南開大學
Nankai University

人 工 智 能 技 术 实 验
实 验 报 告

实验名称：八皇后问题求解

姓　　名：张恒硕

学　　号：2212266

专　　业：智能科学与技术

目录

人工智能学院

2024 年 9 月

一、 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。

二、 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；
3. 对实验进行图形化界面设计，实现按步或按解的展示。

三、 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；
3. 图形化界面设计。

四、 实验步骤与分析

以下按照代码实现的逻辑逐步分析，与实际代码的顺序可能不同，具体请以 nQueens.py 文件为准。

1. 棋盘初始化

- **内容与原理**

首先要根据输入的数字确定棋盘的大小，绘制黑白相间的棋盘，并标出横纵数字。

- **代码**

```
    # 绘制棋盘与解

def board_solution(self, ax, solution_index):
```

```
~
# 绘制棋盘
ax.clear()                    # 清空上一个解
ax.set_xlim(0, self.size)  # 设置 x、y 轴范围，并使之等比例，关闭坐标轴
ax.set_ylim(0, self.size)
ax.set_aspect('equal')
ax.set_axis_off()
# 填色
colors = ['w', 'k']
for i in range(self.size):
    for j in range(self.size):
        ax.add_patch(plt.Rectangle((j, i), 1, 1, color=colors[(i + j) % 2]))
# 标数
for i in range(self.size):
    ax.text(i + 0.5, self.size + 0.5, str(i + 1), ha='center', va='center')
    ax.text(-0.5, self.size - i - 0.5, str(i + 1), ha='center', va='center', rotation=0)
~
```

- 绘制 n*n 的棋盘格，保证横纵等比例，并只显示 n*n 的部分。
- 黑白相间填色，根据一个格子的横纵坐标之和模 2 的结果，给格子填黑色或白色。
- 为横纵轴标数，要保证标到格子所在行，而非节点所在行。按照棋盘习惯，要在左侧由上到下标行，上侧由左到右标列。

2. 皇后占领域检查

- **内容与原理**

  检查新皇后所在行、列、主副对角线上是否有其他皇后。

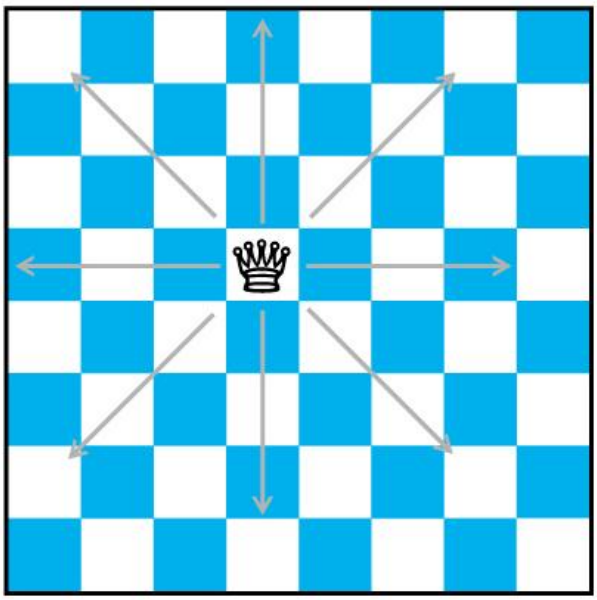

图 4.2 一个皇后的占领域

- **代码**

```
# 判断新皇后是否和已有皇后冲突

def check(self, X, Y):

    return X[0] == Y[0] or X[1] == Y[1] or X[0] - X[1] == Y[0] - Y[1] or X[0] + X[1]

== Y[0] + Y[1]

def valid(self, queens, new_queen):

    for queen in queens:

        if self.check(queen, new_queen):

            return False

    return True
```

  - 检查新皇后和已有的一个皇后的行数、列数、行列差、行列和是否相等。
  - 对每一个已有皇后和新皇后重复占领域检查。

3. 回溯算法

- **内容与原理**

每确定好某行皇后的可行位置后，开始对下一行皇后的位置进行试探，从第一列开始判断该位置是否可以放皇后。如果可以，进行下一个皇后的摆放，全摆完得到一个解法；如果不可以，试探下一个位置，当发现该行没有可以摆放的位置时，回到上一个皇后，将其移往下一个可摆放的位置；如果一直无法摆放，将回溯到最开始的皇后，当此皇后的位置已经在行末，则已找到所有解法。

