

**课程设计报告**

**题目：基于SAT的对角线数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： 计科2303**

**学 号： U202315116**

**姓 名： 邱月**

**指导教师： 卢萍**

**报告日期： 2024-9-28**

**计算机科学与技术学院**

**任务书**

**1设计内容**

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于DPLL算法实现一个完备SAT求解器，对输入的CNF范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

**2设计要求**

要求具有如下功能：

**（1）输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)

**（2）公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)

**（3）DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)

**（4）时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)

**（5）程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)

**（6）SAT应用：**将数双独游戏[5]问题转化为SAT问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为SAT问题的具体方法可参考文献[3]与[6-8]。(15%)

**3参考文献**

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

[2] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[3] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

[4] Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243

[5] 360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

Twodoku： https://en.grandgames.net/multisudoku/twodoku

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

[7] Ins Lynce and Jol Ouaknine. Sudoku as a sat problem. In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale. Springer, 2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

[10] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191

**目录**

**任务书** I

**1引言** 1

1.1课题背景与意义 1

1.1.1课题背景 1

1.1.2课堂意义 1

1.2国内外研究现状 2

1.3课程设计的主要研究工作 2

1.3.1基于DPLL算法的SAT求解器 2

1.3.2基于SAT求解器的对角线数独的生成，归约和求解 3

**2系统需求分析与总体设计** 4

2.1系统需求分析 4

2.1.1SAT求解器 4

2.1.2对角线数独的生成，归约和求解 4

2.2系统总体设计 4

**3系统详细设计** 6

3.1有关数据结构的定义 6

3.2主要算法设计 7

3.2.1cnf文件读取及存储 7

3.2.2DPLL核心算法 8

3.2.3res输出和保存 14

3.2.4对角线数独应用 15

**4系统实现与测试** 18

4.1系统实现 18

4.2系统测试 19

4.2.1solver 模块测试 19

4.2.2X-Sudoku模块测试 22

**5总结与展望** 27

5.1全文总结 27

5.2工作展望 27

**6体会** 28

**参考文献** 29

**附录** 30

**1引言**

**1.1课题背景与意义**

**1.1.1课题背景**

对角线数独是普通数独的一种衍生变体。一般的数独是用1~9的填充一个9\*9的数独残局，使数独的每一行，每一列和每个3\*3的宫格中都包含1~9这九个数，且每行每列和3\*3宫格中的数字没有重复。而对角线数独在此基础上有要求两条对角线上也要包含1~9这九个数且不出现重复。这使解对角线数独变得更加复杂也更加有趣。

而求解数独的难度会随着数独的规模和约束条件增加，而为了高效地解决数独难题，引入SAT问题求解器是一个有效的算法。

SAT问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的NP完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本课题主要基于DPLL算法实现一个高效的SAT问题求解器，进而应用于对角线数独游戏求解。

**1.1.2课堂意义**

对角线数独的求解随着规模和约束条件的增加而变得复杂，人工求解效率低且难度大。引入SAT求解器将数独的约束条件转化为布尔公式求解，可以显著提升求解效率。通过进一步优化算法，还能实现对角线数独的自动生成与验证，增强游戏的互动性与可玩性。

在此过程中，将SAT求解器应用于数独游戏，不仅加深了我们对DPLL算法的理解和改进，同时为SAT求解器在其他复杂问题领域的应用提供了宝贵经验和启发。

**1.2国内外研究现状**

国内外对基于SAT的对角线数独游戏求解程序都进行了大量且深入的研究，并取得了一定的成果。

国外在算法优化、SAT求解器性能提升和应用领域拓展方面都研究出了显著成果，如在DPLL的基础上发展而来CDCL算法，加入了冲突学习和子句传播的技术，显著提升了求解效率，其在求解大规模问题方面表现出色。而针对对角线数独，研究者们提出了多种优化算法。例如，结合启发式方法（如最小剩余值启发式）和局部搜索优化技术来改进SAT求解器在处理对角线数独时的性能。同时，有效的约束传播技术在对角线数独中也被广泛应用。通过提前传播约束，可以减少搜索空间，提高求解效率。此外，研究者们还在继续探索如何更好地利用对角线约束进行剪枝和冲突学习。

而国内研究也在逐步跟进，致力于让其在实际问题中发挥作用，包括硬件验证、优化问题等领域，希望其应用于实际工业问题中。在算法优化方面，大量研究者对对角线数独的SAT建模技术进行了深入探索，在传统数独建模的基础上，研究了如何高效地处理对角线约束，并提出了相应的改进策略。此外，研究者们还尝试通过多种技术手段提高对角线数独的求解效率，例如改进启发式策略、优化冲突学习机制等。

随着研究的不断深入和技术的不断进步，数独问题的建模和求解将不断优化， SAT求解器也将有望应用在更多实际问题和其他领域。

**1.3****课程设计的主要研究工作**

**1.3.1基于DPLL算法的SAT求解器**

为了构建一个高效的SAT求解器，首先需要设计合适的数据结构来存储SAT问题中的变元、文字、子句和公式等信息。这些数据结构必须能够高效地进行操作，例如插入、删除、查找和修改，以支持DPLL算法的高效执行。我们从基本的DPLL算法入手，构建一个能够正确读取、存储和求解给定CNF文件的求解器。在此基础上，我们利用时间处理函数精确记录DPLL算法的执行时间（以毫秒为单位），为后续的性能分析提供数据支持。

在求解器的实现过程中，我们针对DPLL算法的核心步骤进行优化设计。具体来说，我们可以优化存储结构以减少空间开销，提高访问效率；或者改进分支变元的选取策略，如采用VSIDS（变量状态独立分数）启发式算法，以减少回溯次数。此外，还可以结合冲突驱动学习（CDCL）等先进技术，进一步提高求解器的性能。最终，通过性能测试和比较，提供量化的性能提升率，明确优化的效果，为进一步的算法改进提供数据支撑。

**1.3.2基于SAT求解器的对角线数独的生成，归约和求解**

我们将随机生成对角线数独的完整解，即终盘作为基础，再通过挖洞法来生成游戏盘和提示数。挖洞法的核心是选择合理的位置删除数字，同时保证数独具有唯一解，这样可以调节数独的难度。生成的数独盘面需满足对角线数独的特殊规则：每条对角线上的数字不能重复。

为了将数独问题转换为SAT问题进行求解，我们首先将对角线数独的规则约束，如行列唯一性、宫格唯一性以及对角线唯一性，转化为布尔约束条件，并表示为CNF公式。具体而言，每个数字在每个位置的唯一性用布尔变量表示，通过合取范式（CNF）表达其逻辑关系。接着，将这些CNF公式输入到我们设计的SAT求解器中，求解器可以判断数独问题是否有解，以及求出其唯一解。

通过这种归约与求解方法，不仅能自动生成可玩性高的数独游戏，还可以验证生成的数独是否具有唯一解。这种方法为数独游戏开发提供了自动化的技术支持，并展示了SAT求解器在游戏领域的应用潜力，为进一步将其推广至更广泛的问题求解提供了可能。

**2****系统需求分析与总体设计**

**2.1****系统需求分析**

**2.1.1 SAT求解器**

系统需要实现对给定 CNF 文件的正确读取与解析，并将其转换为内部数据结构存储。使用标准 DPLL 算法对 SAT 问题进行求解，并在求解过程中记录执行时间（精确到毫秒）。在此基础上，优化 DPLL 算法的某些方面（如分支变元的选取策略、存储结构等），以提高求解效率。通过对比基础算法和优化算法的求解时间，系统将展示具体的优化性能结果。此外，系统还需支持基于不同策略选择的优化算法求解，并将结果输出到指定文件中。

**2.1.2对角线数独的生成，归约和求解**

　　系统将基于 SAT 求解器模块，实现对角线数独的生成、规约、求解及验证功能。首先，系统随机生成具有唯一解的对角线数独，并将其转换为等价的 SAT 问题格式（CNF 文件）。接着，通过 SAT 求解器对该 CNF 文件进行求解，并将结果转换回数独网格进行展示。此外，系统支持玩家交互功能，如难度选择、解题验证、答案偷看等，以提升游戏的互动性和可玩性。

**2.2系统总体设计**

系统主要分为两个功能模块：SAT 求解器模块和对角线数独游戏模块。

**2.2.1 SAT求解器模块**

CNF 文件读取与存储：读取给定的 CNF 文件，解析并将其存储为内部数据结构。

CNF 文件内容输出：将内部数据结构转换为 CNF 格式并输出，便于查看。

DPLL 求解 SAT 问题：使用标准 DPLL 算法求解 SAT 问题，支持基础和优化两种算法求解。

输出求解结果与运行时间：将求解结果与执行时间输出，并显示至界面或保存至文件。

保存答案至结果文件：将求解的 SAT 问题解保存至指定的 .res 文件，便于后续验证与比较。

**2.2.2对角线数独游戏模块**

数独游戏生成：根据用户选择的难度，生成具有唯一解的对角线数独，并将其转换为 CNF 文件格式。

数独求解：使用 SAT 求解器对生成的数独进行求解，并将结果返回至数独网格展示。

DPLL 答案验证：玩家输入答案后，系统验证其正确性，并提供反馈。

DPLL 求解参考答案：使用 DPLL 算法生成参考答案，支持玩家偷看和查看最终解。

系统界面支持各模块间的灵活切换，用户可随时在不同功能模块间进行切换，实现流畅的交互体验。

下图是整个系统的模块结构图：

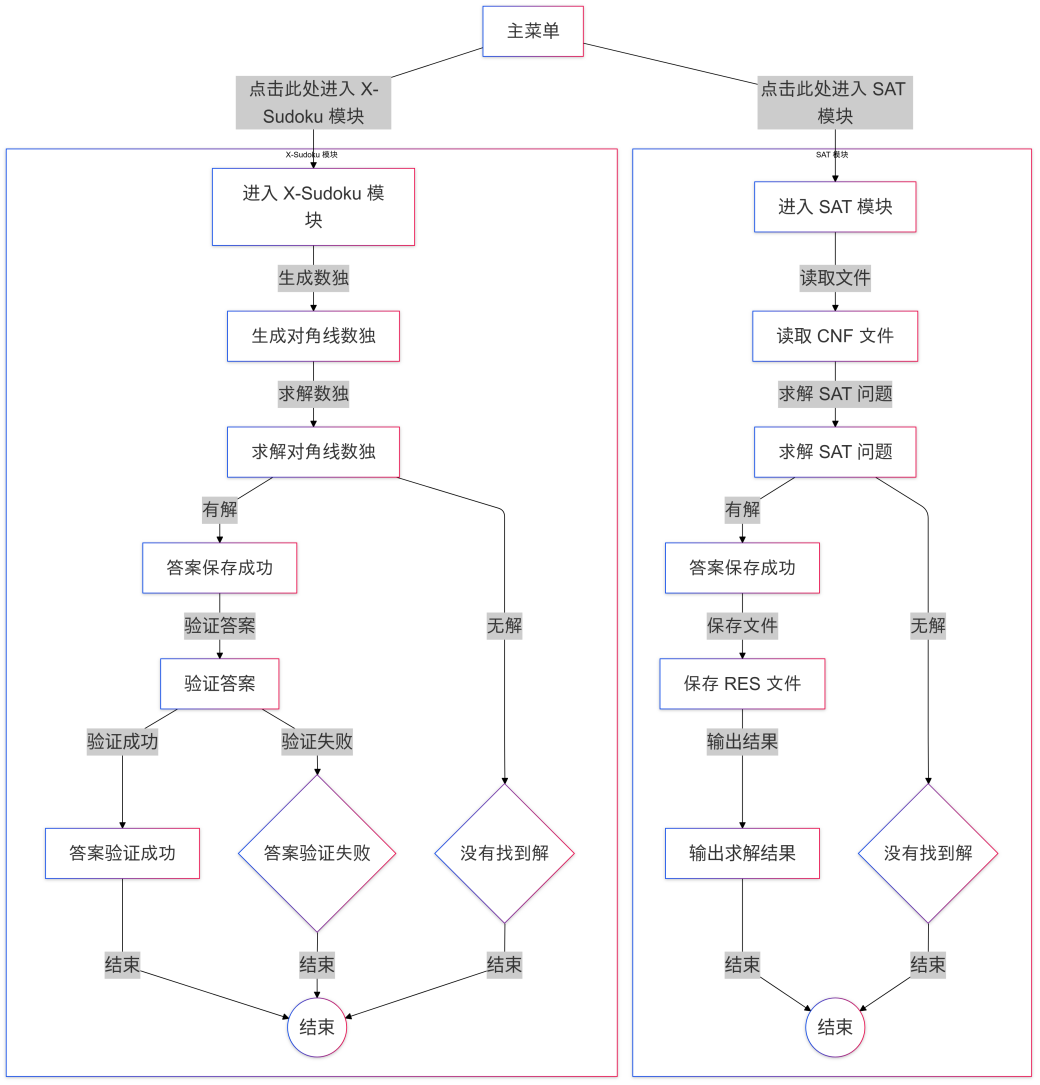


图2-1 系统模块结构图

**3系统详细设计**

**3.1有关数据结构的定义**

系统中主要涉及两种数据类型。

第一种是用于表示 SAT 问题的 CNF 数据，包括子句集和子句中包含的变元。每个子句（Clause）由多个变元（Literal）构成，变元的数据类型为整数，并采用二维链表进行存储。子句是变元的一维链表，多个子句组合成子句集。CNF 公式通过指向第一个子句的指针关联各个子句。

