软件工程作业分析报告

姓名：林璐 学号：23336138

开发任务列表

分类	任务名称	起止时间	详情	难点	改进想法
分析	阅读任务开发材料	0:06.31~0:48.33	通过阅读雨课堂作业要求文档，了解数独任务的背景和要求， 为后续查询相关资料做准备。	一开始对任务要求的理解有偏差，认为需要完成整个数独，实际上是需要完成两个策略。	要仔细阅读开发文档
	阅读百度数独介绍	0:48.33~1:52.71	通过上网查询资料，了解数独游戏的规则。	-----	------
	了解数独LRC策略	1:52.71~3:12.89	通过上网查询资料，比如CSDN、	个别网页	可以多看一些

分类	任务名称	起止时间	详情	难点	改进想法
			百度等网页了解什么是数独LRC策略，同时查看别人的实例代码，了解LRC策略在编程中的实际应用，同时结合任务开发材料，思考如何在自己的代码中实现数独LRC策略。	给出的资料与数独LRC策略不太一样，要注意辨别	网页资料，然后并做好整合
	了解数独possible number策略	3:12.89~4:40.31	通过上网查询资料，比如CSDN等网站，了解什么是数独的possibleNumber策略，同时查看别人的实例代码，了解possible Number策略在编程中的实际应用，同时结合任务开发材料，思考如何在自己的代码中实现数独possibleNumber策略	-----	------
环境	创建py文件和分析报告文件	4:43.58~5:06.50	创建py文件和分析报告文件，方便后续记录代码和开发过程	-----	------
	调整python运行环境	10:56.81~12:03.09	一开始在Python里运行代码发现点击	-----	------

分类	任务名称	起止时间	详情	难点	改进想法
			三角形箭头不能运行代码， 会出现没有安装numpy库的情况， 可我明明已经用pip命令安装 过了。结合之前的编程经验，可能是因为python解释器不能正确 调用的问题，所以在运行的时候要明确写出python解释器的位置 和运行文件的绝对地址。		
编码	利用gpt得到初始代码	5:09.61~6:24.15	将分析部分得到的开发思路和gpt交流，让gpt帮忙生成代码	-----	------
	审查gpt提供的初始代码	6.24:15~10:18:25	仔细阅读gpt提供的初始代码，结合此前分析的思考过程和任务要 求文档，思考gpt提供的初始代码是否正确。在检查过程中，发现 gpt一开始提供的是和我起初理解错误的版本一样，误以为要完成 整个程序的设计，实则是需要完成两个策略。所以又将	gpt并不能提供和我们想象中一样的代码，所以还需要带着思考去	可以反复查看任务开发要求， 比对代码是否符合输入输出要求。

分类	任务名称	起止时间	详情	难点	改进想法
			要求和我自己的思路过程和gpt做了交互，最后查看代码发现思路和自己所想的大致相同，准备在python环境中运行检查。	推敲看代码是不是我们想要的	
验证	设计样例测试初始代码	10:21.36~10:56.81	设计一个测试样例写在程序里的main函数当中，然后尝试运行代码	设计样例的时候要确保样例数独是可以解决的	可以在网上搜索有明确解决步骤的数独例子，这样保证它是能解的
	继续验证初始代码，先测LRC策略部分	12:03.09~17:00.76	成功在python环境下运行代码之后，正式开始验证初始代码的正确性，运行后发现两个策略同时验证会显得输出很长，不方便验证结果，所以先把possible_number的部分注释掉，先测LRC策略的部分。在查看LRC策略输出结果时，发现目前输出不是说某一格能填	输出的结果和想象中的不一致。感觉把两个策略融合起来输出，没有办法	让两个策略分开输出，重新定义结果的输出格式，让输出更加清晰。

分类	任务名称	起止时间	详情	难点	改进想法
			哪个数，而是写出了所有元素的可能，所以输出有问题，需要调试修改。	验证出其中一个策略的正确性。	
	继续验证修改后的LRC部分	21:17.89~22:49.54	继续用测例验证修改后的LRC部分，但是很不幸的发现，修改后的LRC策略并不符合要求，它仍然显示出了所有数的可能情况，而非某个格能确定的那个数，还需要进一步的调试	-----	------
	第三次验证修改后的LRC部分	23:31.43~27:20.99	继续用测例验证修改后的LRC部分，终于程序输出了各个格子能确定填入数独的那个数字。再通过gpt生成了其他测例，包括数独测例未填入的状态和完全填好的状态，方便对照。在运行其他测例的时候发现，输出的格子会有重复的，所以思考在调试中用集合来做输出的数据结构，然后用	输出的元素居然有重复的，会导致输出的列表异常的冗长	利用Python里集合的排他性，消除掉重复元素。