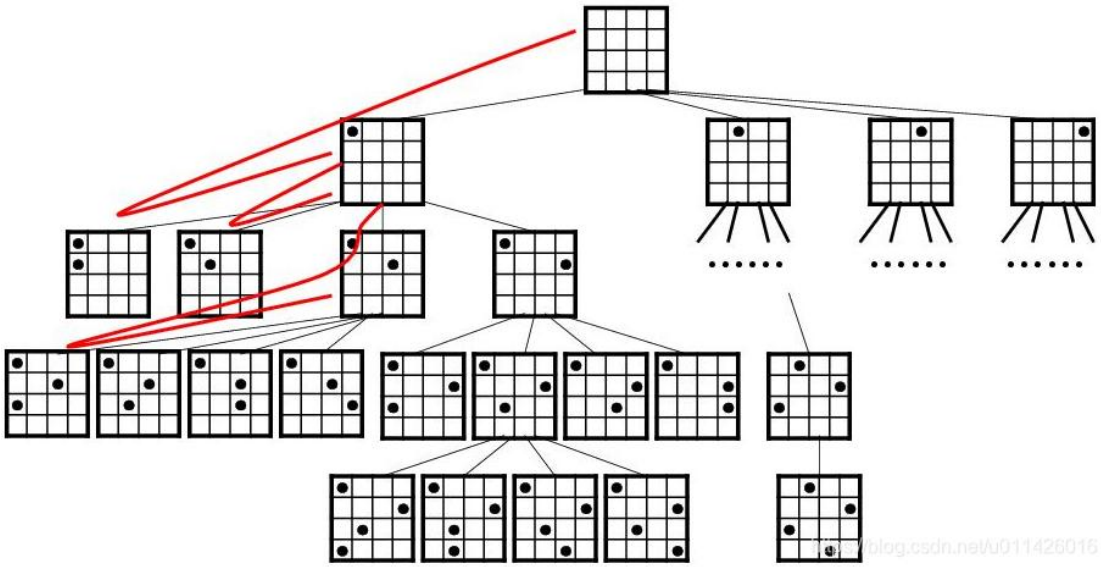

图 4.3 回溯算法思路

- **代码**

```
# 加入与回溯

def n_queens(self, queens=[]):

    row = len(queens)

    # 达到最后一行，保存解

    if row == self.size:

        self.solutions.append(queens.copy())

        return

    for col in range(self.size):
```

```
        new_queen = [row, col]
        if self.valid(queens, new_queen):
            # 新皇后不和已有皇后冲突，添加到列表并向下进行
            queens.append(new_queen)
            if self.n_queens(queens):
                return True
            # 加入的皇后导致无法继续加入新皇后，回溯
            queens.pop()
    return False
```

- 目前已摆放的皇后数是下一个要摆放的皇后的下标，当其达到棋盘尺寸时，说明已摆放所有皇后，收录为一个解。
- 对于每个皇后，从第一列开始试探可摆放的位置，遇到可摆放的位置，则把当前行列纳入已摆放皇后集合中，开始试探下一个皇后。当前皇后无可摆放位置时，从皇后集合中退回上一皇后，并试探后者的下一个可行位置。回溯到皇后集合为空时，结束。

4. 绘制解

- **内容与原理**

在对应的格子中摆放皇后标志。

- **代码**

```
# 绘制棋盘与解
def board_solution(self, ax, solution_index):
    ~
    # 绘制解
    crown_img = mpimg.imread('crown.png')
```

```
    crown_imagebox = OffsetImage(crown_img, zoom=0.1)
    solution = self.solutions[solution_index]
    for queen in solution:
        ab = AnnotationBbox(crown_imagebox, (queen[1] + 0.5, self.size - queen[0] -
0.5), frameon=False)
        ax.add_artist(ab)
```

