软工作业2实现

林绮彤23336139

分析报告：

分类	任务名称	起止时间	详情	难点	改进想法
分析	理解开发任务要求	0:18-1:24	让ai画出任务指导蓝图，让我能够更加清晰地理解要如何实现这个项目	ai指导的内容不一定完全符合作业任务要求，需要自行辨别并修改完善	采用任务文档中提到的几个分类思路--分析、编码、验证、调试来进行开发
分析	书写需求文档（借助deepseek）	1:24-3:28	让ai写出具体的需求文档，较为详尽	1. 形式化描述策略逻辑 2. 平衡文档详略程度 3. 制定可量化的验收指标	无
分析	策略定义确认与学习	3:28-4:20	理解任务需求，明确两个数独策略的定义和要求：LRC策略和Possible Number策略	需要仔细查阅资料并理解，这两个策略都不是很复杂，但是需要考虑如何高效表示和遍历数独网络的行、列和宫格；同时思考两种策略的优先级和调用顺序；确保策略实现不会破坏数独的完整性	1. 添加策略应用示例 2. 设计策略组合测试用例
编码	设计数据结构	4:20-7:24	提前思考如何存储数独的行、列和值，进而方便后续使用	需要考虑到对应的数据结构是否能够与LRC策略以及Possible Number策略适应；数据一致性维护	1. 采用位运算优化 2. 实现增量更新机制
编码	实现LRC策略	7:24-10:40	运用学习到的LRC策略正确编写LRC算法	确保边界条件正确处理，实现高效的宫格遍历，处理策略间的交互和重复应用--考虑如何降低复杂度	未来可以考虑添加单元测试验证单个策略的正确性，或者设计自动测试；

分类	任务名称	起止时间	详情	难点	改进想法
编码	实现possible number策略	10:40-13:45	运用学习到的Possible Number策略，正确编写Possible Number算法	类似的也要正确处理边界条件，实现高效的宫格遍历，高效对数据查找和存储	
编码	实现main函数，方便测试	13:45-15:33	考虑运用两个策略的顺序，实现简单的测试函数	考虑策略的优先顺序是很重要的	
编码	根据调试结果修改代码	15:33-19:15	修改完善代码	可能找不到错误，需要调试或者借助ai的力量	借助ai，让ai列举出错误点
验证	设计样例测试程序	15:49-19:16			
验证	设计新样例测试新程序（调试和修改之后）	20:34-21:34	设计多层次的新测试样例，从简单到复杂依次测试，更有可靠性	如何设计样例，样例是否	
调试	尝试运行，发现出问题进行调试	15:33-19:15	运行出问题，查找语法错误后修改完善代码	人工查找稍有困难，尤其是对于大任务而已，借助ai来查找	
调试	单步调试	21:34-22:30	单步调试		

开发任务详细说明

一、 代码架构分析

1. 类结构设计

JAVA

```
classDiagram
    class SudokuStrat {
        +LRC(int[][]) List<Integer>[][]
        +possibleNumbers(int[][]) List<Integer>[][]
        +print(List<Integer>[][]) void
    }

    class SudokuVisualTest {
        +main(String[]) void
        -testVisual(int[][], String) void
        -printGrid(int[][]) void
        -printSolvedGrid(int[][], List<Integer>[][]) void
        -printAllCandidates(int[][], List<Integer>[][]) void
    }
```

2. 详细代码

LRC策略（Last remaining cell）

JAVA

```
public static List<Integer>[][] LRC(int[][] grid) {
        List<Integer>[][] cands = possibleNumbers(grid);
        for (int n = 1; n <= 9; n++) {
            // Check rows
            for (int r = 0; r < 9; r++) {
                List<int[]> cells = new ArrayList<>();
                for (int c = 0; c < 9; c++) {
                    if (grid[r][c] == 0 && cands[r]
[c].contains(n)) {
                        cells.add(new int[]{r, c});
                    }
                }
                // If only one cell in this row can contain this
number
                if (cells.size() == 1) {
                    int row = cells.get(0)[0];
                    int col = cells.get(0)[1];
                    cands[row][col] = new ArrayList<>
(Collections.singletonList(n));
                }
            }
            // Check columns
            for (int c = 0; c < 9; c++) {
                List<int[]> cells = new ArrayList<>();
                for (int r = 0; r < 9; r++) {
                    if (grid[r][c] == 0 && cands[r]
[c].contains(n)) {
                        cells.add(new int[]{r, c});
                    }
                }
                if (cells.size() == 1) {
                    int row = cells.get(0)[0];
                    int col = cells.get(0)[1];
                    cands[row][col] = new ArrayList<>
(Collections.singletonList(n));
                }
```

```
                }
                // Check blocks
                for (int br = 0; br < 3; br++) {
                    for (int bc = 0; bc < 3; bc++) {
                        List<int[]> cells = new ArrayList<>();
                        for (int r = 0; r < 3; r++) {
                            for (int c = 0; c < 3; c++) {
                                int row = br * 3 + r;
                                int col = bc * 3 + c;
                                if (grid[row][col] == 0 && cands[row]
[col].contains(n)) {
                                    cells.add(new int[]{row, col});
                                }
                            }
                        }
                        if (cells.size() == 1) {
                            int row = cells.get(0)[0];
                            int col = cells.get(0)[1];
                            cands[row][col] = new ArrayList<>
(Collections
                            .singletonList(n));
                        }
                    }
                }
            }
            return cands;
        }
```