第二种数据类型用于表示数独游戏的数据，包括行、列和数值，均为整数类型，采用二维数组进行存储。数独游戏中的每个单元格通过特定公式与 SAT 问题中的变元相对应，即第 i 行第 j 列的值为 k 转换为变元的公式为 (i-1)\*81 + (j-1)\*9 + k。如果单元格不包含该数值，则取该变元值的负数。

表1-1 数据类型表

|  |  |  |  |
| --- | --- | --- | --- |
| 数据结构名 | 数据项 | 数据类型 | 数据名 |
| Literal | value | int | 变元值 |
|  | right | Literal \* | 指向同子句下一个变元 |
| ClauseNode | right | Literal\* | 指向子句中第一个变元 |
|  | down | ClauseNode\* | 指向下一个子句 |
| grid | i | int | 数独行 |
|  | j | int | 数独列 |
|  | k | int | 数独单元格中的值 |

数据结构的逻辑关联如下：每个子句包含多个变元，通过指向变元集第一个变元的头指针进行关联。子句集通过指向第一个子句的指针管理所有子句，形成关于变元的二维链表。数独数据与 SAT 问题变元紧密关联，通过前述公式进行转换，保证数独问题能够通过 SAT 求解器进行解析和求解。

接下来给出数据结构的逻辑关联图：

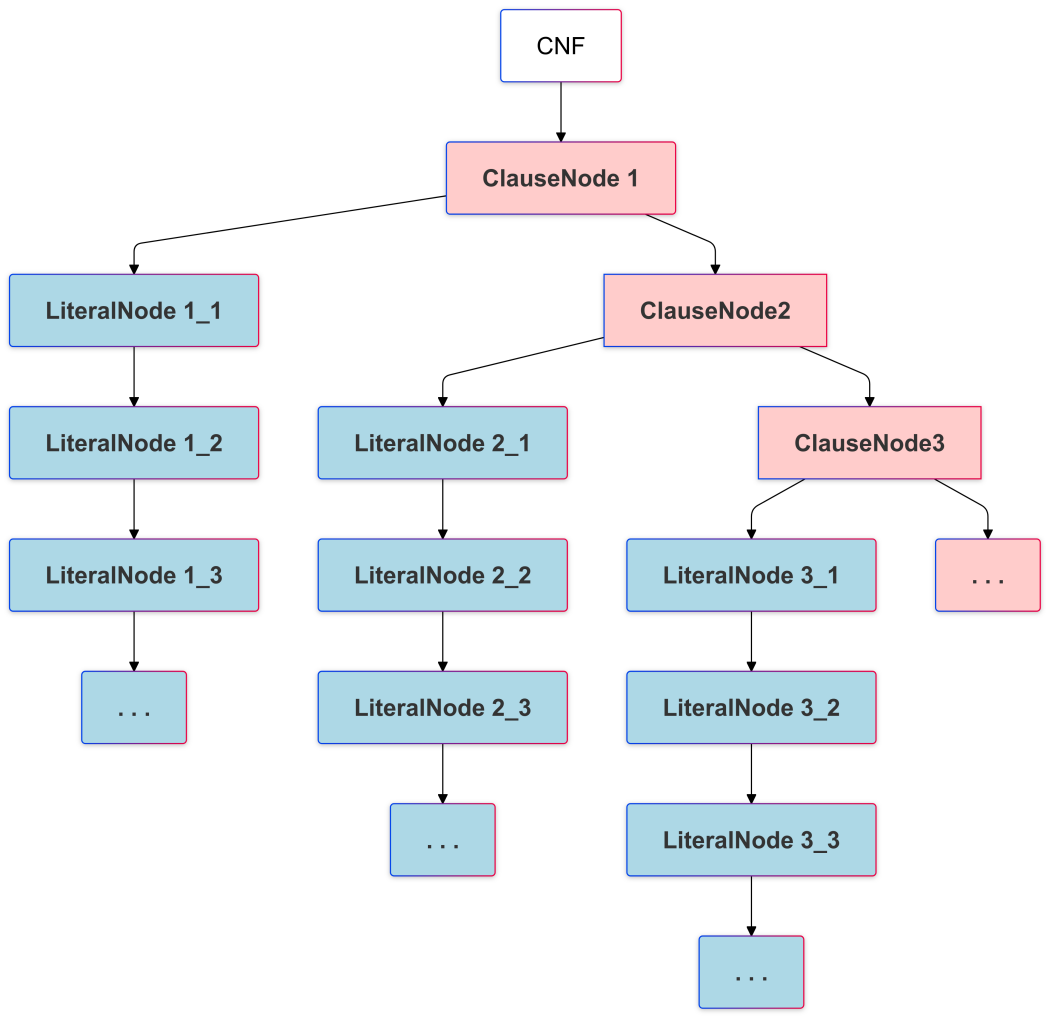


图3-1 数据结构关联图

**3.2 主要算法设计**

本系统的主要算法设计围绕 CNF 文件的读取与存储、DPLL 核心算法的实现以及对角线数独的生成与求解展开。下面对每个主要算法以及核心函数的实现逻辑进行详细描述。

**3.2.1 cnf文件读取及存储**

CnfParser 函数负责从 CNF 文件中读取数据并将其存储为二维链表结构。以下是 CnfParser 函数的详细算法实现：

1. 传入 CNF 文件的路径，尝试打开文件。如果文件无法正常打开，则返回 NULL 并打印错误信息。
2. 跳过文件中的注释行（以 'c' 开头）和头信息行（以 'p' 开头），初始化 CNF 的链表结构。
3. 定义指向当前读取子句的指针 cnf 和指向当前读取变元的指针 literal，初始化 CNF 的尾指针 cnf\_tail。
4. 通过循环逐行读取文件内容，遇到 '0' 结束符时，表示当前子句读取完毕，将该子句指针连接到 CNF 的链表中。
5. 对每个变元，动态分配 LiteralNode 节点并将其数据存储在 literal->data 中，如果变元绝对值大于 max\_var，更新最大变元编号。
6. 完成读取后关闭文件，并将构造好的 CNF 链表头指针返回给调用。

**核心函数：CnfParser**

CnfParser 函数读取 CNF 文件内容并构建 CNF 的链表表示形式。每个子句作为链表的一个节点，子句中的每个变元作为该节点中的 LiteralNode 链表。

**核心函数：clause\_print**

clause\_print 函数用于打印 CNF 公式内容。遍历 CNF 链表，将每个子句中的变元按顺序打印到指定的输出文件中。

下图是CNF 文件读取及存储流程：

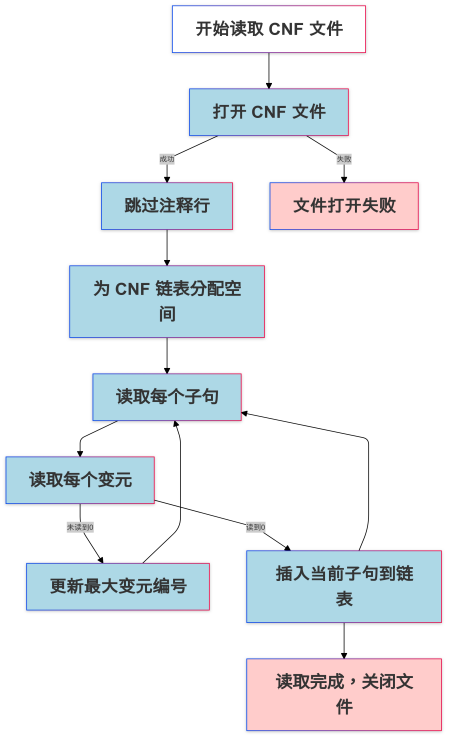


图3-2 CNF文件读取及存储流程

**3.2.2 DPLL核心算法**

DPLL 算法在 solver.c 和 better\_solver.c 文件中实现，是 SAT 问题求解的核心算法。该算法通过单子句规则化简和分裂策略进行求解。以下是 DpllSolver 函数的详细算法实现：

1. **循环遍历子句集**，调用 IsUnitClause 函数查找单子句。如果找到单子句，则将该单子句的值记录在真值表中，并调用 RemoveVar 函数将该变元相关的子句删除。
2. 如果未找到单子句，调用 PickVar 函数选择一个变元进行分裂，并递归调用 DpllSolver 函数检测其可满足性。若满足，则返回 SAT，若不满足，则对变元取反值再次递归调用。
3. 在每次递归调用前，调用 CopyS 函数对当前 CNF 进行备份，并在递归结束后释放备份的空间。

##### Untitled diagram-2024-09-28-125207DPLL算法逻辑如下：

图3-3 DPLL算法逻辑

##### 核心函数：IsUnitClause

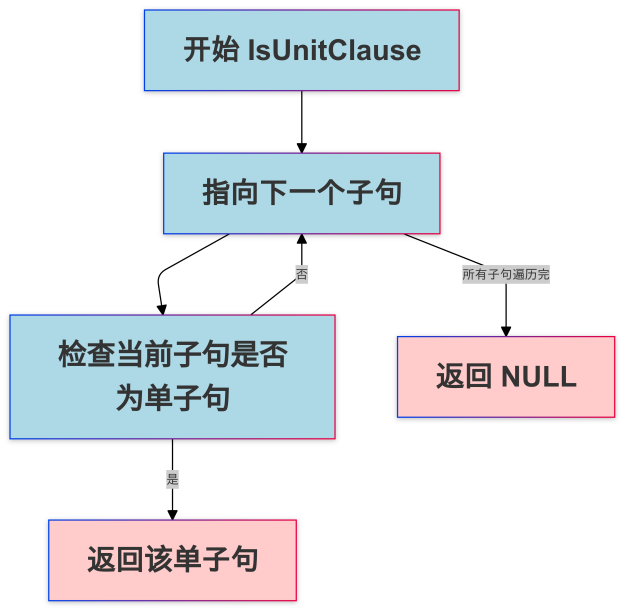
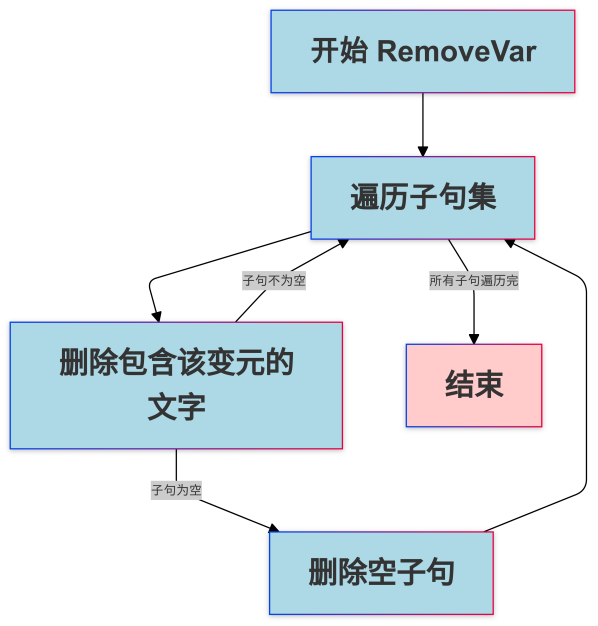
IsUnitClause 函数用于查找子句集中是否存在单子句。遍历每个子句，如果子句中只有一个变元，则返回该子句的指针；否则返回 NULL，表示未找到单子句。

图3-4 IsUnitClause函数实现

##### 核心函数：RemoveVar

RemoveVar 函数用于从 CNF 子句集中删除与指定变元相关的子句和变元。遍历 CNF 链表，调用 DeleteClause 和 DeleteLiteral 函数删除包含该变元的子句和相关的变元。

##### 

图3-5 RemoveVar函数实现

##### 核心函数：DeleteClause

DeleteClause 函数用于删除指定子句及其下所有的变元节点。释放当前子句中所有变元节点的内存空间，并删除该子句节点。

##### 核心函数：DeleteLiteral

DeleteLiteral 函数用于删除子句中与指定变元相关的节点。遍历指定子句中的所有变元节点，删除与指定变元一致的节点，并将其内存释放。

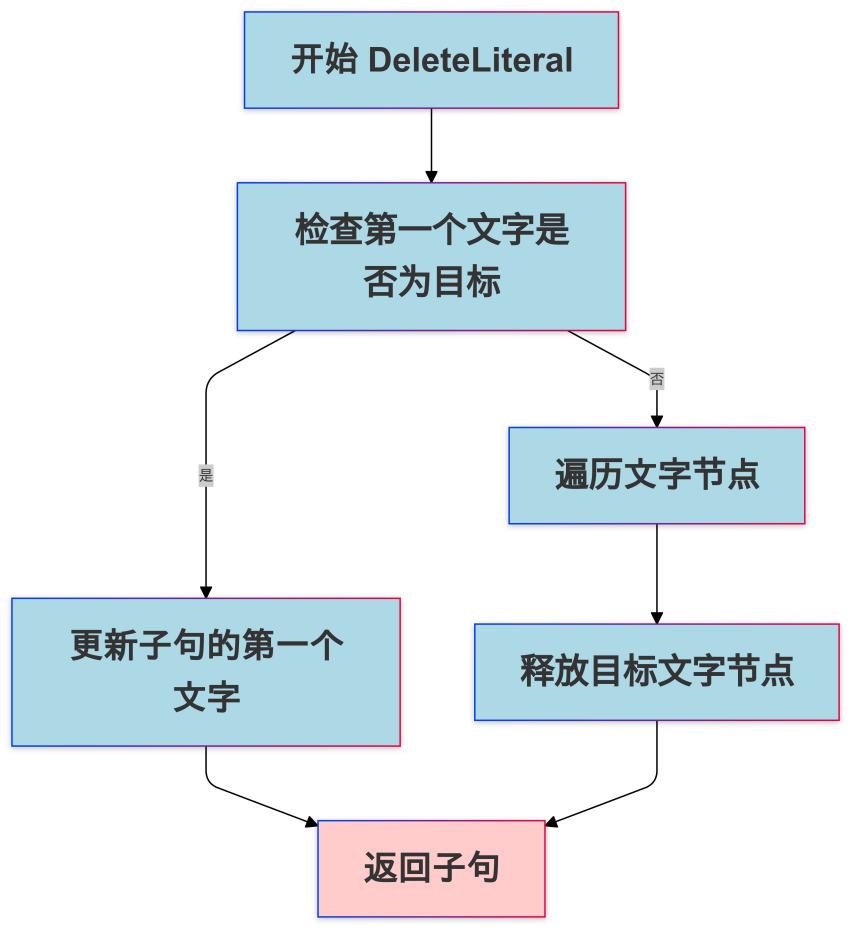
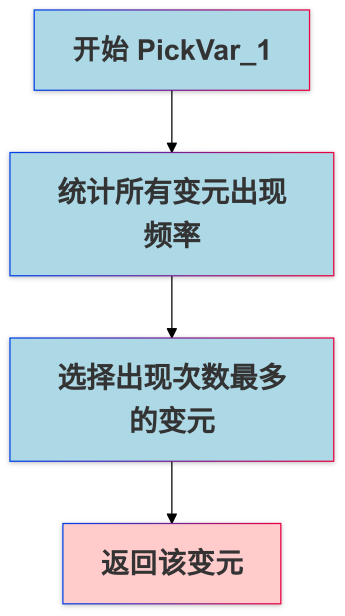