- 导入处理过的无背景的皇后标志图像。
- 根据解中皇后集合在格子中摆放皇后，为了使第一个皇后出现在最上面的一行，要注意纵坐标。

5. 可视化窗口建立

- **内容与原理**

  设计可视化窗口的各个部件，并将文字汉化。

- **代码**

```
# 主界面
def setupUi(self, MainWindow):
    # 主窗口
    MainWindow.resize(1200, 900)
    # 中央部件
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    MainWindow.setCentralWidget(self.centralwidget)
    # 布局器
    layout = QtWidgets.QVBoxLayout(self.centralwidget)
    # 背景
```

```
    self.background = QtWidgets.QLabel(self.centralwidget)
    image = QPixmap("background.jpg")
    self.background.setPixmap(
        image.scaled(MainWindow.size(), QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation))
    self.background.setSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Expanding)
    layout.addWidget(self.background)
    # 输入框及其文字
    self.cin_data = QtWidgets.QLineEdit(self.centralwidget)
    self.cin_data.setGeometry(QtCore.QRect(950, 100, 150, 80))
    self.cin_data.setStyleSheet("QLineEdit { font-family: Arial; font-size: 18pt;
background-color: rgba(128, 128, 128, 50%); border: 1px solid gray; }")
    self.cin_label = QtWidgets.QLabel(self.centralwidget)
    self.cin_label.setGeometry(QtCore.QRect(900, 20, 250, 100))
    # 开始按钮
    self.start_button = QtWidgets.QPushButton(self.centralwidget)
    self.start_button.setGeometry(QtCore.QRect(925, 200, 200, 100))
    self.start_button.setStyleSheet("QPushButton { background-color: rgba(128, 128,
128, 50%); }")
    self.start_button.clicked.connect(self.run)
    # 暂停继续按钮
```

```
    self.pause_resume_button = QtWidgets.QPushButton(self.centralwidget)
    self.pause_resume_button.setGeometry(QtCore.QRect(925, 300, 200, 100))
    self.pause_resume_button.setStyleSheet("QPushButton { background-color:
rgba(128, 128, 128, 50%); }")
    self.pause_resume_button.clicked.connect(self.pause_resume)
    # 步进按钮
    self.go_button = QtWidgets.QPushButton(self.centralwidget)
    self.go_button.setGeometry(QtCore.QRect(925, 400, 200, 100))
    self.go_button.setStyleSheet("QPushButton { background-color: rgba(128, 128,
128, 50%); }")
    self.go_button.clicked.connect(self.step)
    # 结果显示及其文字
    self.answer_lcd = QtWidgets.QLCDNumber(self.centralwidget)
    self.answer_lcd.setGeometry(QtCore.QRect(900, 550, 250, 100))
    self.answer_label = QtWidgets.QLabel(self.centralwidget)
    self.answer_label.setGeometry(QtCore.QRect(900, 480, 250, 100))
    # 运行时间显示
    self.time_label1 = QtWidgets.QLabel(self.centralwidget)
    self.time_label1.setGeometry(QtCore.QRect(900, 650, 250, 100))
    self.time_label2 = QtWidgets.QLabel(self.centralwidget)
    self.time_label2.setGeometry(QtCore.QRect(900, 700, 250, 100))
    # 动图展示及动态下标
```

```
    self.figure = Figure(figsize=(5, 5), dpi=200)
    self.canvas = FigureCanvas(self.figure)
    self.canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
    self.canvas.setParent(self.centralwidget)
    self.canvas.setGeometry(QtCore.QRect(50, 50, 800, 800))
    self.dynamic_label = QtWidgets.QLabel(self.centralwidget)
    self.dynamic_label.setGeometry(QtCore.QRect(100, 750, 300, 100))
    # 校徽
    self.badge = QtWidgets.QLabel(self.centralwidget)
    self.badge.setGeometry(QtCore.QRect(1050, 750, 150, 150))
    image = QPixmap('校徽.png').scaled(self.badge.size(), Qt.KeepAspectRatio,
Qt.SmoothTransformation)
    self.badge.setPixmap(image)
    # 提示
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setGeometry(QtCore.QRect(50, 830, 1000, 100))
    # 文字
    self.Chinese(MainWindow)
# 文本
def Chinese(self, MainWindow):
    self.font = QFont()
    self.font.setFamily('宋体')
```

```
    self.font.setPointSize(18)
    translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(translate("1", "n 皇后问题"))
    self.cin_label.setText(translate("1", "请输入皇后数："))
    self.cin_label.setFont(self.font)
    self.start_button.setText(translate("1", "开始"))
    self.start_button.setFont(self.font)
    self.pause_resume_button.setText(translate("1", "暂停"))
    self.pause_resume_button.setFont(self.font)
    self.go_button.setText(translate("1", "步进"))
    self.go_button.setFont(self.font)
    self.answer_label.setText(translate("1", "解数为："))
    self.answer_label.setFont(self.font)
    self.time_label1.setText(translate("1", "求解耗时为为："))
    self.time_label1.setFont(self.font)
    self.time_label2.setText(translate("1", "0.0s"))
    self.time_label2.setFont(self.font)
    self.label.setText(translate("1", "请先输入皇后数，在按开始按钮后再点击暂停和步进按钮！
"))
    self.label.setFont(self.font)
```