possible Number策略:

JAVA

```
public static List<Integer>[][] possibleNumbers(int[][] grid) {
        List<Integer>[][] cands = new List[9][9];
        for (int r = 0; r < 9; r++) {
            for (int c = 0; c < 9; c++) {
                if (grid[r][c] != 0) {
                    cands[r][c] = new ArrayList<>
(Collections.singletonList(grid[r][c]));
                } else {
                    cands[r][c] = new ArrayList<>();
                    for (int n = 1; n <= 9; n++) {
                        cands[r][c].add(n);
                    }
                    for (int i = 0; i < 9; i++) {
                        if (grid[r][i] != 0) {
                            cands[r]
[c].remove(Integer.valueOf(grid[r][i]));
                        }
                    }
                    for (int i = 0; i < 9; i++) {
                        if (grid[i][c] != 0) {
                            cands[r]
[c].remove(Integer.valueOf(grid[i][c]));
                        }
                    }
                    int br = (r / 3) * 3;
                    int bc = (c / 3) * 3;
                    for (int i = br; i < br + 3; i++) {
                        for (int j = bc; j < bc + 3; j++) {
                            if (grid[i][j] != 0) {
                                cands[r]
[c].remove(Integer.valueOf(grid[i][j]));
                            }
                        }
                    }
                }
            }
        }
```

```
        return cands;
    }
```

main函数部分--最简单的测试同时测试能否运行

JAVA

```java
public static void main(String[] args) {
    // 例子:
    int[][] grid = {
        {5, 3, 0, 0, 7, 0, 0, 0, 0},
        {6, 0, 0, 1, 9, 5, 0, 0, 0},
        {0, 9, 8, 0, 0, 0, 0, 6, 0},
        {8, 0, 0, 0, 6, 0, 0, 0, 3},
        {4, 0, 0, 8, 0, 3, 0, 0, 1},
        {7, 0, 0, 0, 2, 0, 0, 0, 6},
        {0, 6, 0, 0, 0, 0, 2, 8, 0},
        {0, 0, 0, 4, 1, 9, 0, 0, 5},
        {0, 0, 0, 0, 8, 0, 0, 7, 9}
    };
    System.out.println("Possible Number Inference:");
    List<Integer>[][] cands = possibleNumbers(grid);
    print(cands);
    System.out.println("After LRC:");
    List<Integer>[][] cands2 = LRC(grid);
    print(cands2);
```

```
    }


    public static void print(List<Integer>[][] cands) {
        for (int r = 0; r < 9; r++) {
            for (int c = 0; c < 9; c++) {
                System.out.print("Cell[" + r + "][" + c + "]: ");
                System.out.println(cands[r][c]);
            }
        }
    }
```