图3-6 DeleteLiteral函数实现

DPLL 算法中，变元选择策略至关重要。本系统中实现了三种变元选择策略，分别为 PickVar\_1、PickVar\_2 和 PickVar\_3。

#### 算法 PickVar\_1

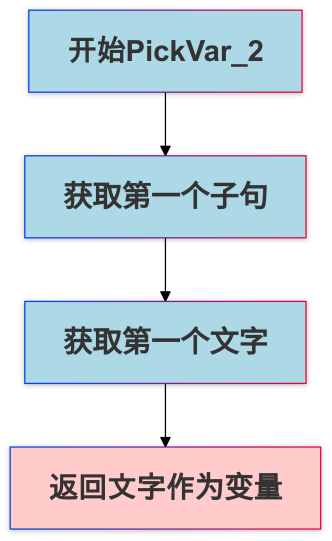
PickVar\_1 选择子句集中出现次数最多的变元，既考虑正变元也考虑负变元的出现次数。该算法先遍历所有子句，统计每个变元的出现频率，然后返回出现频率最高的变元。



#### 图3-7 PickVar\_1函数实现

#### 算法 PickVar\_2

PickVar\_2 选择第一个子句中的第一个变元作为分裂变元。该算法简单地返回子句集中第一个变元，无需额外计算。



#### 图3-8 PickVar\_2函数实现

#### 算法 PickVar\_3

PickVar\_3 在较短的子句中选择出现次数最多的变元，并考虑正负。该算法计算每个变元在较短子句中出现的权重，并选择权重最高的变元。

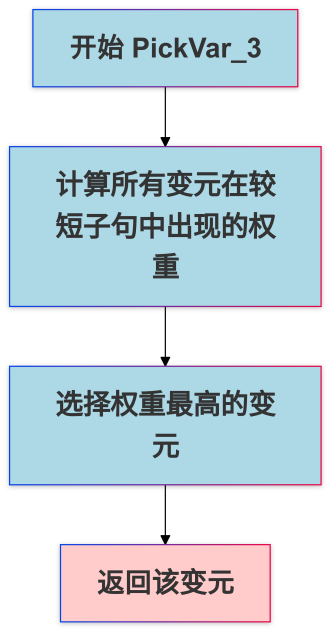
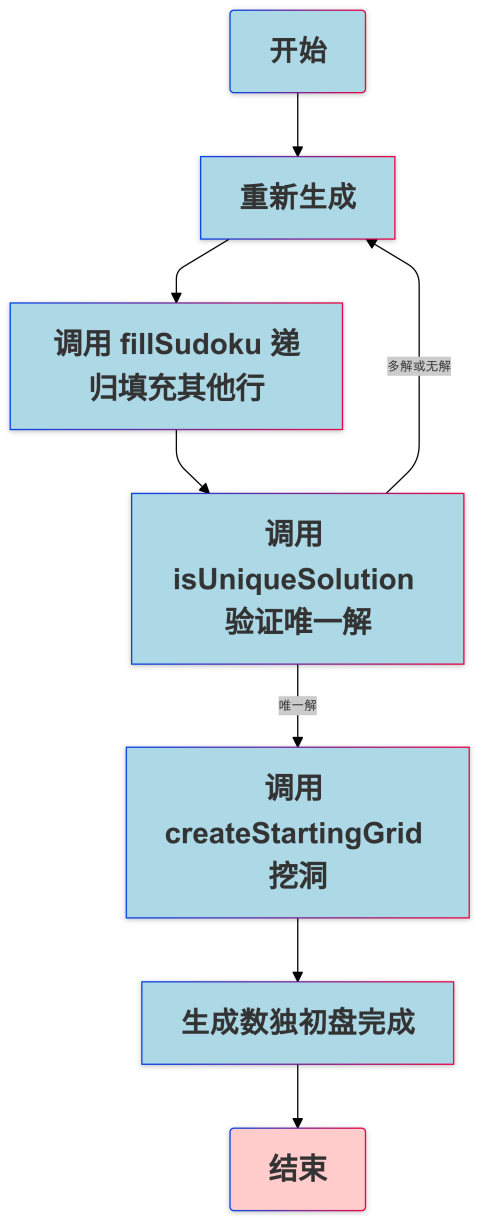


图3-9 PickVar\_3函数实现

**3.2.3 对角线数独核心算法**

对角线数独生成与求解算法包括数独问题的生成、挖洞、求解与验证过程。主要在 generate\_diagonal\_sudoku.c、cnf\_to\_grid.c 和 sudoku\_solver.c 文件中实现。以下是该算法的详细设计：

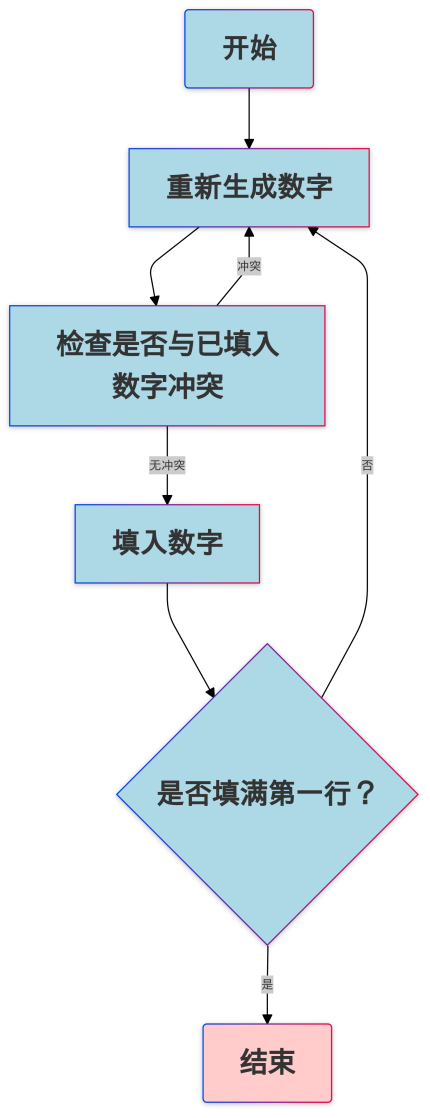
#### 生成对角线数独算法

1. 随机生成第一行数据，采用 randomFirstRow 函数生成数独第一行的数字。
2. 调用 fillSudoku 函数递归填充其他行的数据。该函数通过检查是否符合数独约束（行、列、3x3 子盒及对角线约束），递归地生成合法的数独初盘。
3. 利用 isUniqueSolution 函数验证生成的数独初盘是否有唯一解。若存在多解或无解，重新生成数独初盘。
4. 利用 createStartingGrid 函数根据设定难度挖去一定数量的数，使生成的数独依然满足唯一解的要求。
5. 

#### 图3-9 对角线数独核心算法逻辑

#### 核心函数：randomFirstRow

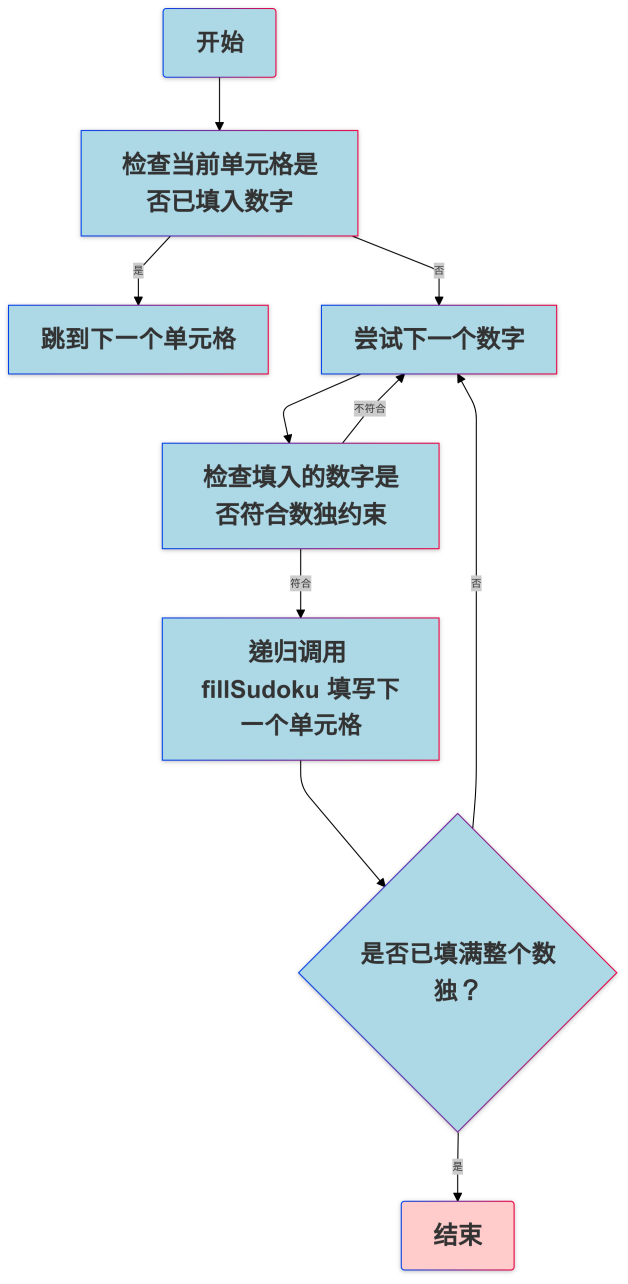
randomFirstRow 函数用于随机生成数独第一行的数字。生成的数字范围是 1 到 9，并且不允许重复。该函数采用随机数生成与冲突检测的策略，直到第一行填充完成。



#### 图3-10 randomFirstRow 函数实现

#### 核心函数：fillSudoku

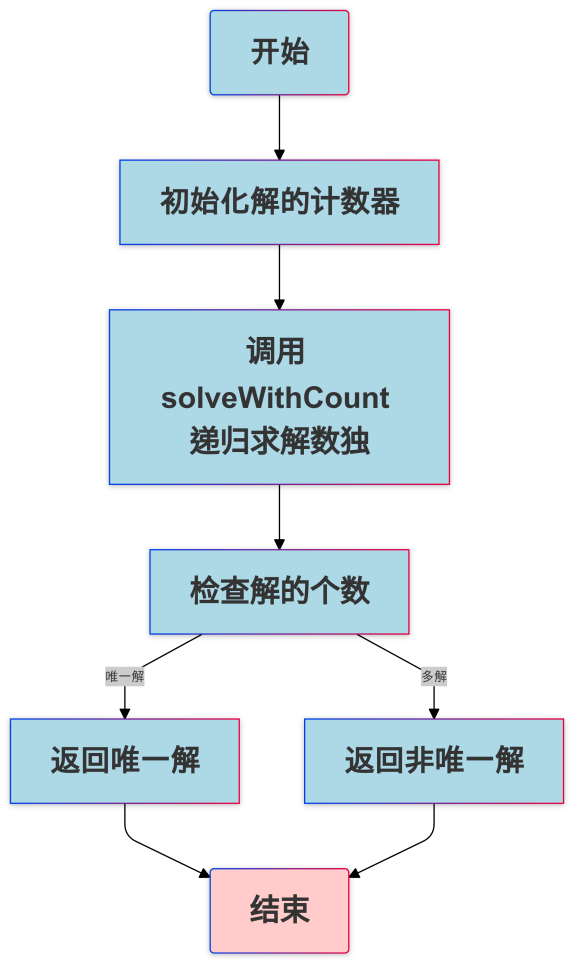
fillSudoku 函数采用递归回溯的方式填充整个数独。每次尝试将一个合法的数字填入空格，递归调用自身填充下一个空格，若遇到冲突，则回溯到上一个空格重新尝试其他数字，直到整个数独被填满。