- 指定窗口尺寸，设定中央部件和布局器。
- 添加背景图，自适应窗口大小。

- 设计一些列部件，进行位置和大小的确定，并将文本部分的框半透明化。
- 为文本部分添加汉语，setText()函数本身无法以汉字作为参数，需要专门的函数。设计文本的字体和大小。
- 添加校徽作为装饰物，导入的校徽图像是经过处理的无背景图像。

6. 输入

- **内容与原理**

输入皇后数量，判断输入合法性。

图 4.6-1 输入

- **代码**

```
# 链接 1：start

def run(self):

    try:

        n = int(self.cin_data.text())

        if n <= 0:

            raise ValueError("请输入一个正数！")

    except ValueError as e:
```

```
        QtWidgets.QMessageBox.warning(None, "错误输入", f"不接受{e}作为参数")
        return
~
```

- 输入一个数字，按开始按钮，如果不是正数，会弹出警告窗提示输入正数。

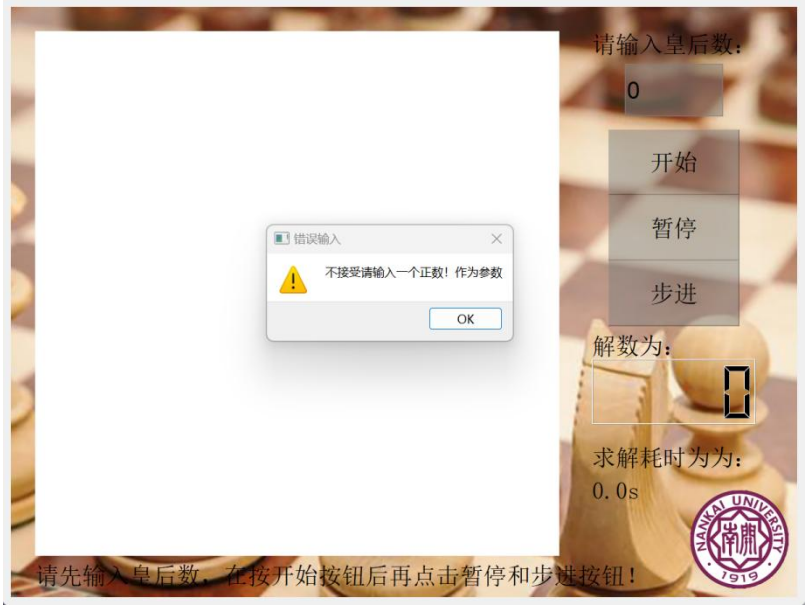

图 4.6-2 输入非正数

- 其他错误（小数、非数），会弹出警告窗提示不接受这样的参数。

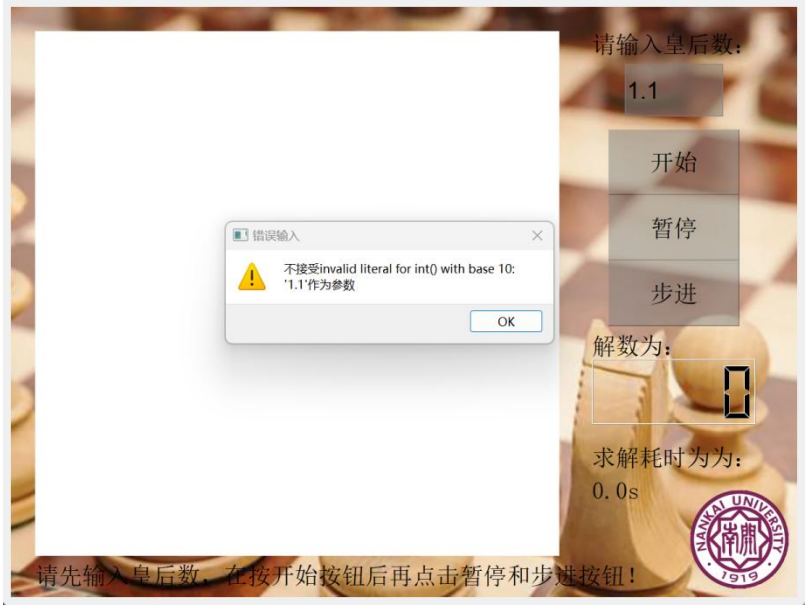

图 4.6-3 输入其他非法参数

7. 开始按钮与动图显示

- **内容与原理**

点击开始按钮后，开始整个代码的运行，输出动图和结果（后续部分说明）。

图 4.7 开始按钮与动图显示

- **代码**

```
# 更新动图部分

def update_plot(self):

    if self.current_solution_index < len(self.solver.solutions):

        self.solver.board_solution(self.ax, self.current_solution_index)

        self.canvas.draw()

        self.current_solution_index += 1

        self.dynamic_label.setText(f"当前为第{self.current_solution_index}个解")