整体代码：

JAVA

```java
import java.util.*;


public class SudokuStrat {

    public static List<Integer>[][] LRC(int[][] grid) {

        List<Integer>[][] cands = possibleNumbers(grid);

        for (int n = 1; n <= 9; n++) {

            // Check rows

            for (int r = 0; r < 9; r++) {

                List<int[]> cells = new ArrayList<>();

                for (int c = 0; c < 9; c++) {

                    if (grid[r][c] == 0 && cands[r]
[c].contains(n)) {

                        cells.add(new int[]{r, c});

                    }

                }

                // If only one cell in this row can contain this
number

                if (cells.size() == 1) {

                    int row = cells.get(0)[0];

                    int col = cells.get(0)[1];

                    cands[row][col] = new ArrayList<>
```

```
(Collections.singletonList(n));
                }
            }

            // Check columns
            for (int c = 0; c < 9; c++) {
                List<int[]> cells = new ArrayList<>();
                for (int r = 0; r < 9; r++) {
                    if (grid[r][c] == 0 && cands[r]
[c].contains(n)) {
                        cells.add(new int[]{r, c});
                    }
                }
                if (cells.size() == 1) {
                    int row = cells.get(0)[0];
                    int col = cells.get(0)[1];
                    cands[row][col] = new ArrayList<>
(Collections.singletonList(n));
                }
            }
```

```
            // Check blocks
            for (int br = 0; br < 3; br++) {
                for (int bc = 0; bc < 3; bc++) {
                    List<int[]> cells = new ArrayList<>();
                    for (int r = 0; r < 3; r++) {
                        for (int c = 0; c < 3; c++) {
                            int row = br * 3 + r;
                            int col = bc * 3 + c;
                            if (grid[row][col] == 0 && cands[row]
[col].contains(n)) {
                                cells.add(new int[]{row, col});
                            }
                        }
                    }
                    if (cells.size() == 1) {
                        int row = cells.get(0)[0];
                        int col = cells.get(0)[1];
                        cands[row][col] = new ArrayList<>
(Collections.singletonList(n));
                    }
                }
```

```
            }

        }

        return cands;

    }


    public static List<Integer>[][] possibleNumbers(int[][] grid)
{

        List<Integer>[][] cands = new List[9][9];

        for (int r = 0; r < 9; r++) {

            for (int c = 0; c < 9; c++) {

                if (grid[r][c] != 0) {

                    cands[r][c] = new ArrayList<>
(Collections.singletonList(grid[r][c]));

                } else {

                    cands[r][c] = new ArrayList<>();

                    for (int n = 1; n <= 9; n++) {

                        cands[r][c].add(n);

                    }



                    for (int i = 0; i < 9; i++) {

                        if (grid[r][i] != 0) {
```

```
                            cands[r]
[c].remove(Integer.valueOf(grid[r][i]));

                        }

                    }

                    for (int i = 0; i < 9; i++) {

                        if (grid[i][c] != 0) {

                            cands[r]
[c].remove(Integer.valueOf(grid[i][c]));

                        }

                    }

                    int br = (r / 3) * 3;

                    int bc = (c / 3) * 3;

                    for (int i = br; i < br + 3; i++) {

                        for (int j = bc; j < bc + 3; j++) {

                            if (grid[i][j] != 0) {

                                cands[r]
[c].remove(Integer.valueOf(grid[i][j]));

                            }

                        }

                    }

                }

            }
```

```
        }

        return cands;
    }


    public static void main(String[] args) {
        // 例子:
        int[][] grid = {
            {5, 3, 0, 0, 7, 0, 0, 0, 0},
            {6, 0, 0, 1, 9, 5, 0, 0, 0},
            {0, 9, 8, 0, 0, 0, 0, 6, 0},
            {8, 0, 0, 0, 6, 0, 0, 0, 3},
            {4, 0, 0, 8, 0, 3, 0, 0, 1},
            {7, 0, 0, 0, 2, 0, 0, 0, 6},
            {0, 6, 0, 0, 0, 0, 2, 8, 0},
            {0, 0, 0, 4, 1, 9, 0, 0, 5},
            {0, 0, 0, 0, 8, 0, 0, 7, 9}
        };
        System.out.println("Possible Number Inference:");
        List<Integer>[][] cands = possibleNumbers(grid);
        print(cands);
```

```
        System.out.println("After LRC:");
        List<Integer>[][] cands2 = LRC(grid);
        print(cands2);
    }

    public static void print(List<Integer>[][] cands) {
        for (int r = 0; r < 9; r++) {
            for (int c = 0; c < 9; c++) {
                System.out.print("Cell[" + r + "][" + c + "]: ");
                System.out.println(cands[r][c]);
            }
        }
    }

}
```

