软件工程HW2-数独推理策略实现工程分析报告

学生姓名：陈宝怡 学号：23336029

一、分析报告概述

- **报告目的**：本次报告旨在总结代码实现两种数独策略的开发历程，为后期项目复盘和项目拓展提供重要资料，同时也为深入了解OOAD开发过程积累经验。
- **报告内容**：报告数梳理了完整的数独策略实现开发历程，包含分析、编码、验证和调试四个主要步骤，同时也涵盖了对开发过程和项目本身的理解。

二、开发任务总述

- **开发背景**：课程任务需要
- **开发要求**：根据给定的函数定义，用代码实现数独的两个推理策略（Last Remaining Cell & Possible Number）。可用任何编程语言（开发选用python语言），自行设计数据结构和算法。设计测例，验证、确保函数正确。

三、具体开发流程

1、分析阶段

- **任务名称**：搜索数独相关内容，了解两种策略以及代码实现案例。
- **具体过程**：
  - 在浏览器上搜索两种策略的详细内容和原理，比如LRC策略是如何从确定值出发确定候补值
  - 搜索有无代码实现数独游戏的案例
  - 参考以上搜索结果，总结函数实现的大概逻辑
- **分析结果**：
- Last Remaining Cell：唯一剩余格（遍历行列宫已经出现的数字，未填数字标记数组？）

```
# 从确定值出发--如果一个区域（行列宫）当前单元格之外，其余单元格均不能填写而某个候选值，则可以确定当前单元格的确定值是此值。
# 9 * 9的二维数组表示数独棋盘，元素为0-9，0表示未填写
# 返回数独推理结果，9 * 9 * 9的三维数组，每个元素为当前单元格的候选值
def lastRemainingCellInference (board) {
  初始化res（都为0都不能作为候选值）
  遍历棋盘：
      如果遇到空白单元格：
          判断是否唯一可填（函数）：
              唯一可填则标记res相应位置为0
```

```
    返回res
}
```

- Possible Numberzai：确定可以填入的合法数字集合

```
# 未确定单元格的候选值应当排除其同行列格的确定值，据此可以推断每个未确定单元格的候选值
# 9 * 9的二维数组表示数独棋盘，元素为0-9，0表示未填写
# 返回数独推理结果，9 * 9 * 9的三维数组，每个元素为当前单元格的候选值
def possibleNumberInference (board) {
    初始化res数组（所有概率都为1即都是候选值）
    根据初始棋盘情况更新候选值（已经确定的值则只有对应数字的为1）
    遍历未填单元格：
        遍历未填单元格所在列：
            根据已填数字更新空白单元格候选值
        遍历行和宫同上
    返回res数组
}
```

2、编码阶段

- **任务名称**：根据分析结果编写数独推理策略实现代码
- **具体流程**：
  - 开发选用python语言，整体分为def lastRemainingCellInference ()和def possibleNumberInference ()两大主要函数，同时为了实现某一策略，也含有一些辅助函数
  - 编写def lastRemainingCellInference ()函数以及相关辅助函数
  - 编写def possibleNumberInference ()函数
  - 根据任务输出要求编写主函数

3、验证阶段

- **任务名称**：验证两个推理策略代码的正确性
- **具体流程**：
  - 根据题目给出的例子编写测试用例（数独棋盘board）
  - 运行代码
  - 检验实验结果得出的各个单元格候补值（集）是否和用例预期结果相符

4、调试阶段

- **任务名称**：若程序运行结果与预期不符，修改代码
- **具体流程**：
  - LRC策略得到的结果与PN策略不同，重新思考函数逻辑
  - 回顾函数定义修改代码
  - 重新运行程序验证实验结果