#### 图3-11 fillSudoku 函数实现

#### 核心函数：isUniqueSolution

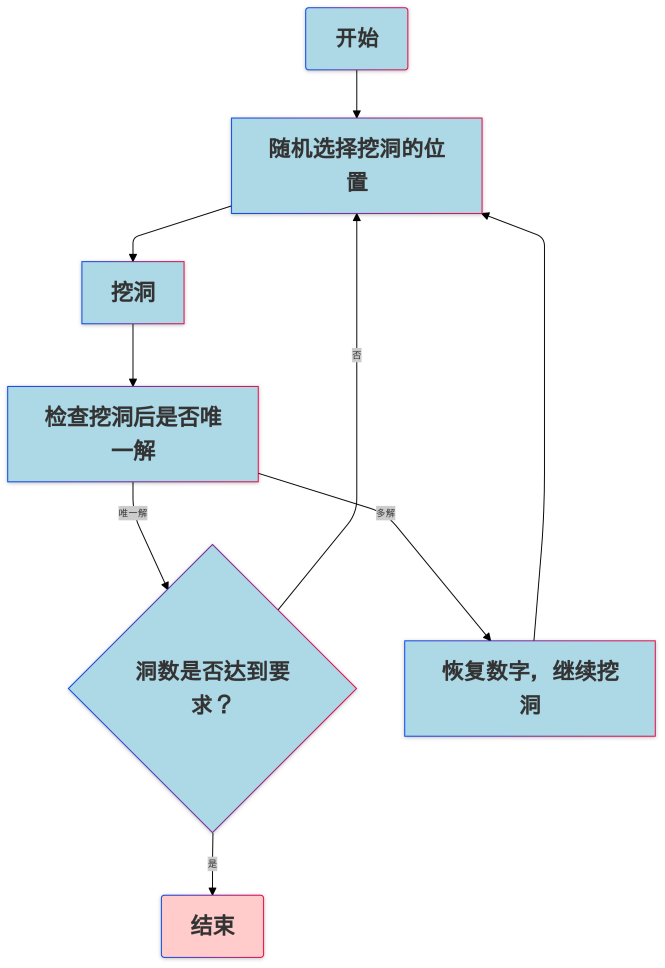
isUniqueSolution 函数通过递归回溯判断当前生成的数独是否有唯一解。该函数尝试填充数独并计数解的个数，若解的个数超过 1，则表示存在多解，需要重新生成。



#### 图3-12 isUniqueSolution函数实现

#### 挖洞算法

1. createStartingGrid 函数根据设定的难度参数，随机选择一定数量的格子挖去其内容，形成数独题目。
2. 挖洞过程中，调用 isUniqueSolution 验证挖洞后的数独是否依然满足唯一解的要求。若某个位置的挖洞破坏了唯一解，则恢复该位置的数值。
3. 持续挖洞直到达到设定的挖洞数量。



#### 图3-13挖洞算法逻辑

#### 数独求解算法

1. sudoku\_solver.c 中的 DpllSolver 函数采用 DPLL 算法求解数独的 SAT 问题表示。
2. 数独被转换为 CNF 形式后，调用 DpllSolver 进行求解。求解成功时，解码并还原数独解，求解失败时，表示该数独无解。
3. 最终结果通过文件 solution.cnf 和 natural\_solution.cnf 输出。

**4系统实现与测试**

**4.1系统实现**

系统实现基于C语言开发，并且在Windows操作系统上进行调试和运行。硬件环境包括处理器、8GB以上内存和存储设备。系统通过模块化设计，包含四大主要模块：主界面模块、CNF 解析器模块、求解器模块和对角线数独模块。

主界面模块负责实现用户界面的展示和功能跳转。通过调用对应的函数实现不同功能的界面切换。CNF 解析器模块实现对 CNF 文件的读取、存储以及结果文件的输出。求解器模块基于 DPLL 算法实现 SAT 问题的求解，主要操作包括对子句的操作（如删除、简化）和变元选择策略的优化。对角线数独模块则通过挖洞算法生成数独游戏，同时能够将数独问题转换为 SAT 问题，进行求解与校验。

**4.2****系统测试**

**4.2.1 SAT求解器模块测试**

（1）本模块主要实现基于DPLL算法实现SAT求解器，即对读取的cnf文件正确求解其可满足性的功能。设计目标是可以根据不同的SAT问题选取不同的分裂变元政策，以达到优化算法时间，求解更多规模文件的目的。

（2）测试数据主要分为以下选取方向：功能测试算例（包括满足算例和不满足算例，规模较小，主要验证算法可行性），性能测试算例（包括不同特规模的满足算例和不满足算例），特殊算例（包括现有算法不能求解的算例，用以优化测试）。

（3）测试大纲如下：

函数功能测试：主要采用功能测试算例，测试模块各函数是否实现预期功能；

DPLL算法测试：主要采用功能测试算例，测试满足数据和不满足数据求解结果是否正确，对DPLL算法进行基本实现；

SAT求解性能测试：主要采用性能测试算例数据和特殊数据，测试模块对大型数据的处理能力，并作出相应优化，测试优化效果；

（4）测试运行结果：

模块各函数基本实现，DPLL算法功能基本实现，SAT求解器基本实现：

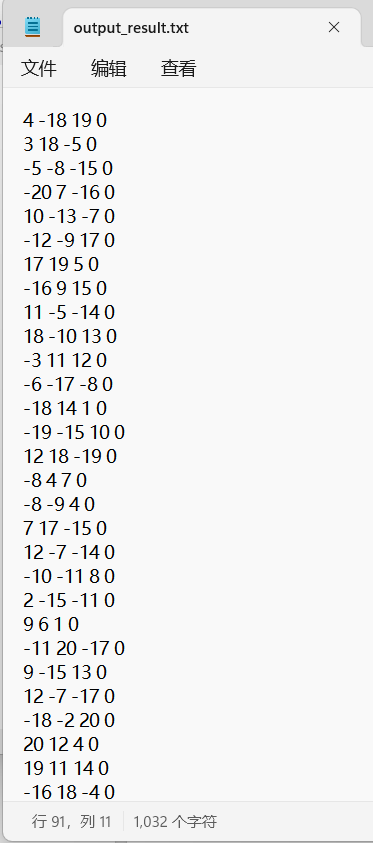


图4-1 基准满足算例读取结果图



图4-2基准满足算例求解图

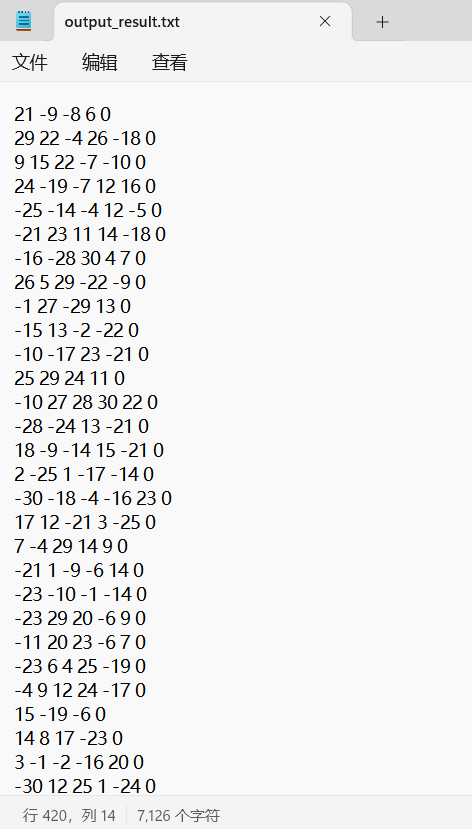


图4-3基准不满足算例读取结果图



图4-4基准不满足算例求解图

模块具有基本的性能处理能力，经优化可以有效节省算法求解时间，处理规模较大的cnf文件：



图4-5性能测试算例求解图

1. 经运行结果确认，该模块基本实现预定目标和功能，可以实现一定规模的SAT问题求解。

**4.2.2 X-Sudoku模块测试**

（1）本模块主要实现基于SAT求解器对角线数独游戏，即根据难度随机生成对应的对角线数独游戏，求解数独，检查答案、偷看答案和查看答案等功能。设计目标是可以根据用户需求生成对应难度的对角线数独游戏，为用户求解数独提供一定的交互性，游戏具有趣味性。

（2）测试数据主要分为以下选取方向：选取不同难度的对角线数独，分别求解正确，错误，系统自身生成的答案，验证对角线数独的各项功能。

（3）测试大纲如下：

选择简单难度五次，通过观察生成的不同数独格局检验随机性；再分别选择基础、中级和专家难度各一次，观察是否能够通过难度控制挖洞数目；最后完整游玩中级难度的对角线数独题目，检验”检查答案“、”偷看答案“和”查看答案“等等一次游玩过程中的全部功能。