测试代码--更加详细

JAVA

```java
import java.util.*;

public class SudokuVisualTest {

    public static void main(String[] args) {

        // 测试各种数独谜题

        testVisual(createEasyPuzzle(), "Easy Puzzle");

        testVisual(createMediumPuzzle(), "Medium Puzzle");

        testVisual(createHardPuzzle(), "Hard Puzzle");

        testVisual(createExtremelyHardPuzzle(), "Extremely Hard
Puzzle");

    }

    private static void testVisual(int[][] grid, String testName)
{

        System.out.println("===== " + testName + " =====");

        // 打印原始网格

        System.out.println("Original Grid:");

        printGrid(grid);

        // 从Possible Number获取候选值

        List<Integer>[][] possibleCands =
SudokuStrat.possibleNumbers(grid);

        // 从Last Remaining Cell获取候选值
```

```
        List<Integer>[][] lastCands = SudokuStrat.LRC(grid);

        // 统计每种方法解决的单元格数量

        int possibleSolved = countSolved(grid, possibleCands);

        int lastSolved = countSolved(grid, lastCands);

        System.out.println("\nPossible Number solved " +
possibleSolved + " cells.");

        System.out.println("Last Remaining Cell solved " +
lastSolved + " additional cells.");

        // 打印显示已解决单元格的网格

        System.out.println("\nAfter applying strategies (solved
cells marked):");

        printSolvedGrid(grid, lastCands);

        // 打印所有未解决单元格及其候选值

        System.out.println("\nComplete list of cell candidates:");

        printAllCandidates(grid, lastCands);

        System.out.println("\n");

    }

    private static void printGrid(int[][] grid) {

        System.out.println("┌─────────┬─────────┬─────────┐");

        for (int i = 0; i < 9; i++) {

            System.out.print("│ ");

            for (int j = 0; j < 9; j++) {
```

```
                System.out.print(grid[i][j] == 0 ? "." : grid[i]
[j]);

                if (j % 3 == 2) System.out.print(" │ ");

                else System.out.print(" ");

            }

            System.out.println();

            if (i % 3 == 2 && i < 8)

                System.out.println("├─────────┼─────────┼─────────┤");

        }

        System.out.println("└─────────┴─────────┴─────────┘");

    }

    private static void printSolvedGrid(int[][] grid,
List<Integer>[][] cands) {

        System.out.println("┌─────────┬─────────┬─────────┐");

        for (int i = 0; i < 9; i++) {

            System.out.print("│ ");

            for (int j = 0; j < 9; j++) {

                if (grid[i][j] != 0) {

                    System.out.print(grid[i][j] + " ");

                } else if (cands[i][j].size() == 1) {

                    // 使用固定宽度显示已解决的单元格
```

```
                    System.out.print("*" + cands[i][j].get(0) +
"");
                } else {
                    System.out.print(". ");
                }
                if (j % 3 == 2) System.out.print("| ");
            }
            System.out.println();
            if (i % 3 == 2 && i < 8)
                System.out.println("├─────────┼─────────┼─────────┤");
        }
        System.out.println("└─────────┴─────────┴─────────┘");
        System.out.println("注: *表示策略推断出的数字");
    }
    // 新方法：打印所有未解决单元格的候选值
    private static void printAllCandidates(int[][] grid,
List<Integer>[][] cands) {
        int count = 0;
        // 按行列顺序打印每个未解决单元格的候选值
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
```

```
                // 只打印未填充且有多个候选值的单元格
                if (grid[i][j] == 0 && cands[i][j].size() > 1) {
                    System.out.println("Cell[" + i + "][" + j +
"]: " + cands[i][j]);
                    count++;
                }
            }
        }
        if (count == 0) {
            System.out.println("没有未解决的单元格或所有单元格都已确定
唯一值。");
        } else {
            System.out.println("总共有 " + count + " 个未解决的单元
格。");
        }
    }
    // 此方法保留用于打印样本（前5个）候选值，如果需要的话
    private static void printSampleCandidates(int[][] grid,
List<Integer>[][] cands) {
        // 打印一些有趣单元格的候选值
        int count = 0;
        for (int i = 0; i < 9 && count < 5; i++) {
```

```
            for (int j = 0; j < 9 && count < 5; j++) {
                if (grid[i][j] == 0 && cands[i][j].size() > 1) {
                    System.out.println("Cell[" + i + "][" + j +
"]: " + cands[i][j]);
                    count++;
                }
            }
        }
    }

    private static int countSolved(int[][] grid, List<Integer>[][]
cands) {
        int solved = 0;
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if (grid[i][j] == 0 && cands[i][j].size() == 1) {
                    solved++;
                }
            }
        }
        return solved;
    }
```

```
    // 测试案例生成器
    private static int[][] createEasyPuzzle() {
        return new int[][] {
            {5, 3, 0, 0, 7, 0, 0, 0, 0},
            {6, 0, 0, 1, 9, 5, 0, 0, 0},
            {0, 9, 8, 0, 0, 0, 0, 6, 0},
            {8, 0, 0, 0, 6, 0, 0, 0, 3},
            {4, 0, 0, 8, 0, 3, 0, 0, 1},
            {7, 0, 0, 0, 2, 0, 0, 0, 6},
            {0, 6, 0, 0, 0, 0, 2, 8, 0},
            {0, 0, 0, 4, 1, 9, 0, 0, 5},
            {0, 0, 0, 0, 8, 0, 0, 7, 9}
        };
    }
    private static int[][] createMediumPuzzle() {
        return new int[][] {
            {0, 2, 0, 6, 0, 8, 0, 0, 0},
            {5, 8, 0, 0, 0, 9, 7, 0, 0},
            {0, 0, 0, 0, 4, 0, 0, 0, 0},
            {3, 7, 0, 0, 0, 0, 5, 0, 0},
```

```
            {6, 0, 0, 0, 0, 0, 0, 0, 4},
            {0, 0, 8, 0, 0, 0, 0, 1, 3},
            {0, 0, 0, 0, 2, 0, 0, 0, 0},
            {0, 0, 9, 8, 0, 0, 0, 3, 6},
            {0, 0, 0, 3, 0, 6, 0, 9, 0}
        };
    }
    private static int[][] createHardPuzzle() {
        return new int[][] {
            {8, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 3, 6, 0, 0, 0, 0, 0},
            {0, 7, 0, 0, 9, 0, 2, 0, 0},
            {0, 5, 0, 0, 0, 7, 0, 0, 0},
            {0, 0, 0, 0, 4, 5, 7, 0, 0},
            {0, 0, 0, 1, 0, 0, 0, 3, 0},
            {0, 0, 1, 0, 0, 0, 0, 6, 8},
            {0, 0, 8, 5, 0, 0, 0, 1, 0},
            {0, 9, 0, 0, 0, 0, 4, 0, 0}
        };
    }
```

```
    private static int[][] createExtremelyHardPuzzle() {
        return new int[][] {
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 3, 0, 8, 5},
            {0, 0, 1, 0, 2, 0, 0, 0, 0},
            {0, 0, 0, 5, 0, 7, 0, 0, 0},
            {0, 0, 4, 0, 0, 0, 1, 0, 0},
            {0, 9, 0, 0, 0, 0, 0, 0, 0},
            {5, 0, 0, 0, 0, 0, 0, 7, 3},
            {0, 0, 2, 0, 1, 0, 0, 0, 0},
            {0, 0, 0, 0, 4, 0, 0, 0, 9}
        };
    }
}
```