        self.dynamic_label.setFont(self.font)

    else:

        self.timer.stop()
```

```
# 链接 1：start
def run(self):
    ~
    # 清空（为多次运行服务）
    self.figure.clf()
    self.canvas.draw()
    # 实例化问题
    self.start = 1
    ~
    # 动图
    self.ax = self.figure.add_subplot(111)
    self.current_solution_index = 0
    self.timer = QtCore.QTimer()
    self.timer.timeout.connect(lambda: self.update_plot())
    self.timer.start(100)  # 每个解停留 0.1s
```

- 点击开始按钮后，首先清空动图显示框，并初始化一个新的，这是防止在多次运行“开始”按钮功能的过程中，遗留上一次代码运行的痕迹（主要是横纵轴的标数）。
- 开始 QT 动画的计时器，运行槽函数：当已展示解数小于总解数时，绘制解，并在左下角标注次序。函数会一直运行到画完所有解，两个解之间间隔 0.1s。

8. 暂停、继续按钮

- **内容与原理**

通过控制计时器，使槽函数可以停止和重启。

图 4.8-1 暂停、继续按钮

- 代码

```
# 链接 2：pause_resume

def pause_resume(self):

    if self.start == 0:

        QtWidgets.QMessageBox.warning(None, "错误操作", "请不要在未运行时暂停！")

        return

    if self.timer.isActive():

        self.timer.stop()

        self.pause_resume_button.setText("继续")

    else:

        self.timer.start(100)

        self.pause_resume_button.setText("暂停")
```