（4）测试运行结果：

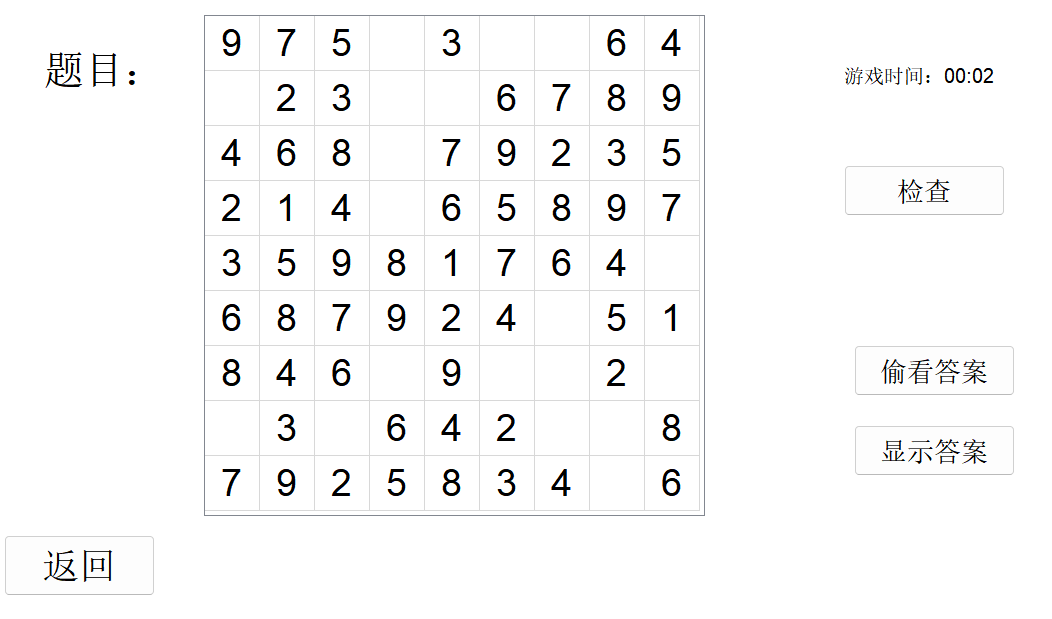


图4-6简单难度格局一

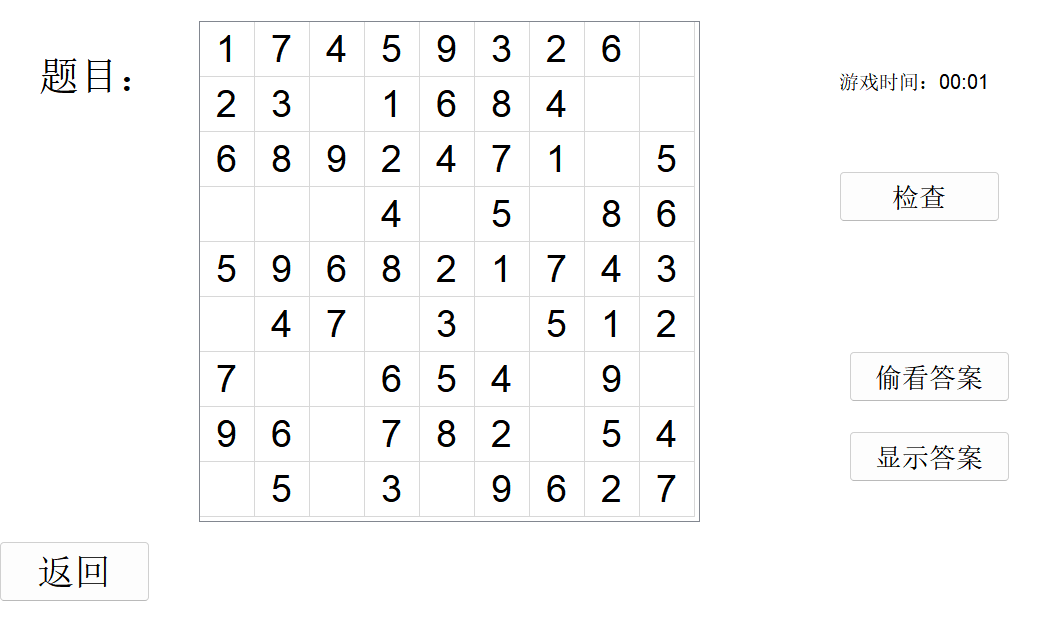


图4-7简单难度格局二

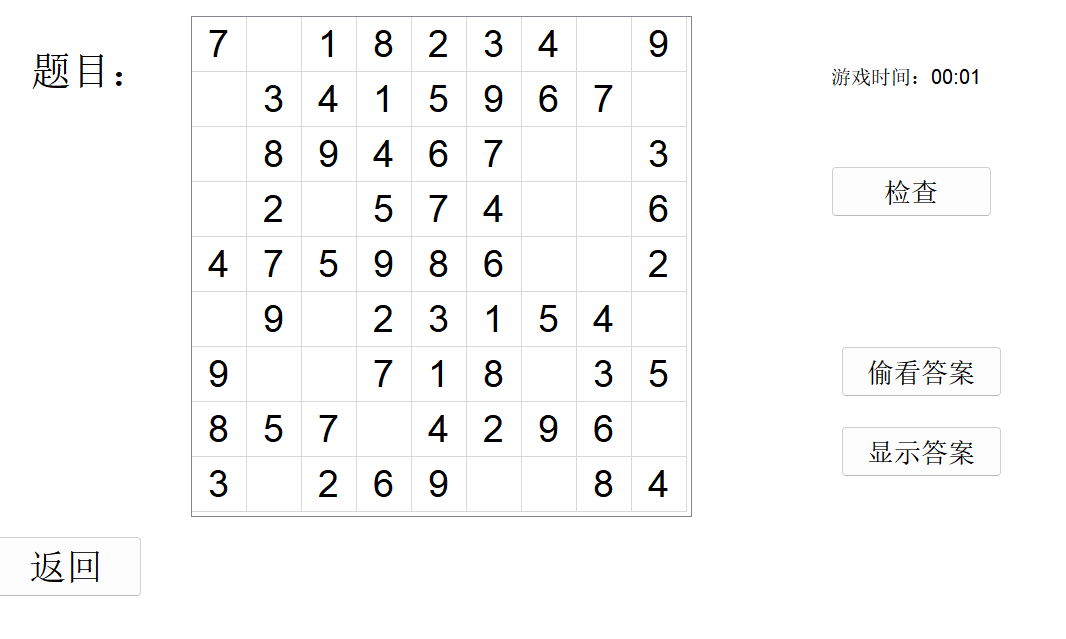


图4-8简单难度格局三

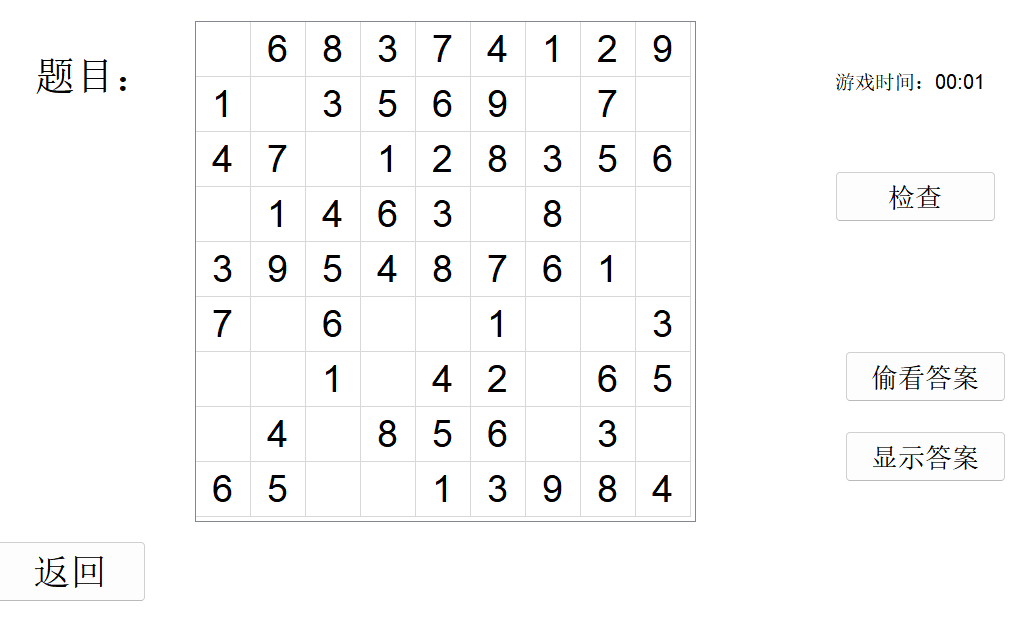


图4-9简单难度格局四

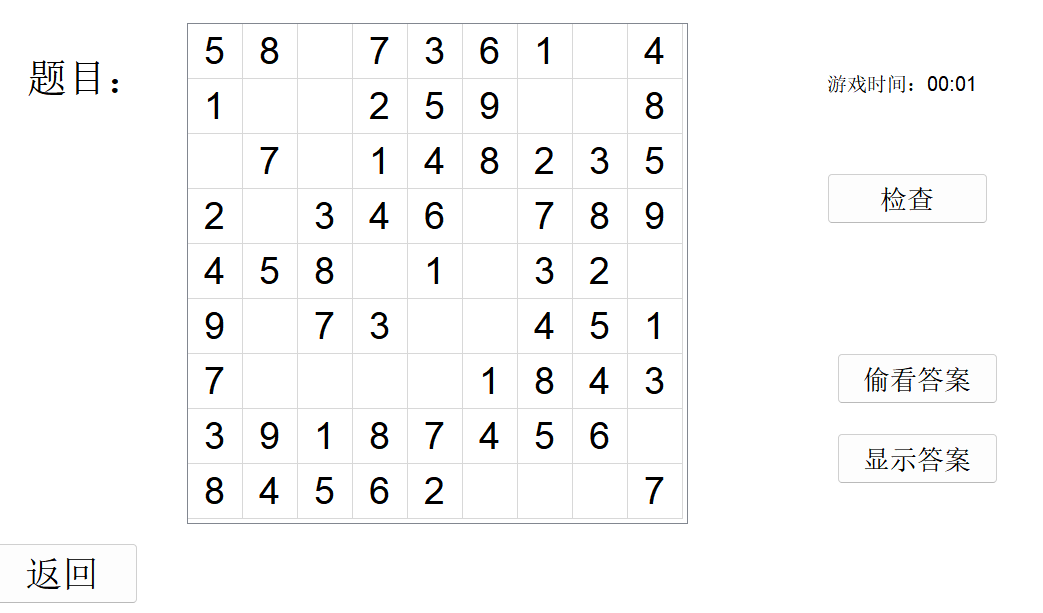


图4-10 简单难度格局五

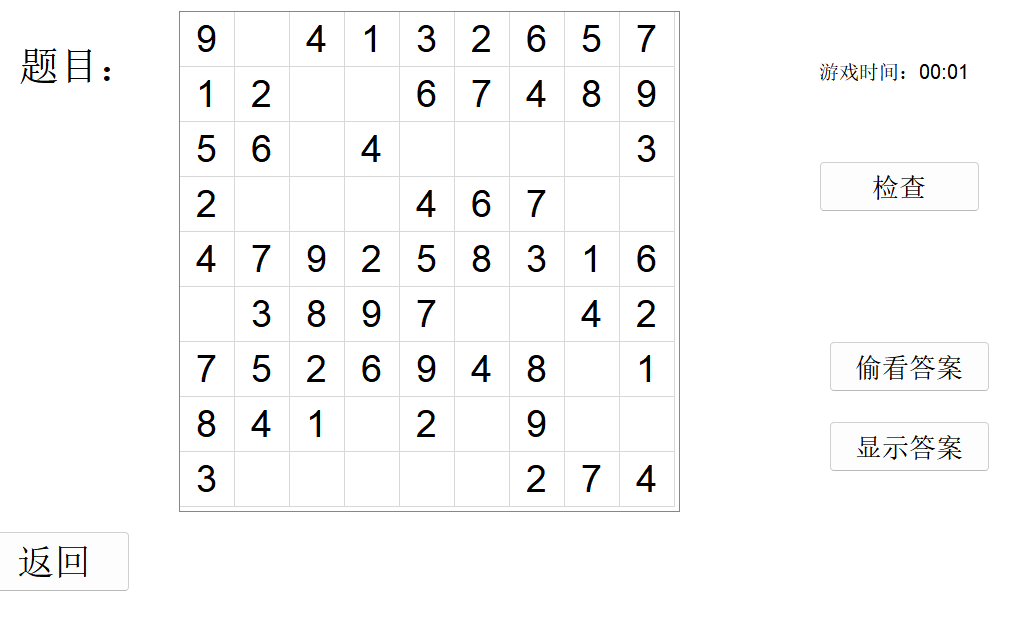


图4-11基础数独格局

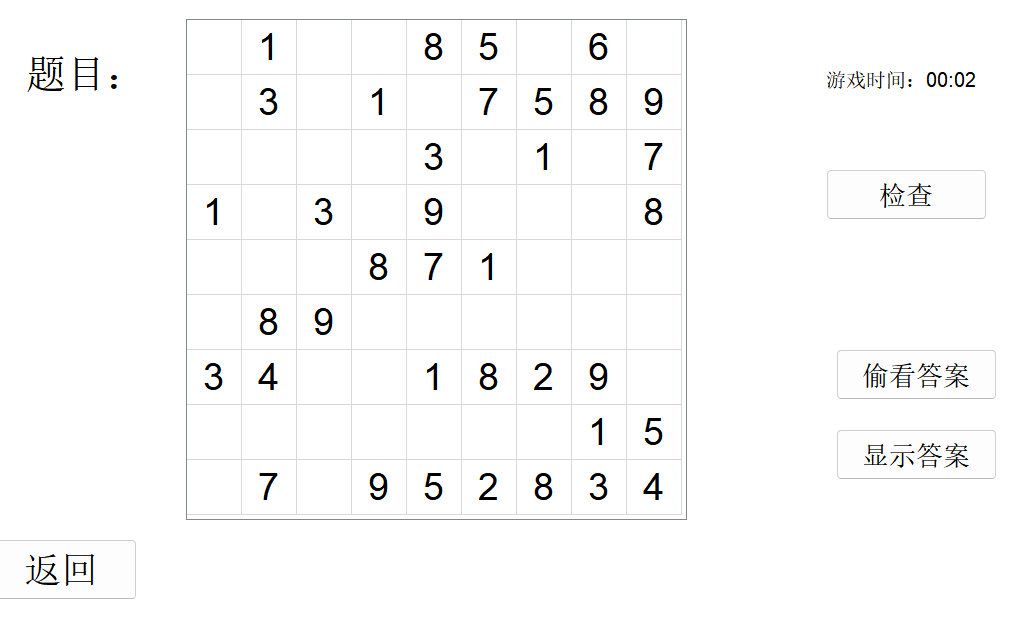


图4-12 中级数独格局

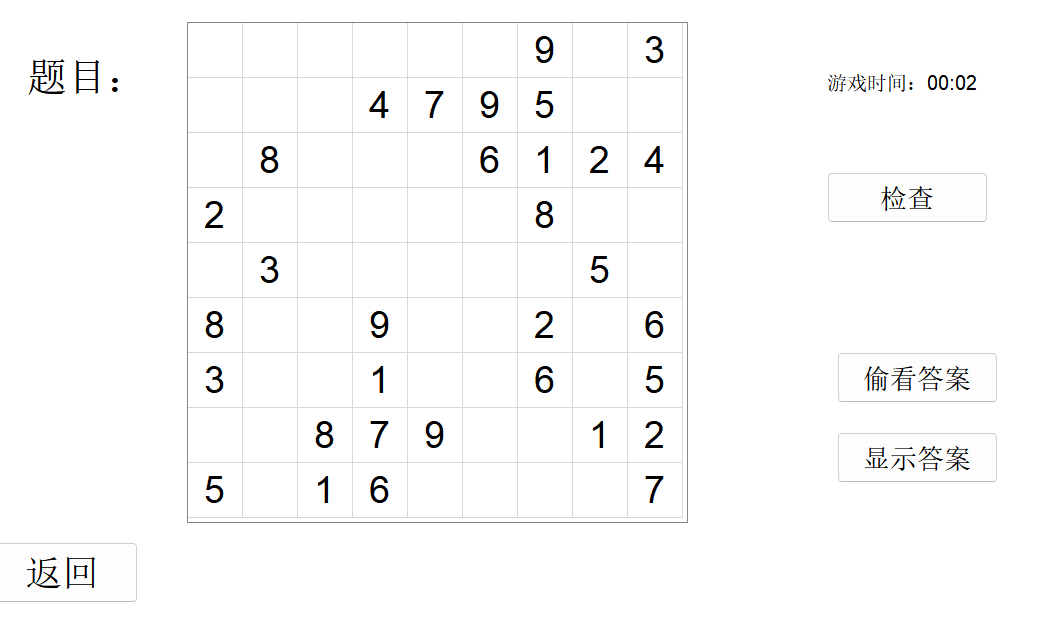


图4-13 专家数独格局

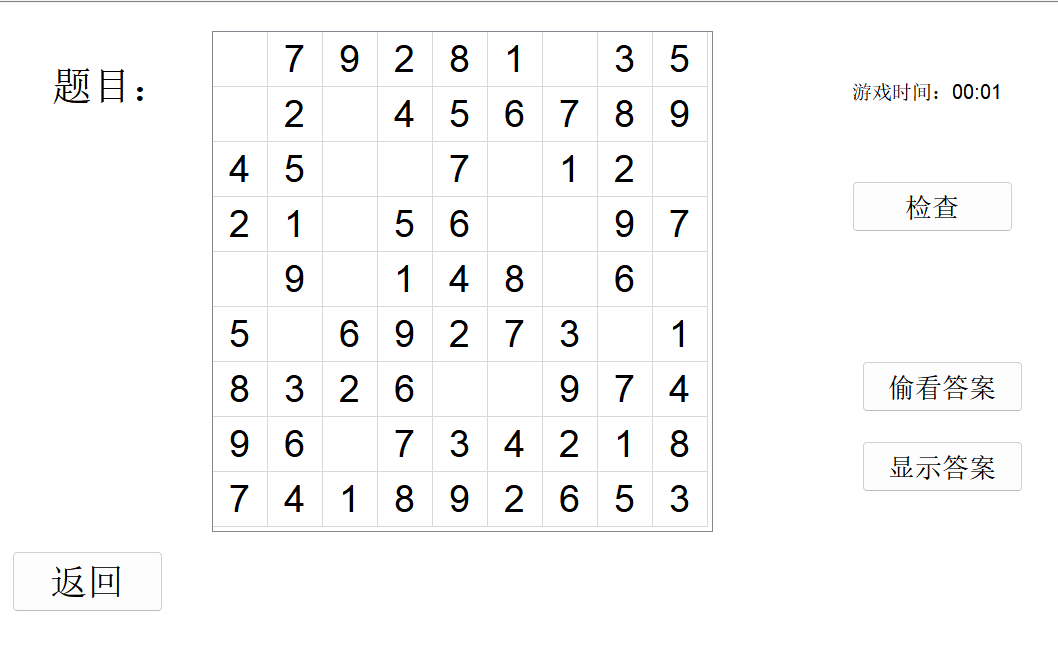


图4-14 生成简单数独格局

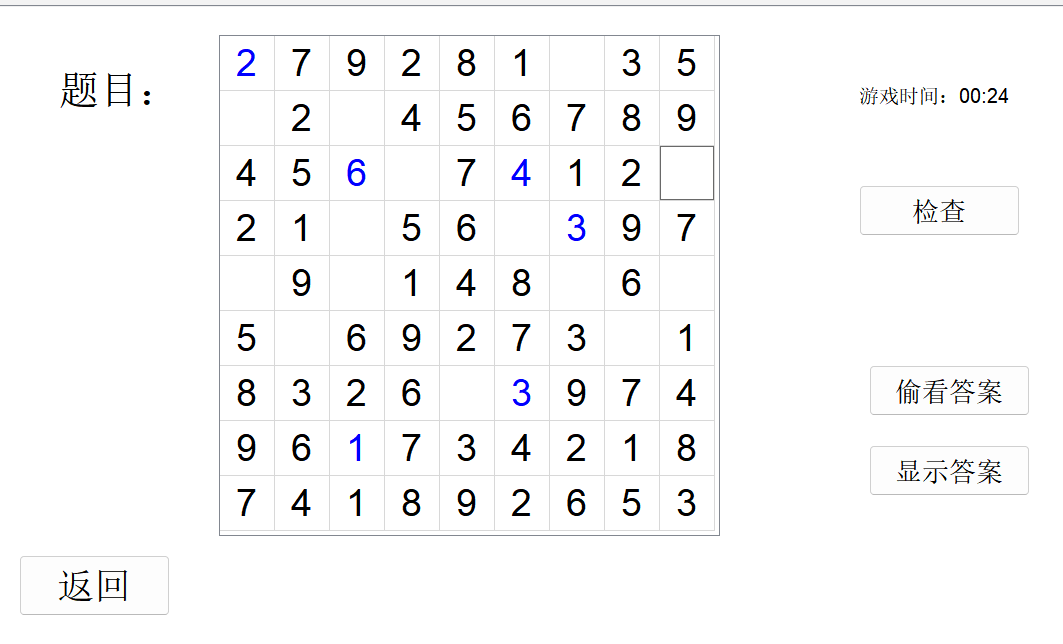


图4-15 填写功能测试

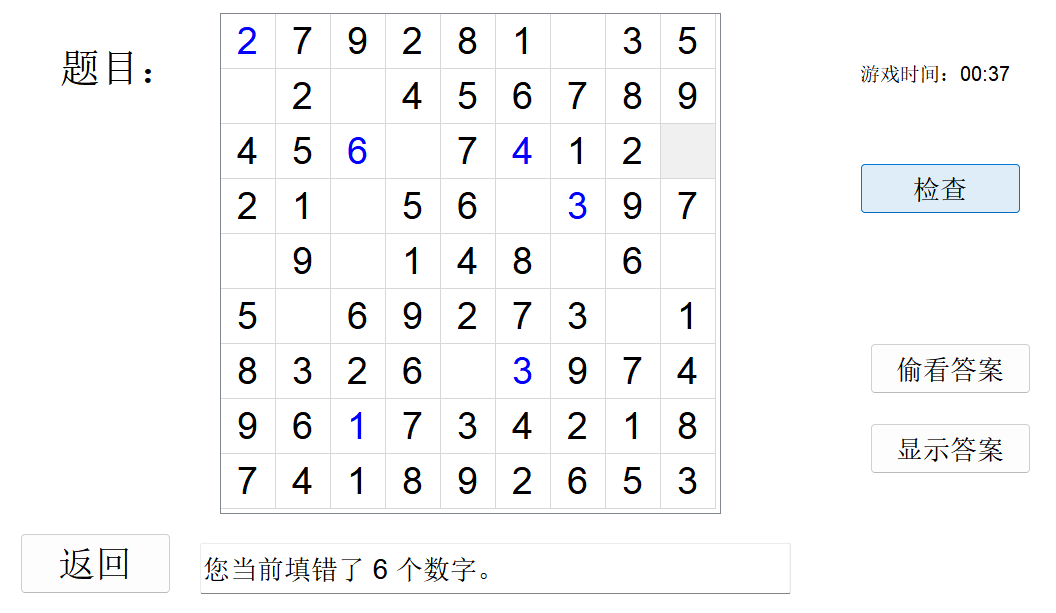


图4-16 检查功能测试

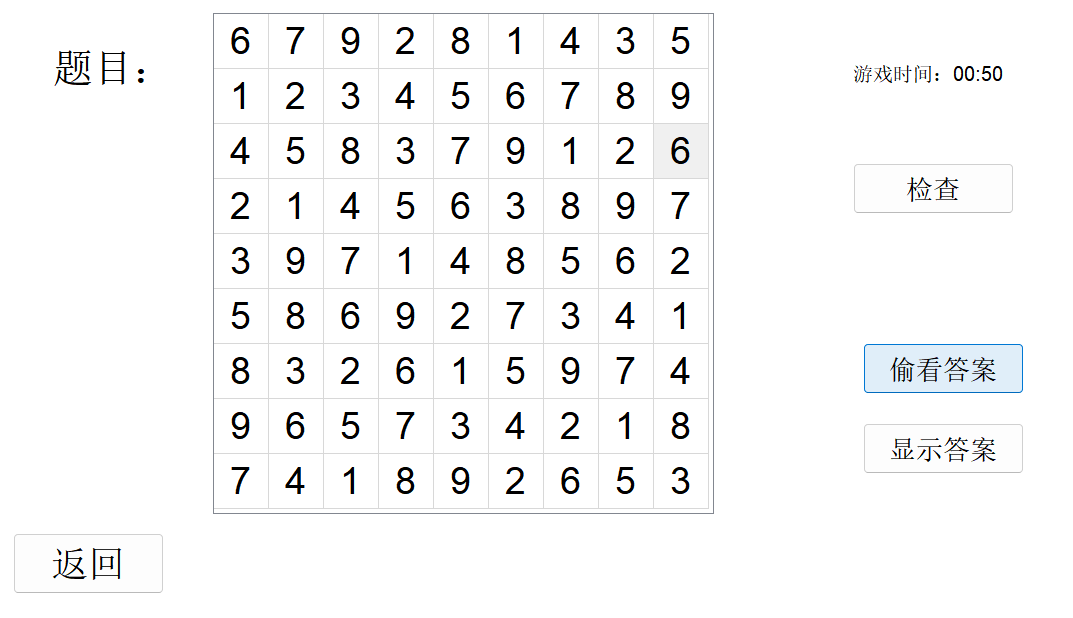


图4-17 偷看功能测试

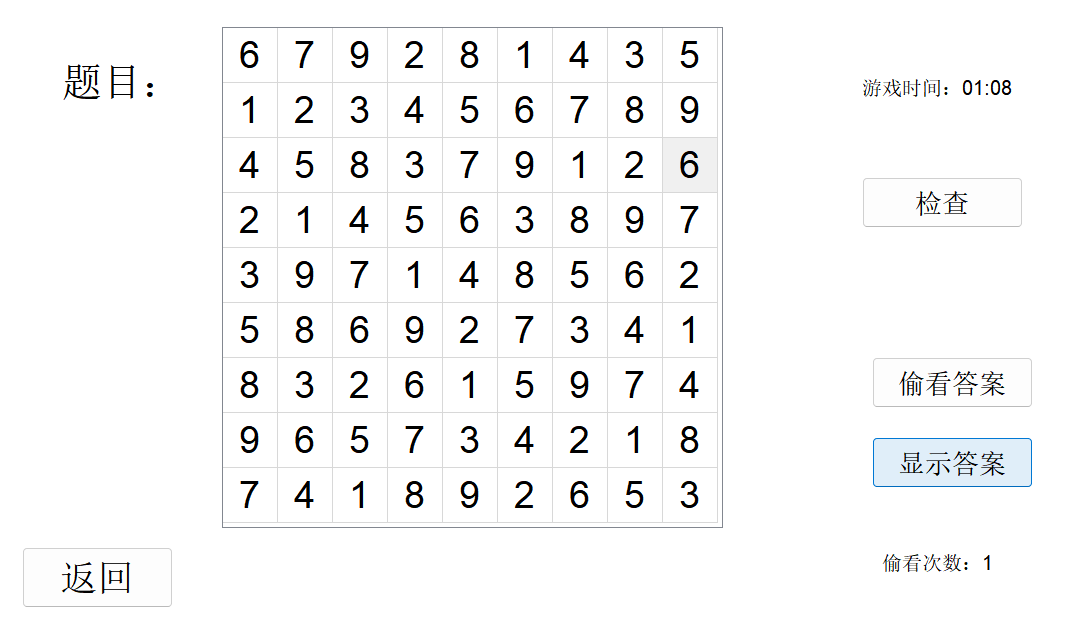


图4-18 查看答案功能测试

（5）经运行结果确认，该模块基本实现预定目标和功能，具有一定界面交互性和游玩趣味性。

**5 总结与展望**

**5.1****全文总结**

我一共花费了十天的时间完成本次课程设计。我按照时间顺序分别完成的工作如下：

1. 研读任务书，查阅相关论文资料，了解CNF文件的存储结构、DPLL算法的主要思想和思路、DPLL的优化方案思路、挖洞法生成数独题目的方法等等。
2. 配置PyQt5 Qt Designer所需环境（此过程我在miniconda中的虚拟环境实现），并简单设计页面，实现页面之间的跳转。

（3）实现了存储数据结构，能够对cnf文件进行存储；

（4）制作SAT求解器，逐步实现DPLL算法相关的函数，并选取算例进行测试。

（5）寻找优化算法的方法，此过程中我尝试过写CDCL算法，但是因为其过于复杂而放弃，最终选择仅从变元选择方面进行优化。

（6）编写数独格局生成程序、cnf与数独格局转换程序等等，实现对角线数独游戏的基本功能。

（7）测试整个程序确认功能全部实现之后，使用pyinstaller对所有涉及到的代码文件和程序进行打包，生成X-sudoku.exe文件。

