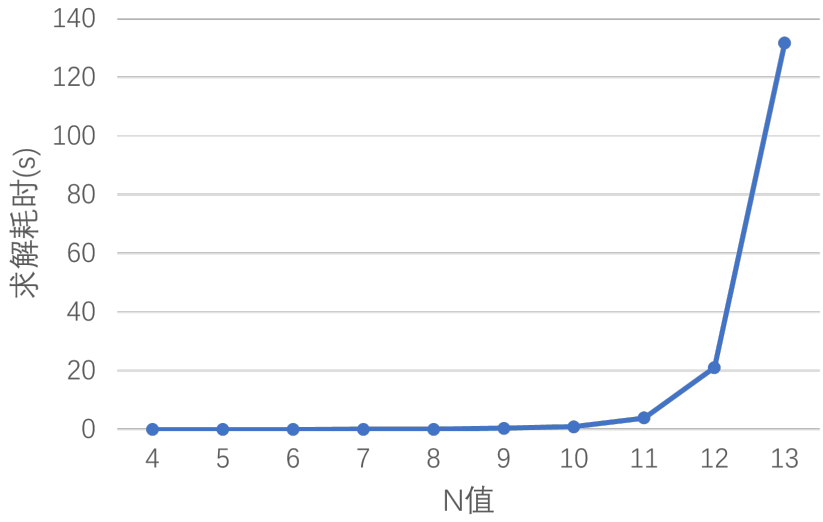


图 6.8: 耗时与 N 值的关系图

7 分析总结

1. 在此之前，我把回溯和递归两个概念混淆，通过本实验中我对两者的概念有了更清晰的认识。递归是一种**算法结构**，递归会出现在子程序中自己调用自己或间接地自己调用自己，例如计算阶乘的算法。回溯是一种**算法思想**，可以用递归实现。回溯就是一种试探的策略，类似于穷举，但回溯有“剪枝”功能，可以在发现当前情况无解时提前回溯到上一种情况。

2. 在本实验中，我学习了 N 皇后回溯算法，初步了解了 Pyqt5 的可视化方法，实现了 N 皇后问题的解决和可视化界面的设计。

3. 程序的不足

   - 未设计提前终止搜索的按键。N 值过大时搜索时间很长，若想要提前终止搜索只能强行退出，会出现 Python 停止工作的情况，不利于程序的稳定运行。
   - 当 N 值过大时，回溯算法无法在短时间内进行求解，需要启发式算法进行求解。