  - 开始时，按钮显示暂停，此时（未点击开始时）按下暂停会弹出警告窗提示未在运行时不可暂停。

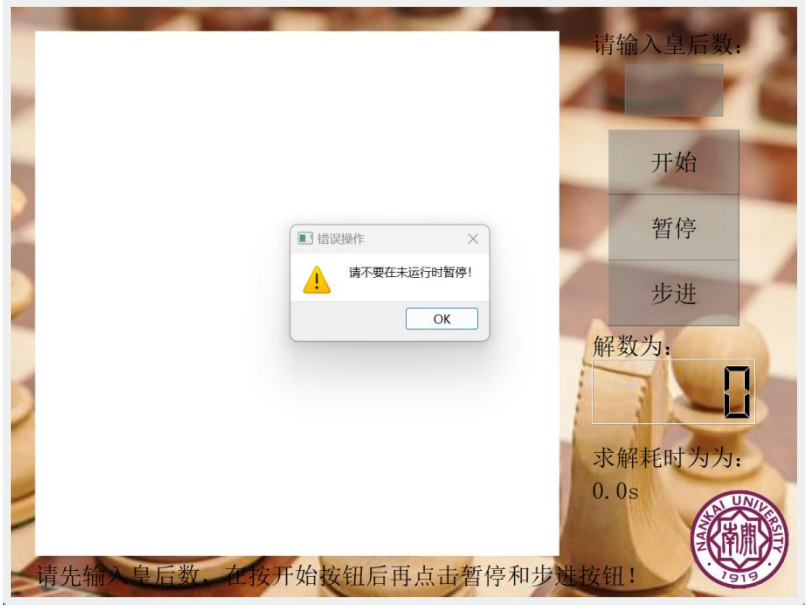

图 4.8-2 未开始时暂停

- 在开始后，按暂停按钮，会停止动图更新，图标变更为继续，此时再按可在原位置继续开始运行。

9. 步进按钮

- **内容与原理**

在开始后暂停，可以步进每一步。

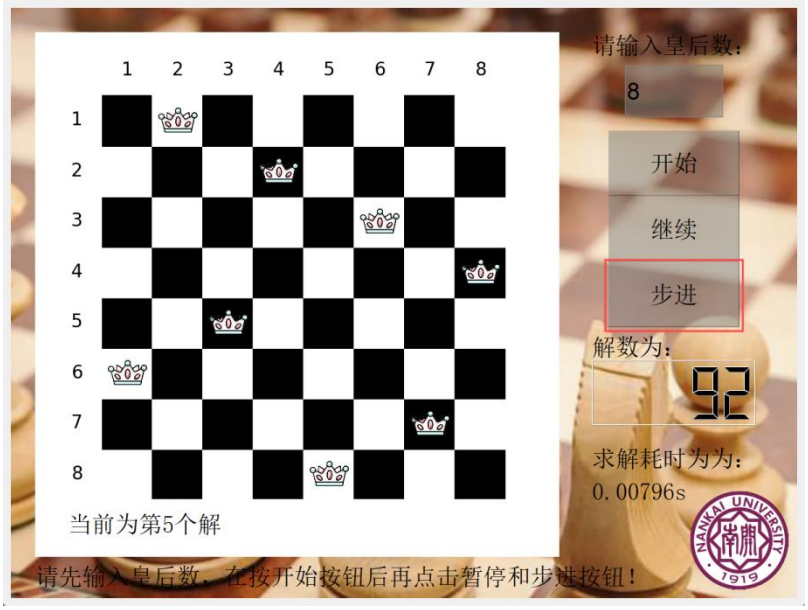

图 4.9-1 步进按钮

- **代码**

```
# 链接 3：go

def step(self):

    if self.start == 0:

        QtWidgets.QMessageBox.warning(None, "错误操作", "请不要在未运行时步进！")

        return

    if self.timer.isActive():

        self.timer.stop()

        self.pause_resume_button.setText("继续")

        QtWidgets.QMessageBox.warning(None, "错误操作", "请不要在运行非暂停时步进！")

    else:

        if self.current_solution_index == len(self.solver.solutions):

            QtWidgets.QMessageBox.warning(None, "错误操作", "已经是最后一种解了，无法
继续步进！")

            return

        self.update_plot()
```