**5.1工作展望**

在今后的学习和工作中，我将主要围绕着如下几个方面进行改进:

（1）通过实践进一步加深自己对数据结构与算法部分知识的理解和应用水平。

（2）学习更多界面设计相关知识，使得界面外观更加生动形象、操作更加用户友好；

（3）通过不断的项目实践，提高自己编写工程代码的能力。

# 6 体会

这一次课程设计在刚开学的两周内进行，时间较为紧张，实话说给我带来的压力还是挺大的。不过我在处理每一步的过程中都有诸多的收获，所以我觉得确实对我来说是一个非常好的提高自身代码水平和实践经验的经历。

受到周围一部分同学的影响，这一次我决定尝试用图形化界面来展现我程序的功能实现，这一个过程还是比较艰难。我在网上按照教程一步一步实现了虚拟环境的搭建，并在cmd中实现了相关文件的下载和配置。这个过程由于python版本的兼容问题，进行了很多次，耗费了大量时间，还是有一点折磨。这一点提醒了我，以后下载文件的时候一定要注意文件之间的版本兼容问题。

在剩下的代码实现过程中，我一般是先把一个功能实现了之后，和界面链接，之后测试看是否能在界面上得到预期结果。这样做比较直观，但是给一些Bug的排除带来了困难。尤其是我在实现数独题目的生成与求解的过程中，出现了求解之后题目数字都会改变的窘况，我反复排查了很多次，检查过界面代码、数独求解代码、cnf文件和数独转换代码、数独题目生成代码等等，甚至怀疑过很多次自己在自然编码和语义编码的转换的过程中写的公式不对，很长时间没有进展。最终在生成的对角线数独规则cnf文件中发现行数过少，从而发现是约束条件不足导致的。所以说很多时候应该逐个文件进行调试功能才比较好。

总的来说，我对自己最终整个程序的实现还是比较满意的。这一次课设的实践机会使得我对图形化界面设计、数据结构与算法的部分知识与经验得到了显著的提升。今后应该也会规避在这一次课设中犯过的一些错误，节省时间。

**参考文献**

[1] 华中科技大学综合课程设计，华中科技大学计算机科学与技术学院程序设计综合课程设计课程组，2024年5月

[2] 陈稳. 基于DPLL的SAT算法的研究与应用.硕士学位论文，电子科技大学，2011

[3] 360百科：数独游戏<https://baike.so.com/doc/3390505-3569059.html>

Twodoku： https://en.grandgames.net/multisudoku/twodoku

[4] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[5] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191

[6] QT Designer +VS Code+PyQt5 :https://blog.csdn.net/weixin\_45676151/article/details/119117035