开发流程表格形式

分类	任务名称	起止时间	详情	难点	改进办法
分析	在浏览器上搜索两种策略的详细内容和原理	0:00:00-0:00:42	在浏览器上搜索**两种策略的详细内容和原理**，比如LRC策略是如何从确定值出发确定候补值，其中具体是如何遍历。	起初因关于PN策略的信息较少，**无法区分两个推理策略的不同之处**	向AI询问了LRC和PN两个策略从功能层面、执行顺序和求解能力等区别：**PN主要用于分析空白单元格的潜在可能性，而LRC策略是基于PN策略的得到的信息逐步缩小问题规模**
	搜索有无代码实现数独游戏的案例	0:00:43-0:01:38	在CSDN浏览了一些**解数独游戏的python代码**，其中包含许多算法（回溯、遍历搜索等），为实现函数提供了思路	/	/

分类	任务名称	起止时间	详情	难点	改进办法
	参考以上搜索结果，总结函数实现的大概逻辑	0:01:39-0:02:56	使用obsidian记录分析过程，同时基于前面的搜索结果开始**构想函数实现逻辑。**便于后续编码时有清晰的逻辑指引。	在编写函数逻辑时难以体现处两个策略的功能不同之处	重新回顾题目中函数定义，从定义出发，一步步翻译成代码实现的逻辑
编码	编写last_remaining_inference函数	0:02:57-0:04:52	将前面编写的函数基本逻辑转化为python代码。在实现LRC策略时，借助辅助函数if_unique_valid帮助判断该空白单元格是否唯一可填（即LRC的核心思想）	在编写代码过程中**忽略了时间复杂度**，只是采用了最直接的遍历算法而没有想过算法的优化	通过**设计合理的数据结构和算法流程**，如使用辅助数组记录已出现的数字，减少不必要的遍历和判断，提高算法的执行效率
	编写possible_number_inference函数	0:04:53-0:06:20	同实现LRC策略过程相似，根据想好的函数逻辑编写代码	在Possible Number策略中，多层循环嵌套导致代码可读性较差，且容易出现性能问题	同上，应进行代码优化，**减少不必要的循环和判断**
	根据任务输出要求编写主函数	0:06:21-0:07:12	查看题目要求输出，即需要得到一个三维列表的候选值（集）	/	/

分类	任务名称	起止时间	详情	难点	改进办法
验证	根据题目给出的例子编写测试用例（数独棋盘board）	0:07:13-0:07:27	以题目中给出的PN例题作为测试用例，将数独棋盘转化成一个二维列表	/	/
	运行代码，并检验程序运行结果	0:07:28-0:07:30	运行代码，程序输出两个推理策略得出的候补值（集）。输出形式为Cell (坐标): {候补值}，逐一检查得出的候补值与预期结果是否相同。	设计全面的测试用例，**确保算法在各种情况下都能正确运行**，以及如何准确评估算法的性能	借助AI生成不同难度和类型的数独测试用例，进行全面的测试
调试	LRC策略得到的结果与PN策略不同，重新思考函数逻辑	0:07:31-0:08:10	验证结果时发现LRC推理出的候补值与预期不符，检查代码是否与预期函数逻辑相符，然后检查函数逻辑问题	发现定义的函数逻辑与LRC策略的原理有些不同，**理解的推理过程与题目给出的函数定义有出入**	查找更多的参考资料，同时询问AI过于函数逻辑是否有问题。
	修改代码重新运行	0:08:11-0:08:38	程序运行结果显示LRC策略和PN策略得出的候补值相同，并且与预期结果相同	/	/

四、开发过程总结

- **程序运行结果**：得到的候补值（集）与预期结果相符。

```
----------------------------------------
Possible Number Inference Result:

Cell (0,0):[3, 6]
Cell (0,1):[7]
Cell (0,2):[1, 6]
Cell (0,3):[4]
Cell (0,4):[1, 3, 5]
Cell (0,5):[8]
Cell (0,6):[1, 3, 5]
Cell (0,7):[2]
Cell (0,8):[9]
Cell (1,0):[3, 6, 9]
Cell (1,1):[1, 6, 9]
Cell (1,2):[2]
Cell (1,3):[1, 5, 7, 9]
Cell (1,4):[1, 3, 5]
```