二、性能评估

时间复杂度分析

方法	最坏情况	平均情况
possibleNumbers()	$O(n^3)$	$O(n^2)$
LRC()	$O(n^3)$	$O(n^2)$
组合调用	$O(n^4)$	$O(n^3)$

三、缺陷与改进

1. 现存问题

- 缺陷**1**：LRC策略未处理多解冲突
  - 复现步骤：在宫格已有两个空缺时可能误判
  - 修复建议：添加冲突检测逻辑（需要有更多更高效的策略）
- 缺陷**2**：候选数删除使用Integer.valueOf导致性能损耗
  - 性能数据：占总体运行时间的23%
  - 优化方案：改用HashSet或位运算

2. 后续可以进行的优化

- 策略增强：
  - 使用更加高效的策略，引入更多策略
- 交互改进：
  - 提供GUI界面演示策略应用过程
  - 实现求解步骤回放功能

四、需求分析文档附件

1. 引言

1.1 目的

本文档旨在定义数独游戏基本策略求解器的功能需求和非功能需求，明确系统边界和开发目标。

1.2 范围

本系统将实现一个能够应用两种基本策略（Last Remaining Cell和Possible Number）自动解决数独问题的程序，不包括图形用户界面和高级解题策略。

1.3 定义和缩写

- **Last Remaining Cell**：唯一候选数法，当某单元格在行、列或宫中只有一个可能数字时填入该数字
- **Possible Number**：显性唯一候选数法，当某数字在某行、列或宫中只有一个可能位置时填入该数字
- 数独板：9x9的网格，表示数独游戏的当前状态