分类	任务名称	起止时间	详情	难点	改进想法
			集合的排他性把重复元素消除掉。		
	第四次验证修改后的LRC部分	29:11.42~31:47.45	继续用测例验证修改后的LRC部分，发现后续的测例都能正确输出 某个格子能填出来的唯一值。	-----	------
	测试possibleNumber策略部分	31:47.63~36:05.02	测试完LRC策略后，开始测试possibleNumber策略。通过gpt设计出 不同的测例，然后测试possibleNumber策略，观察其能否输出每一个 空的格子的所有能填的数字，验证的时候所有的测试样例都通过了。	-----	------
调试	修改LRC策略	17:00.76~21:17.89	根据开发文档，再次确认LRC策略的所有要求，然后再和gpt交互， 尝试修改出符合要求的LRC策略的代码。	调试修改代码的时候容易陷入混乱，不知道目前写出	要多看任务开发文档，检查思路是 否正确

分类	任务名称	起止时间	详情	难点	改进想法
				来LRC策略的输出是不是已经有所修改，还是仍然陷入原来想法里面。	
	继续修改LRC策略	22:49.94~23:31.43	向gpt指明要求输出是某一格能确定的那个数。	说明需求的时候不够具体会产生歧义	要明确开发需求
	继续修改LRC策略的输出	27:20.99~29:11.42	从验证中发现数独输出里有重复的，会导致输出的结果非常长，不够直观。所以在输出的时候加上集合类型，这样能够消去原来相同的元素。	-----	------

代码

```
import numpy as np

def find_possible_numbers(grid, row, col):
    """返回某个单元格可以填入的所有可能数字"""
    if grid[row, col] != 0:
        return []

    possible = set(range(1, 10))

    # 去除所在行、列已存在的数字
    possible -= set(grid[row, :])
    possible -= set(grid[:, col])

    # 去除所在 3x3 宫格已存在的数字
    start_row, start_col = (row // 3) * 3, (col // 3) * 3
    possible -= set(grid[start_row:start_row+3, start_col:start_col+3].flatten())

    return list(possible)

def lastRemainingCellInference(grid):
    """使用 Last Remaining Cell 策略推理确定的值，返回确定的单元格集合"""
    candidates = np.zeros((9, 9, 9), dtype=int)
    determined_values_set = set()  # 使用集合避免重复记录

    # 计算每个空格的初始候选数集
    for i in range(9):
        for j in range(9):
            if grid[i, j] == 0:
                possible = find_possible_numbers(grid, i, j)
                for num in possible:
                    candidates[i, j, num - 1] = 1

    # 遍历所有数字 1-9，检查 Last Remaining Cell 规则
    for num in range(1, 10):
        for i in range(9):
            # 检查行
            row_positions = [(i, j) for j in range(9) if candidates[i, j, num - 1]
== 1]
            if len(row_positions) == 1:
                row, col = row_positions[0]
                if (row, col, num) not in determined_values_set:  # 检查是否已经记录
过该位置
                    grid[row, col] = num
                    determined_values_set.add((row, col, num))  # 记录已填充的单元格

            # 检查列
            col_positions = [(j, i) for j in range(9) if candidates[j, i, num - 1]
== 1]
            if len(col_positions) == 1:
                row, col = col_positions[0]
                if (row, col, num) not in determined_values_set:
                    grid[row, col] = num
```

```
                    determined_values_set.add((row, col, num))

        # 检查 3x3 宫格
        for box_row in range(3):
            for box_col in range(3):
                positions = []
                for i in range(3):
                    for j in range(3):
                        row, col = box_row * 3 + i, box_col * 3 + j
                        if candidates[row, col, num - 1] == 1:
                            positions.append((row, col))
                if len(positions) == 1:
                    row, col = positions[0]
                    if (row, col, num) not in determined_values_set:
                        grid[row, col] = num
                        determined_values_set.add((row, col, num))

    # 返回所有确定的单元格（行, 列, 数字）
    return list(determined_values_set)

def possible_number(grid):
    """返回每个空单元格的所有可能数字集合"""
    possible_numbers_grid = np.zeros((9, 9), dtype=object)  # 用于存储每个格子的候选
数字集合

    for i in range(9):
        for j in range(9):
            if grid[i, j] == 0:  # 如果该格为空格
                possible_numbers_grid[i, j] = find_possible_numbers(grid, i, j)  #
计算并存储候选数字

    return possible_numbers_grid


# 示例数独（0 表示空白格）
sudoku_grid = np.array([
    [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]
])

# 计算确定的值
determined_values = lastRemainingCellInference(sudoku_grid)
print(determined_values)

#测试一下possiblenumber策略

# 获取每个空单元格的所有可能数字
possible_numbers_grid = possible_number(sudoku_grid)

# 打印每个格子的候选数字
```

```
for i in range(9):
    for j in range(9):
        if possible_numbers_grid[i, j]:
            print(f"Cell ({i}, {j}) can have the following possible numbers:
{possible_numbers_grid[i, j]}")
```