- **经验教训**：
  - 通过本次开发，对数独的两种策略有了更加深入的理解，同时在OOAD开发流程、算法设计、工程实现等方面积累了经验。学会了借助AI和网络资源解决开发过程中的难题。
  - 不足之处在于开发前期忽略了算法的时间复杂度问题，选择了暴力遍历的方法实现函数功能，导致代码在性能方面有不足之处。同时在理解策略原理时不够深入，在编写代码时函数逻辑不清晰。
- **改进方向**：
  - 优化算法：深入研究数独算法，探寻更多优化方法。
  - 深化对于开发原理的分析理解：锻炼自己遇到开发需求时对于原理和流程的分析能力，确保开发过程中头脑清晰。

最后附上具体代码

```python
def last_remaining_inference(board):
    res=[[[0]*9 for _ in range(9)] for _ in range(9)]

    for i in range(9):
        for j in range(9):
            if board[i][j]==0:
                for num in range(1,10):
                    if if_unique_valid(board,i,j,num)==True:
                        res[i][j][num-1]=1
```

```
    return res

def if_unique_valid(board,i,j,num):
    # 判断行内其余格可不可以填
    for col in range(9):
        if col!=j and board[i][col]==num:
            return False
    # 判断列
    for row in range(9):
        if row!=i and board[row][j]==num:
            return False
    # 判断宫
    box_r=(i//3)*3
    box_c=(j//3)*3
    for u in range(box_r,box_r+3):
        for v in range(box_c,box_c+3):
            if (u!=i or v!=j) and board[u][v]==num:
                return False

    return True
def possible_number_inference(board):
    res=[[[1]*9 for _ in range(9)] for _ in range(9)]

    for i in range(9):
        for j in range(9):
            if board[i][j]!=0:
                for num in range(9):
                    res[i][j][num]=0
                res[i][j][board[i][j]-1]=1

    for i in range(9):
        for j in range(9):
            if board[i][j]==0:
                # 遍历未填单元格所在列（根据确定值排除）
                for col in range(9):
                    if board[i][col]!=0:
                        res[i][j][board[i][col]-1]=0
                # 遍历所在行
                for row in range(9):
                    if board[row][j]!=0:
                        res[i][j][board[row][j]-1]=0
                # 遍历宫格
                box_r=(i//3)*3
                box_c=(j//3)*3
                for u in range(box_r,box_r+3):
                    for v in range(box_c,box_c+3):
                        if board[u][v]!=0:
                            res[i][j][board[u][v]-1]=0

    return res
```

```
test_board = [
    [0, 7, 0, 4, 0, 8, 0, 2, 9],
    [0, 0, 2, 0, 0, 0, 0, 0, 4],
    [8, 5, 4, 0, 2, 0, 0, 0, 7],
    [0, 0, 8, 3, 7, 4, 2, 0, 0],
    [0, 2, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 3, 2, 6, 1, 7, 0, 0],
    [0, 0, 0, 0, 9, 3, 6, 1, 2],
    [2, 0, 0, 0, 0, 0, 4, 0, 3],
    [1, 3, 0, 6, 4, 2, 0, 7, 0]
]


last_res=last_remaining_inference(test_board)
possible_res=possible_number_inference(test_board)

print('Last Remaining Cell Result:\n')
for i in range(9):
    for j in range(9):
        print(f'Cell ({i},{j}):{[k+1 for k,v in enumerate(last_res[i][j]) if
v==1]}')

print('---------------------------------------')

print("Possible Number Inference Result:\n")
for i in range(9):
    for j in range(9):
        print(f'Cell ({i},{j}):{[k+1 for k,v in enumerate(possible_res[i]
[j]) if v==1]}')
```