- 未点击开始时，按下步进会弹出警告窗提示未在运行时不可步进。这是本程序未实现的功能，已在下面标注文字进行提示，为保证稳定性，会弹窗警告。

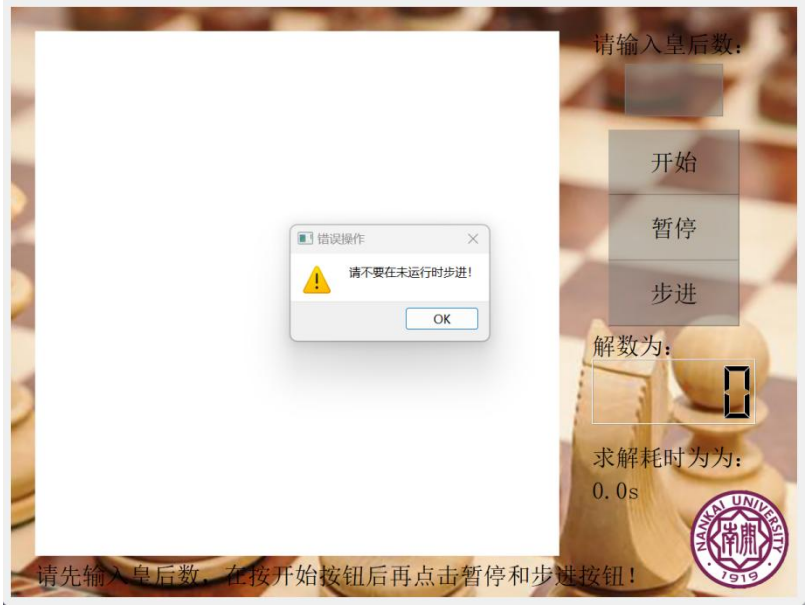

图 4.9-2 未开始时步进

- 在开始后，未点击暂停时，按下步进会弹出警告窗提示在运行时不可步进。

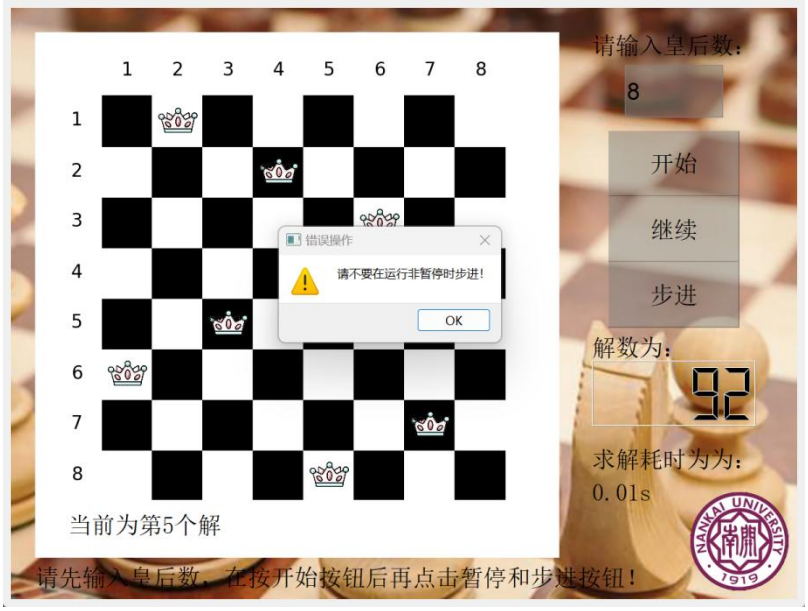

图 4.9-3 开始后未暂停时步进

- 在暂停后，步进可以逐个解进行观察。但在遍历完一个问题的所有解后，继续步进会弹出警告窗提示无法继续步进。

图 4.9-4 遍历解后无法继续步进

10. 结果与运行时间

- **内容与原理**

按动开始按钮，在窗口的右下角会优先显示该问题的解数和求解时长。

图 4.10 结果与运行时间

- **代码**

```
# 实例化问题

self.start = 1

start_time = time.time()
```

```
self.solver = n_Queens(n)

self.solver.n_queens()

end_time = time.time()

# 结果、运行时间展示

self.answer_lcd.display(len(self.solver.solutions))

self.time_label2.setText(f"{round(end_time - start_time, 5)}s")

self.time_label2.setFont(self.font)
```

- 在开始后，会先求解问题，此时不会更新动图，在求解完后，会显示解数和代码运行时长。后者由于位数限制，皇后数量过小或过大都无法完全显示。

11. 主函数

- **内容与原理**

  代码的主函数。

- **代码**

```
app = QApplication(sys.argv)

MainWindow = QMainWindow()

ui = Ui_MainWindow()

ui.setupUi(MainWindow)

MainWindow.show()

sys.exit(app.exec())
```

- 创建窗口，展示窗口，退出窗口。

五、 实验结果

1. 皇后问题求解

表 1 n 皇后问题

皇后数	解数	运行时间（单位为 s，保留五位小数，过小的显示不全）
1	1	0.0
2	0	0.0
3	0	0.0
4	2	0.0
5	10	0.0
6	4	0.00099
7	40	0.00193
8	92	0.009
9	352	0.044
10	724	0.2271
11	2680	1.33092
12	14200	7.7239
由于显示位置有限，不再继续增加皇后，仅展示以上 12 种情况。		