附录 A 代码

```
import sys
import time
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QMessageBox,
    QLabel, QLineEdit
from PyQt5.QtGui import QPainter, QColor, QPen, QPixmap
from PyQt5.QtCore import Qt

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

        self.Num_of_Queens = 8
        self.Draw_Flag = 0
        self.results = dict()
        self.current_result = []

    def initUI(self):
        self.label_numberbox = QLabel('皇后数量: ', self)
        self.label_numberbox.move(440, 20)
        self.label_numberbox.resize(60, 20)

        # 皇后的数量N
        self.number_input = QLineEdit(self)
        self.number_input.move(500, 20)
        self.number_input.resize(40, 20)

        # 耗时
        self.time_text = QLabel('搜索耗时: ', self)
        self.time_used = QLabel(self)
        self.time_text.move(440, 100)
        self.time_used.move(530, 100)

        # 开始与退出
        Start_Button = QPushButton('开始搜索',self)
        Exit_Button = QPushButton('退出',self)
        Start_Button.clicked.connect(self.Start)
        Exit_Button.clicked.connect(self.Exit)
        Start_Button.move(475, 140)
        Exit_Button.move(475, 200)
```

```
40
41          # 输出所有解
42          OutputButton = QPushButton('输出所有解', self)
43          OutputButton.pressed.connect(self.solution)
44          OutputButton.move(475, 260)
45
46          self.setGeometry(300, 300, 620, 410)
47          self.setWindowTitle('N-皇后问题')
48          self.show()
49
50      def paintEvent(self, e):
51          if self.Draw_Flag == 0:
52              self.Draw_Board()
53          else:
54              self.Draw_Chess()
55
56
57      def Start(self):
58          queen_input = self.number_input.text()
59          try:
60              Num = int(queen_input)
61              if Num <= 0:
62                  # 输入不是正整数，弹出警告窗口
63                  QMessageBox.warning(self, '警告',
                        '您输入的皇后数量非法，请输入一个正整数！')
64                  self.number_input.clear() # 清空文本框内容
65                  return
66              else:
67                  self.Draw_Flag = 1
68                  self.Num_of_Queens = Num
69
70
71                  time_begin = time.time() # 开始计时
72                  self.results = self.Get_All_Results(self.Num_of_Queens) # 求解
73                  time_end = time.time() # 结束计时
74
75                  Time_cost = round(time_end - time_begin, 5)
76                  Time_cost = str(Time_cost)
77                  self.time_used.setText((Time_cost) + 's')
78                  self.time_used.adjustSize()
79
80                  Num_of_results = len(self.results)
```

```
81
82              self.Draw_Flag = 0
83
84              QMessageBox.information(self, '结果', '共有' + str(Num_of_results) +
                    '种解')
85          except ValueError:
86              # 无法转换为整数，弹出警告窗口
87              QMessageBox.warning(self, '警告',
                    '您输入的皇后数量非法，请输入一个正整数！')
88
89      def Get_All_Results(self, num):
90          all_results = []
91          current_board = []
92          self.Get_All_Results_Current(current_board, all_results)
93          return all_results
94
95      def Get_All_Results_Current(self, current_board, all_results):
96          next_x = len(current_board)
97          for next_y in range(self.Num_of_Queens):
98              flag = False
99              # 遍历下一行可摆的位置
100             for i in range(next_x):
101                 if next_y == current_board[i] or next_y == current_board[i] + next_x
                        - i or next_y == current_board[i] - next_x + i:
102                     flag = True
103                     break
104
105             # 判断是否与之前的棋子冲突
106             if not flag:
107                 new_board = current_board.copy()
108                 new_board.append(next_y)
109                 if len(new_board) == self.Num_of_Queens:
110                     all_results.append(new_board.copy())
111                     self.current_result = new_board.copy()
112                     self.repaint()
113                 else:
114                     self.Get_All_Results_Current(new_board, all_results)
115
116     def Exit(self):
117         self.close()
118
119     def Draw_Board(self):
```

```
        qp = QPainter()
        qp.begin(self)

        pen = QPen(Qt.black, 2, Qt.SolidLine)
        qp.setPen(pen)
        qp.drawLine(5, 5, 405, 5)
        qp.drawLine(5, 5, 5, 405)
        qp.drawLine(5, 405, 405, 405)
        qp.drawLine(405, 5, 405, 405)

        temp = int(400 / self.Num_of_Queens) # 棋盘格大小

        for i in range(self.Num_of_Queens):
            for j in range(self.Num_of_Queens):
                if (i % 2 == 0 and j % 2 == 0) or (i % 2 == 1 and j % 2 == 1):
                    qp.setBrush(QColor(128, 128, 128)) # 设置棋盘格为灰色
                    qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)
                else:
                    qp.setBrush(QColor(255, 255, 255)) # 设置棋盘格为白色
                    qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)

        qp.end()

    def Draw_Chess(self):
        temp = int(400 / self.Num_of_Queens)
        qp = QPainter()
        qp.begin(self)
        pen = QPen(Qt.black, 2, Qt.SolidLine)
        qp.setPen(pen)
        qp.drawLine(5, 5, 405, 5)
        qp.drawLine(5, 5, 5, 405)
        qp.drawLine(5, 405, 405, 405)
        qp.drawLine(405, 5, 405, 405)

        for i in range(self.Num_of_Queens):
            for j in range(self.Num_of_Queens):
                if (i % 2 == 0 and j % 2 == 0) or (i % 2 == 1 and j % 2 == 1):
                    qp.setBrush(QColor(128, 128, 128))
                    qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)
                else:
                    qp.setBrush(QColor(255, 255, 255))
                    qp.drawRect(int(5 + i * temp), int(5 + j * temp), temp, temp)
```

```
162
163         queen = QPixmap('queen.jpg')
164
165         for i in range(self.Num_of_Queens):
166             x = 5 + temp * i
167             y = 5 + temp * (self.current_result[i])
168             A = int(x + 0.1 * temp)
169             B = int(y + 0.1 * temp)
170             C = int(0.8 * temp)
171             D = int(0.8 * temp)
172             qp.drawPixmap(A, B, C, D, queen)
173
174         qp.end()
175
176     def solution(self):
177         if len(self.results) == 0:
178             QMessageBox.information(self, '提示', '没有搜索出可行解')
179         else:
180             self.Draw_Flag = 1
181
182             for i in range(len(self.results)):
183                 self.current_result = self.results[i]
184                 self.repaint()
185                 time.sleep(0.5)
186
187
188     def closeEvent(self, event):
189         reply = QMessageBox.question(self, '提示',
190                                      "确定退出?", QMessageBox.Yes |
191                                      QMessageBox.No, QMessageBox.No)
192         if reply == QMessageBox.Yes:
193             event.accept()
194         else:
195             event.ignore()
196
197 if __name__ == '__main__':
198     app = QApplication(sys.argv)
199     w = Window()
200     sys.exit(app.exec_())
```