題目：八皇後問題解法

Title: The solution of eight queen problem

|  |  |
| --- | --- |
| 學 號  Student No. | ： 譚炳浩 |
| 學 院  Faculty | ： 2240018506 |
| 課 程  Program | ： 人工智能基礎 |
| 專 業  Major | ： 智能技術 |
| 指導老師  Supervisor | ： 高開周 |
| 日 期  Date | ： 2024年9月17日 |

1. Introduction

The 8 Queens Problem is a classic combinatorial optimization challenge that involves placing 8 queens on an 8×8 chessboard so that no two queens threaten each other. This means no two queens can share the same row, column, or diagonal. The queen, being the most powerful chess piece, can move in any direction, making this a fundamental problem in algorithm design and artificial intelligence.

1. Algorithm

2.1 Backtracking Algorithm

One common method for solving the 8 Queens Problem is the backtracking algorithm. Backtracking is a trial-and-error method that explores all possible options to find a solution. The specific steps are as follows:

1. Start from the first row and place the first queen in one of the columns.

2. Recursively place the next queen in the next row, ensuring that it does not conflict with previously placed queens.

3. If the current placement is not valid, backtrack to the previous queen's position and try the next column.

4. Repeat this process until all 8 queens are successfully placed.

2.2 State Checking

To ensure that queens do not threaten each other, we need to maintain three sets:

1. Column Set: Records whether a column already has a queen placed.

2. Main Diagonal Set: Records the status of the main diagonals (from top-left to bottom-right).

3. Secondary Diagonal Set: Records the status of the secondary diagonals (from top-right to bottom-left).

1. Code

import matplotlib.pyplot as plt  
import numpy as np  
import os  
  
 # 检查该列是否有皇后  
 # 遍历当前行之前的所有行（for i in range(row)）。  
 # 检查是否有皇后在同一列（board[i] == col）。  
 # 检查是否有皇后在同一斜线（abs(board[i] - col) == abs(i - row)）。  
 # 如果发现冲突，返回 False，否则返回 True。  
def is\_safe(board, row, col):  
 for i in range(row):   
 if board[i] == col or abs(board[i] - col) == abs(i - row):  
 return False  
 return True  
  
  
 # 该函数使用递归的方法解决 N 皇后问题。  
 # 输入: board（棋盘状态）、row（当前行）、solutions（存储所有解决方案的列表）。  
 # 当 row 等于棋盘的长度时，表示所有皇后都已成功放置，此时将当前棋盘的状态添加到 solutions 列表中。  
def solve\_n\_queens\_util(board, row, solutions):  
 if row == len(board):  
 solutions.append(board.copy())  
 return   
 # 遍历当前行的每一列（for col in range(len(board))）。  
 # 使用 is\_safe 函数检查在 (row, col) 位置放置皇后是否安全。  
 # 如果安全，将当前列索引放置到 board 的当前行（board[row] = col）。  
 # 递归调用 solve\_n\_queens\_util 尝试在下一行放置皇后（row + 1）。  
 # 回溯：如果放置皇后后没有找到有效的解决方案，将当前位置重置为 -1 （board[row] = -1），以便尝试其他列。  
 for col in range(len(board)):  
 if is\_safe(board, row, col):  
 board[row] = col # 放置皇后  
 solve\_n\_queens\_util(board, row + 1, solutions) # 递归放置下一个皇后  
 board[row] = -1 # 回溯，移除皇后  
  
  
  
def solve\_n\_queens(n):  
 board = [-1] \* n # 初始化棋盘，每行无皇后  
 solutions = []  
 solve\_n\_queens\_util(board, 0, solutions)  
 return solutions  
  
  
  
def draw\_solution(solution, index):  
 n = len(solution)  
 board = np.zeros((n, n))  
 for row in range(n):  
 board[row][solution[row]] = 1 # 将皇后位置标记为 1  
  
 plt.imshow(board, cmap='binary')  
 plt.xticks(range(n))  
 plt.yticks(range(n))  
 plt.gca().invert\_yaxis()  
 plt.grid(False)  
   
 # 保存图像到本地  
 filename = f"solution\_{index + 1}.png"  
 plt.savefig(filename)  
 plt.close() # 关闭当前图像，以释放内存  
 print(f"Saved: {filename}")  
  
# 解决8皇后问题并绘制所有解  
solutions = solve\_n\_queens(8)  
print(f"Total solutions: {len(solutions)}") # 打印解决方案的数量  
for index, solution in enumerate(solutions):  
 draw\_solution(solution, index)

The code is based on the matplotlib and numpy libraries.

1. Result

There are 92 distinct solutions to the 8 Queens Problem. Each solution is represented as a list, indicating the column position of the queen in each row.

文本

低可信度描述已自动生成

Fig. 1. Code running process

QR 代码

描述已自动生成

Fig. 2. The one restult of the code. the solution [0,6,4,7,1,3,5,2] shows the column placements of the queens