由上表可知，在皇后数增长的过程中，解数和代码运行时间都基本成几何增长，下面两图也反应了这一现象。

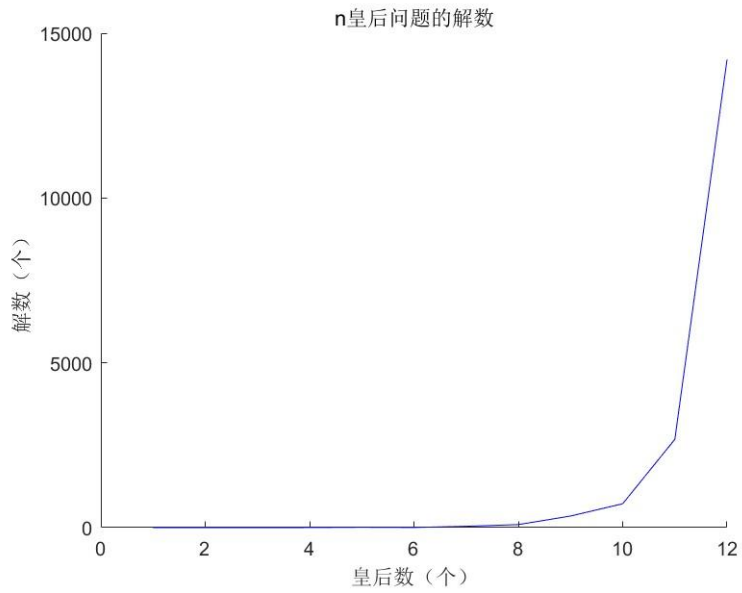

图 5.1-1 解数折线图

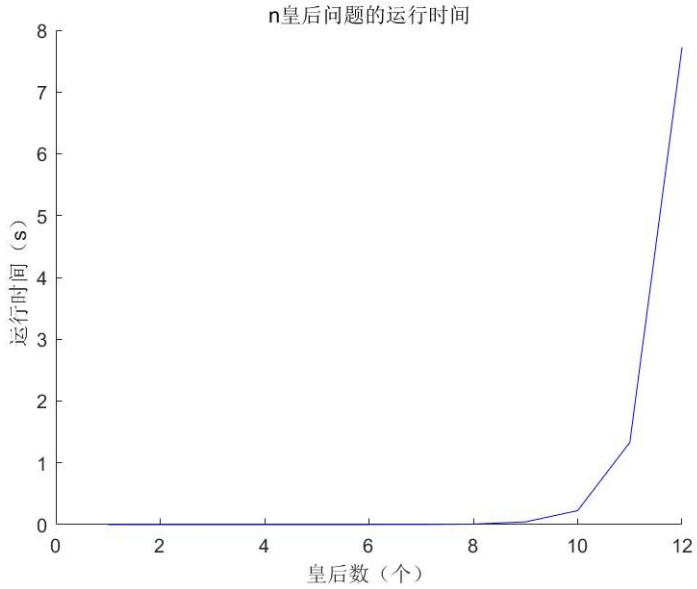

图 5.1-2 运行时间折线图

2. 可视化展示

由于在前面已经展示了程序运行的多种情况的截图，这里仅展示 12 皇后问题的一个截图。

图 5.2 12 皇后问题第 30 个解

六、 分析总结

1. 成果总结

本次实验成功实现了 n 皇后问题的求解和相应的可视化程序，并且考虑了很多特殊情况，程序的稳定性较高。

2. 问题分析与改进思路

- 问题解决思路：在皇后问题的迭代回溯过程中，可以采用其他的判断皇后占领域的方法：比如加入一个新皇后，就在棋盘中标记她的占领域，之后的皇后根据这个棋盘标记矩阵进行安置，这样可以减少试探的次数，但可能需要动态地更新这个标记矩阵。
- 步进问题：步进按钮无法在为按开始前使用，也未与输入的数字关联，这些问题可能需要更改代码逻辑来改进。