**附录**

**1.main.py (调控所有界面）**

import sys

import subprocess

import os

from PyQt5.QtWidgets import QApplication, QMainWindow, QStackedWidget, QFileDialog, QTableWidgetItem

from PyQt5.QtCore import QTimer, Qt

from PyQt5 import QtGui, QtCore

from Ui\_start import Ui\_MainWindow as Ui\_StartWindow

from Ui\_rules import Ui\_RulesWindow

from Ui\_mode import Ui\_MainWindow as Ui\_ModeWindow

from Ui\_upload import Ui\_MainWindow as Ui\_UploadWindow

from Ui\_difficulty import Ui\_MainWindow as Ui\_DifficultyWindow

from Ui\_game import Ui\_MainWindow as Ui\_GameWindow

class MainApp(QMainWindow):

def \_\_init\_\_(self):

super(MainApp, self).\_\_init\_\_()

self.stacked\_widget = QStackedWidget()

self.setCentralWidget(self.stacked\_widget)

# 动态获取C程序的路径

if hasattr(sys, '\_MEIPASS'):

self.c\_programs\_dir = os.path.join(sys.\_MEIPASS, 'c')

else:

self.c\_programs\_dir = os.path.join(os.path.dirname(os.path.abspath(\_\_file\_\_)), '../c')

# 创建并添加各个界面到 QStackedWidget

self.start\_ui = Ui\_StartWindow()

self.rules\_ui = Ui\_RulesWindow()

self.mode\_ui = Ui\_ModeWindow()

self.upload\_ui = Ui\_UploadWindow()

self.difficulty\_ui = Ui\_DifficultyWindow()

self.game\_ui = Ui\_GameWindow()

# 加载界面

self.start\_widget = QMainWindow()

self.start\_ui.setupUi(self.start\_widget)

self.stacked\_widget.addWidget(self.start\_widget)

self.rules\_widget = QMainWindow()

self.rules\_ui.setupUi(self.rules\_widget)

self.stacked\_widget.addWidget(self.rules\_widget)

self.mode\_widget = QMainWindow()

self.mode\_ui.setupUi(self.mode\_widget)

self.stacked\_widget.addWidget(self.mode\_widget)

self.upload\_widget = QMainWindow()

self.upload\_ui.setupUi(self.upload\_widget)

self.stacked\_widget.addWidget(self.upload\_widget)

self.difficulty\_widget = QMainWindow()

self.difficulty\_ui.setupUi(self.difficulty\_widget)

self.stacked\_widget.addWidget(self.difficulty\_widget)

self.game\_widget = QMainWindow()

self.game\_ui.setupUi(self.game\_widget)

self.stacked\_widget.addWidget(self.game\_widget)

# 绑定按钮事件

self.start\_ui.rules.clicked.connect(self.show\_rules)

self.start\_ui.game.clicked.connect(self.show\_mode)

self.rules\_ui.rules\_back.clicked.connect(self.show\_main)

self.mode\_ui.upload.clicked.connect(self.show\_upload)

self.mode\_ui.puzzle.clicked.connect(self.show\_difficulty)

self.mode\_ui.mode\_back.clicked.connect(self.show\_main)

self.difficulty\_ui.difficulty\_back.clicked.connect(self.show\_mode)

self.upload\_ui.upload\_back.clicked.connect(self.show\_mode)

self.upload\_ui.file\_choose.clicked.connect(self.on\_file\_choose\_clicked)

self.upload\_ui.solve\_button.clicked.connect(self.on\_solve\_button\_clicked)

self.upload\_ui.better\_solve\_button\_1.clicked.connect(lambda: self.on\_better\_solve\_button\_clicked(1))

self.upload\_ui.better\_solve\_button\_2.clicked.connect(lambda: self.on\_better\_solve\_button\_clicked(2))

self.upload\_ui.better\_solve\_button\_3.clicked.connect(lambda: self.on\_better\_solve\_button\_clicked(3))

self.upload\_ui.source\_button.clicked.connect(self.open\_source\_file)

self.upload\_ui.save\_button.clicked.connect(self.open\_saved\_file)

self.upload\_ui.dpll\_button.clicked.connect(self.open\_dpll\_file)

self.upload\_ui.cdcl\_button.clicked.connect(self.open\_cdcl\_file)

# 绑定难度按钮，传递不同的参数来生成对角线数独题目

self.difficulty\_ui.easy.clicked.connect(lambda: self.prepare\_game("easy"))

self.difficulty\_ui.simple.clicked.connect(lambda: self.prepare\_game("simple"))

self.difficulty\_ui.intermediate.clicked.connect(lambda: self.prepare\_game("intermediate"))

self.difficulty\_ui.expert.clicked.connect(lambda: self.prepare\_game("expert"))

self.game\_ui.game\_back.clicked.connect(self.show\_difficulty)

self.game\_ui.check\_button.clicked.connect(self.check\_solution)

self.game\_ui.peek\_button.clicked.connect(self.peek\_solution)

self.game\_ui.show\_button.clicked.connect(self.show\_solution)

self.timer = QTimer()

self.timer.timeout.connect(self.update\_game\_time)

self.peek\_count = 0

self.game\_time\_seconds = 0

self.is\_answer\_shown = False

self.answer\_file\_path = "solution.txt"

self.previous\_user\_input = [] # 用于存储用户输入的状态

def get\_selected\_difficulty(self):

"""根据用户选择返回不同难度级别"""

if self.difficulty\_ui.easy.isChecked():

return "easy"

elif self.difficulty\_ui.simple.isChecked():

return "simple"

elif self.difficulty\_ui.intermediate.isChecked():

return "intermediate"

elif self.difficulty\_ui.expert.isChecked():

return "expert"

return "easy"

def prepare\_game(self, difficulty):

"""根据选择的难度准备游戏"""

self.timer.start(1000)

self.peek\_count = 0

self.game\_time\_seconds = 0

self.is\_answer\_shown = False

self.game\_ui.peek\_count.setVisible(False)

self.game\_ui.game\_time.setVisible(True)

# 确定挖洞数量

difficulty\_levels = {

"easy": "1", # 假设"easy"对应20个挖洞

"simple": "2", # "simple" 对应30个挖洞

"intermediate": "3", # "intermediate" 对应40个挖洞

"expert": "4" # "expert" 对应50个挖洞

}

# 获取数值型的难度级别

difficulty\_level = difficulty\_levels.get(difficulty, "1")

# 调用C程序生成对角线数独题目

semantic\_file\_path = "generated\_puzzle.cnf" # 生成的语义编码CNF文件路径

natural\_file\_path = "altered\_puzzle.cnf" # 生成的自然编码CNF文件路径

generator\_path = os.path.join(self.c\_programs\_dir, 'generate\_diagonal\_sudoku.exe')

solver\_path = os.path.join(self.c\_programs\_dir, 'sudoku\_solver.exe')

try:

result = subprocess.run([generator\_path, difficulty\_level, semantic\_file\_path, natural\_file\_path], check=True, capture\_output=True, text=True, encoding='utf-8')

print(f"Puzzle generation completed, CNF files saved to: {semantic\_file\_path} and {natural\_file\_path}")

print(f"C program output: {result.stdout}")

# 运行解算器

result = subprocess.run([solver\_path, natural\_file\_path], check=True, capture\_output=True, text=True, encoding='utf-8')

print("Solution file generated.")

print(f"Solver output: {result.stdout}")

# 转换解文件为数独网格格式

if not self.convert\_solution\_to\_grid("solution.cnf"):

print("Error converting solution to grid format.")

return

except subprocess.CalledProcessError as e:

print(f"Error generating puzzle or solution: {e}")

print(f"C program error output: {e.stderr}")

return

except Exception as e:

print(f"Other error: {e}")

return

# 清空游戏界面并显示新生成的数独

self.show\_game()

self.display\_puzzle(semantic\_file\_path)

def convert\_solution\_to\_grid(self,solution\_cnf\_path):

"""

使用 cnf\_to\_grid.exe 将 CNF 格式的解转换为数独网格格式。

"""

converter\_path = os.path.join(self.c\_programs\_dir, 'cnf\_to\_grid.exe')

grid\_output\_path = "solution.txt" # 转换后的数独网格路径

try:

# 调用转换器程序

result = subprocess.run([converter\_path, solution\_cnf\_path, grid\_output\_path], check=True, capture\_output=True, text=True, encoding='utf-8')

print(f"Solution converted and saved to: {grid\_output\_path}")

print(f"C program output: {result.stdout}")

except subprocess.CalledProcessError as e:

print(f"Error converting solution: {e}")

print(f"C program error output: {e.stderr}")

return False

except Exception as e:

print(f"Other error: {e}")

return False

return True

def display\_puzzle(self, cnf\_file\_path):

"""将CNF格式的数独题目转换为可显示的数独格局"""

parser\_path = os.path.join(self.c\_programs\_dir, 'cnf\_to\_grid.exe')

puzzle\_file\_path = "sudoku\_puzzle.txt" # 生成的数独格局路径

try:

result = subprocess.run([parser\_path, cnf\_file\_path, puzzle\_file\_path], check=True, capture\_output=True, text=True, encoding='utf-8')

print(f"Puzzle parsed and grid generated, saved to: {puzzle\_file\_path}")

print(f"C program output: {result.stdout}")

except subprocess.CalledProcessError as e:

print(f"Parsing error: {e}")

print(f"C program error output: {e.stderr}")

return

except Exception as e:

print(f"Other error: {e}")

return

# 在游戏界面显示生成的数独网格

self.load\_puzzle\_to\_ui(puzzle\_file\_path)

def load\_puzzle\_to\_ui(self, puzzle\_file\_path):

"""加载解析后的数独网格并显示在游戏界面上"""

try:

with open(puzzle\_file\_path, 'r') as f:

puzzle\_lines = f.readlines()

if not puzzle\_lines:

print("Error: Puzzle file is empty or not properly formatted.")

return

print(f"Loaded puzzle from file: {puzzle\_file\_path}")

table = self.game\_ui.sudoku\_table

table.setRowCount(9)

table.setColumnCount(9)

for row\_idx, line in enumerate(puzzle\_lines):

numbers = line.strip().split()

for col\_idx, num in enumerate(numbers):

item = QTableWidgetItem("" if num == '0' else num)

item.setTextAlignment(Qt.AlignCenter)

if num != '0': # 0 表示空格

item.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled) # 禁止修改

item.setForeground(QtGui.QBrush(Qt.black))

else:

# 允许输入

item.setFlags(Qt.ItemIsSelectable | Qt.ItemIsEnabled | Qt.ItemIsEditable)

item.setForeground(QtGui.QBrush(Qt.blue))

table.setItem(row\_idx, col\_idx, item)

# 强制刷新 UI

table.viewport().update()

table.repaint()

# 调用 Qt 事件循环来确保 UI 刷新

QtGui.QGuiApplication.processEvents()

print("Successfully loaded puzzle into the UI.") # 添加调试信息

except Exception as e:

print(f"Error loading puzzle to UI: {e}")

def on\_file\_choose\_clicked(self):

# 每次选择文件前，重置UI状态

self.upload\_ui.judge.clear()

self.upload\_ui.saved\_path.clear()

self.upload\_ui.optimization\_rate.clear()

self.upload\_ui.dpll\_time.clear()

self.upload\_ui.dpll\_address.clear()

self.upload\_ui.cdcl\_time.clear()

self.upload\_ui.cdcl\_address.clear()

self.upload\_ui.judge\_label.hide()

self.upload\_ui.judge.hide()

self.upload\_ui.saved\_path\_label.hide()

self.upload\_ui.saved\_path.hide()

self.upload\_ui.rate\_label.hide()

self.upload\_ui.optimization\_rate.hide()

self.upload\_ui.dpll\_time\_label.hide()

self.upload\_ui.dpll\_time.hide()

self.upload\_ui.dpll\_address\_label.hide()

self.upload\_ui.dpll\_address.hide()

self.upload\_ui.cdcl\_time\_label.hide()

self.upload\_ui.cdcl\_time.hide()

self.upload\_ui.cdcl\_address\_label.hide()

self.upload\_ui.cdcl\_address.hide()

self.upload\_ui.dpll\_button.hide()

self.upload\_ui.cdcl\_button.hide()

file\_path, \_ = QFileDialog.getOpenFileName(self, "选择文件", "", "CNF Files (\*.cnf);;All Files (\*)")

if file\_path:

file\_path = os.path.abspath(file\_path)

print(f"Chosen file path: {file\_path}")

self.upload\_ui.cnf\_path.setText(file\_path)

self.upload\_ui.cnf\_path.show()

if not os.path.exists(file\_path):

print("Error: File does not exist.")

return

cnf\_parser\_path = os.path.join(self.c\_programs\_dir, 'cnf\_parser.exe')

if not os.path.exists(cnf\_parser\_path):

print(f"Error: cnf\_parser.exe not found at {cnf\_parser\_path}")

return

saved\_path = os.path.abspath("output\_result.txt")

try:

# 指定编码为 utf-8

result = subprocess.run([cnf\_parser\_path, file\_path, saved\_path], check=True, capture\_output=True, text=True, encoding='utf-8')

print(f"Parsing completed, results saved to: {saved\_path}")

print(f"C program output: {result.stdout}")

except subprocess.CalledProcessError as e:

print(f"Parsing error: {e}")

print(f"C program error output: {e.stderr}")

except Exception as e:

print(f"Other error: {e}")

self.upload\_ui.saved\_path.setText(saved\_path)

self.upload\_ui.saved\_path.show()

self.upload\_ui.saved\_path\_label.show()

self.upload\_ui.solve\_button.show()

self.upload\_ui.source\_button.show()

self.upload\_ui.save\_button.show()

def on\_solve\_button\_clicked(self):