2. 总体描述

2.1 产品前景

本产品是一个数独解题辅助工具，旨在帮助用户理解基本解题策略，验证解题思路，并作为更复杂数独求解器的基础。

2.2 用户特征

- 初级数独爱好者：学习基本解题策略
- 教育工作者：用于教学演示
- 开发者：作为更复杂求解器的基础组件

2.3 假设与依赖

- 假设输入的数独问题有解且可通过两种基本策略解决
- 依赖标准输入输出环境

3. 详细需求

3.1 功能需求

3.1.1 输入功能

- **FR1.1**：系统应能接受9x9数独板的输入
  - 输入格式：文本文件或命令行输入，空单元格用0或点号表示
  - 示例格式：

```
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

- **FR1.2**：系统应验证输入的数独板是否符合规则
  - 检查行、列、宫中是否有重复数字
  - 检查是否为9x9网格
  - 检查是否包含非法字符

3.1.2 解题功能

- **FR2.1**：系统应实现Last Remaining Cell策略
  - 对于每个空单元格，检查其所在行、列和宫已存在的数字
  - 如果只有一个可能的数字，则填入该数字
- **FR2.2**：系统应实现Possible Number策略
  - 对于每个数字1-9，检查每行/列/宫
  - 如果该数字在某行/列/宫中只有一个可能位置，则填入该数字
- **FR2.3**：系统应能交替应用两种策略直到无法继续或问题解决
  - 记录应用策略的顺序和次数
  - 当两种策略都无法继续应用时终止

3.1.3 输出功能

- **FR3.1**：系统应能显示解题后的数独板
  - 格式清晰的9x9网格
  - 区分原始数字和新填入的数字
- **FR3.2**：系统应能提供解题过程摘要
  - 应用的策略类型和次数
  - 填入的数字总数
  - 解题是否完成

3.2 非功能需求

3.2.1 性能需求

- **NR1.1**：对于简单数独（仅需基本策略即可解决），解题时间应小于1秒
- **NR1.2**：系统应能处理至少1000次策略应用循环

3.2.2 可靠性需求

- **NR2.1**：系统不应改变原始数独板中已填数字
- **NR2.2**：当输入无解时，系统应明确提示而非无限循环

3.2.3 可用性需求

- **NR3.1**：命令行界面应提供清晰的用法说明
- **NR3.2**：错误信息应明确指示问题所在

3.2.4 可维护性需求

- **NR4.1**：代码应有适当注释，特别是策略实现部分
- **NR4.2**：系统应模块化设计，便于添加新策略

3.3 约束

- **C1**：初始版本仅实现两种基本策略，不处理需要猜测的高级数独

- **C2**：使用Python或Java等高级语言实现
- **C3**：不依赖外部数独解题库

4. 用例模型

4.1 主要用例：解决数独

参与者：用户

前置条件：用户提供有效的数独问题

主成功场景：

1. 用户启动程序并输入数独问题
2. 系统验证输入有效性
3. 系统应用Last Remaining Cell策略
4. 系统应用Possible Number策略
5. 重复3-4步直到无法继续或问题解决
6. 系统显示解题结果和过程摘要

扩展：

- 2a. 输入无效：
  1. 系统显示错误信息
  2. 返回步骤1
- 5a. 无法完全解题：
  1. 系统显示部分解题结果
  2. 注明哪些单元格未解决

5. 未来可能的扩展

1. 实现更高级的解题策略
2. 添加图形用户界面
3. 支持不同尺寸的数独变体
4. 添加解题步骤可视化
5. 生成可解的数独题目

6. 验收标准

1. 能正确解决仅需两种基本策略的数独问题
2. 对于无法完全解决的问题，能正确识别并报告
3. 处理标准输入格式并产生标准输出
4. 代码结构清晰，有适当文档和注释
5. 包含测试用例验证所有主要功能

附录A：示例输入输出

示例输入：

```
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

预期输出：

```
解题结果:
5 3 4 |6 7 8 |9 1 2
6 7 2 |1 9 5 |3 4 8
1 9 8 |3 4 2 |5 6 7
------+-------+------
8 5 9 |7 6 1 |4 2 3
4 2 6 |8 5 3 |7 9 1
7 1 3 |9 2 4 |8 5 6
------+-------+------
9 6 1 |5 3 7 |2 8 4
2 8 7 |4 1 9 |6 3 5
3 4 5 |2 8 6 |1 7 9

解题摘要:
应用Last Remaining Cell策略: 15次
应用Possible Number策略: 12次
填入数字总数: 27
解题状态: 完全解决
```

附录B：术语表

- 单元格：数独板中的单个格子，可填入数字1-9
- 行：数独板中的水平9个单元格
- 列：数独板中的垂直9个单元格

- 宫：数独板中的3x3粗线框内的9个单元格