"""执行 solver.exe """

cnf\_file\_path = self.upload\_ui.cnf\_path.text()

if not cnf\_file\_path:

print("No CNF file selected.")

return

result\_file\_path = cnf\_file\_path + ".res"

solver\_path = os.path.join(self.c\_programs\_dir, 'solver.exe')

try:

# 指定编码为 utf-8

result = subprocess.run([solver\_path, cnf\_file\_path], check=True, capture\_output=True, text=True, encoding='utf-8')

print(f"Solving completed, results saved to: {result\_file\_path}")

print(f"Solver output: {result.stdout}")

self.upload\_ui.dpll\_address.setText(result\_file\_path)

self.upload\_ui.dpll\_address.show()

# 读取生成的结果文件

if os.path.exists(result\_file\_path):

with open(result\_file\_path, 'r') as f:

lines = f.readlines()

result\_status = None

solve\_time = None

for line in lines:

if line.startswith('s'):

result\_status = line.strip()

elif line.startswith('t'):

solve\_time = line.strip().split(' ')[1]

if result\_status == 's 1':

self.upload\_ui.dpll\_time.setText(f"{solve\_time} ms")

self.upload\_ui.judge.setText("有解")

elif result\_status == 's 0':

self.upload\_ui.dpll\_time.setText(f"{solve\_time} ms")

self.upload\_ui.judge.setText("无解")

except subprocess.CalledProcessError as e:

print(f"Error running solver: {e}")

print(f"Solver error output: {e.stderr}")

except Exception as e:

print(f"Other error: {e}")

self.upload\_ui.dpll\_time\_label.show()

self.upload\_ui.dpll\_time.show()

self.upload\_ui.dpll\_address\_label.show()

self.upload\_ui.dpll\_address.show()

self.upload\_ui.dpll\_button.show()

self.upload\_ui.judge\_label.show()

self.upload\_ui.judge.show()

self.calculate\_and\_display\_optimization\_rate()

def on\_better\_solve\_button\_clicked(self, pickvar\_choice):

"""执行 better\_solver.exe 并传递参数以选择不同的 PickVar 函数"""

cnf\_file\_path = self.upload\_ui.cnf\_path.text()

if not cnf\_file\_path:

print("No CNF file selected.")

return

result\_file\_path = f"{cnf\_file\_path}\_faster.res"

solver\_path = os.path.join(self.c\_programs\_dir, 'better\_solver.exe')

try:

# 运行带参数的 better\_solver.exe

result = subprocess.run([solver\_path, cnf\_file\_path, str(pickvar\_choice)], check=True, capture\_output=True, text=True, encoding='utf-8')

print(f"Solving completed with PickVar\_{pickvar\_choice}, results saved to: {result\_file\_path}")

print(f"Solver output: {result.stdout}")

self.upload\_ui.cdcl\_address.setText(result\_file\_path)

self.upload\_ui.cdcl\_address.show()

# 读取生成的结果文件

if os.path.exists(result\_file\_path):

with open(result\_file\_path, 'r') as f:

lines = f.readlines()

result\_status = None

solve\_time = None

for line in lines:

if line.startswith('s'):

result\_status = line.strip()

elif line.startswith('t'):

solve\_time = line.strip().split(' ')[1]

if result\_status == 's 1':

self.upload\_ui.cdcl\_time.setText(f"{solve\_time} ms")

self.upload\_ui.judge.setText("有解")

elif result\_status == 's 0':

self.upload\_ui.cdcl\_time.setText(f"{solve\_time} ms")

self.upload\_ui.judge.setText("无解")

except subprocess.CalledProcessError as e:

print(f"Error running solver: {e}")

print(f"Solver error output: {e.stderr}")

except Exception as e:

print(f"Other error: {e}")

self.upload\_ui.cdcl\_time\_label.show()

self.upload\_ui.cdcl\_time.show()

self.upload\_ui.cdcl\_address\_label.show()

self.upload\_ui.cdcl\_address.show()

self.upload\_ui.cdcl\_button.show()

self.upload\_ui.judge\_label.show()

self.upload\_ui.judge.show()

self.calculate\_and\_display\_optimization\_rate()

def calculate\_and\_display\_optimization\_rate(self):

"""计算优化率并显示"""

dpll\_time\_text = self.upload\_ui.dpll\_time.text().replace(" ms", "")

cdcl\_time\_text = self.upload\_ui.cdcl\_time.text().replace(" ms", "")

# 检查 dpll\_time 和 cdcl\_time 是否都有值

if cdcl\_time\_text and (not dpll\_time\_text or dpll\_time\_text == "(未实现)"):

# 如果 dpll\_time 没有值或为初始状态，且 cdcl\_time 有值，显示为无限大

self.upload\_ui.optimization\_rate.setText("∞")

self.upload\_ui.rate\_label.show()

self.upload\_ui.optimization\_rate.show()

elif dpll\_time\_text and cdcl\_time\_text:

# 如果 dpll\_time 和 cdcl\_time 都有值，则计算优化率

dpll\_time = float(dpll\_time\_text)

cdcl\_time = float(cdcl\_time\_text)

optimization\_rate = ((dpll\_time - cdcl\_time) / dpll\_time) \* 100

self.upload\_ui.optimization\_rate.setText(f"{optimization\_rate:.2f}%")

self.upload\_ui.rate\_label.show()

self.upload\_ui.optimization\_rate.show()

def open\_source\_file(self):

file\_path = self.upload\_ui.cnf\_path.text()

if file\_path:

QtGui.QDesktopServices.openUrl(QtCore.QUrl.fromLocalFile(file\_path))

def open\_saved\_file(self):

saved\_path = self.upload\_ui.saved\_path.text()

if saved\_path:

QtGui.QDesktopServices.openUrl(QtCore.QUrl.fromLocalFile(saved\_path))

def open\_dpll\_file(self):

dpll\_path = self.upload\_ui.dpll\_address.text()

if dpll\_path:

QtGui.QDesktopServices.openUrl(QtCore.QUrl.fromLocalFile(dpll\_path))

def open\_cdcl\_file(self):

cdcl\_path = self.upload\_ui.cdcl\_address.text()

if cdcl\_path:

QtGui.QDesktopServices.openUrl(QtCore.QUrl.fromLocalFile(cdcl\_path))

def check\_solution(self):

"""检查用户的答案是否正确"""

if not os.path.exists(self.answer\_file\_path):

print("Error: Solution file not found.")

return

correct\_grid = []

with open(self.answer\_file\_path, 'r') as f:

correct\_grid = [line.strip().split() for line in f.readlines()]

table = self.game\_ui.sudoku\_table

wrong\_count = 0

for row in range(table.rowCount()):

for col in range(table.columnCount()):

item = table.item(row, col)

if item and item.text().isdigit():

if item.text() != correct\_grid[row][col]:

wrong\_count += 1

if wrong\_count == 0:

self.game\_ui.check\_information.setText("填写全部正确！")

else:

self.game\_ui.check\_information.setText(f"您当前填错了 {wrong\_count} 个数字。")

self.game\_ui.check\_information.setVisible(True)

QTimer.singleShot(5000, lambda: self.game\_ui.check\_information.setVisible(False))

def peek\_solution(self):

"""偷看答案"""

table = self.game\_ui.sudoku\_table # 确保 table 在函数开始时定义

if not self.is\_answer\_shown:

self.peek\_count += 1

if not os.path.exists(self.answer\_file\_path):

print("Error: Solution file not found.")

return

# 清空并保存用户当前输入的状态

self.previous\_user\_input = []

for row in range(table.rowCount()):

row\_data = []

for col in range(table.columnCount()):

item = table.item(row, col)

row\_data.append(item.text() if item else "")

self.previous\_user\_input.append(row\_data)

# 显示答案

with open(self.answer\_file\_path, 'r') as f:

answer\_lines = [line.strip().split() for line in f.readlines()]

for row in range(table.rowCount()):

for col in range(table.columnCount()):

item = table.item(row, col)

if item and item.flags() & Qt.ItemIsEditable: # 如果是可编辑单元格，则显示答案

item.setText(answer\_lines[row][col])

item.setForeground(QtGui.QBrush(Qt.black))

QTimer.singleShot(5000, lambda: self.hide\_peek())

def hide\_peek(self):

"""隐藏偷看的答案"""

table = self.game\_ui.sudoku\_table

for row in range(table.rowCount()):

for col in range(table.columnCount()):

item = table.item(row, col)

if item and item.flags() & Qt.ItemIsEditable: # 如果是可编辑单元格，则隐藏

# 恢复用户之前输入的内容

item.setText(self.previous\_user\_input[row][col] if self.previous\_user\_input else "")

item.setForeground(QtGui.QBrush(Qt.blue)) # 恢复为蓝色字体

def show\_solution(self):

"""显示完整的答案"""

if not os.path.exists(self.answer\_file\_path):

print("Error: Solution file not found.")

return

with open(self.answer\_file\_path, 'r') as f:

answer\_lines = [line.strip().split() for line in f.readlines()]

table = self.game\_ui.sudoku\_table

for row in range(table.rowCount()):

for col in range(table.columnCount()):

item = table.item(row, col)

if item:

item.setText(answer\_lines[row][col])

item.setForeground(QtGui.QBrush(Qt.black))

# 停止计时但显示时间

self.timer.stop()

self.game\_ui.game\_time.setVisible(True) # 确保游戏时间显示

self.game\_ui.game\_time.setText(f"游戏时间：{self.game\_time\_seconds // 60:02}:{self.game\_time\_seconds % 60:02}")

# 显示偷看次数

self.game\_ui.peek\_count.setVisible(True)

self.game\_ui.peek\_count.setText(f"偷看次数：{self.peek\_count}")

self.is\_answer\_shown = True

def update\_game\_time(self):

"""更新游戏时间"""

if not self.is\_answer\_shown:

self.game\_time\_seconds += 1

minutes = self.game\_time\_seconds // 60

seconds = self.game\_time\_seconds % 60

self.game\_ui.game\_time.setText(f"游戏时间：{minutes:02}:{seconds:02}")

def show\_rules(self):

self.stacked\_widget.setCurrentWidget(self.rules\_widget)

def show\_mode(self):

self.stacked\_widget.setCurrentWidget(self.mode\_widget)

def show\_upload(self):

self.stacked\_widget.setCurrentWidget(self.upload\_widget)

def show\_difficulty(self):

self.stacked\_widget.setCurrentWidget(self.difficulty\_widget)

def show\_game(self):

self.game\_ui.check\_information.setVisible(False)

table = self.game\_ui.sudoku\_table

if table:

table.horizontalHeader().setDefaultSectionSize(55)

table.verticalHeader().setDefaultSectionSize(55)

table.horizontalHeader().setMinimumSectionSize(55)

table.verticalHeader().setMinimumSectionSize(55)

table.setFont(QtGui.QFont("Arial", 22))

for row in range(table.rowCount()):

for col in range(table.columnCount()):

item = table.item(row, col)

if item is None:

item = QTableWidgetItem()

table.setItem(row, col, item)

item.setText("")

item.setTextAlignment(Qt.AlignCenter)

# 强制刷新 UI

table.viewport().update()

table.repaint()

self.stacked\_widget.setCurrentWidget(self.game\_widget)

def show\_main(self):

self.stacked\_widget.setCurrentWidget(self.start\_widget)

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

main\_window = MainApp()

main\_window.show()

sys.exit(app.exec\_())

**2 Ui\_rules.py（调控规则页面）**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'c:\Users\Qiu\Desktop\mess\sudoku\_project\python\rules.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_RulesWindow(object):

def setupUi(self, RulesWindow):

RulesWindow.setObjectName("RulesWindow")

RulesWindow.resize(1110, 659)

self.centralwidget = QtWidgets.QWidget(RulesWindow)

self.centralwidget.setObjectName("centralwidget")

self.listWidget = QtWidgets.QListWidget(self.centralwidget)

self.listWidget.setGeometry(QtCore.QRect(0, 0, 1111, 631))

self.listWidget.setObjectName("listWidget")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(380, -20, 661, 221))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(48)

self.label.setFont(font)

self.label.setObjectName("label")

self.textEdit = QtWidgets.QTextEdit(self.centralwidget)

self.textEdit.setGeometry(QtCore.QRect(50, 140, 1041, 451))

font = QtGui.QFont()

font.setFamily("Bahnschrift SemiLight")

font.setPointSize(28)

self.textEdit.setFont(font)

self.textEdit.setReadOnly(True)

self.textEdit.setObjectName("textEdit")

self.rules\_back = QtWidgets.QPushButton(self.centralwidget)

self.rules\_back.setGeometry(QtCore.QRect(60, 30, 151, 61))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.rules\_back.setFont(font)

self.rules\_back.setObjectName("rules\_back")

RulesWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(RulesWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 1110, 26))

self.menubar.setObjectName("menubar")

RulesWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(RulesWindow)

self.statusbar.setObjectName("statusbar")

RulesWindow.setStatusBar(self.statusbar)

self.retranslateUi(RulesWindow)

QtCore.QMetaObject.connectSlotsByName(RulesWindow)

def retranslateUi(self, RulesWindow):

\_translate = QtCore.QCoreApplication.translate

RulesWindow.setWindowTitle(\_translate("RulesWindow", "MainWindow"))

self.label.setText(\_translate("RulesWindow", "游戏规则"))

self.textEdit.setHtml(\_translate("RulesWindow", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"

"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"

"p, li { white-space: pre-wrap; }\n"

"</style></head><body style=\" font-family:\'Bahnschrift SemiLight\'; font-size:28pt; font-weight:400; font-style:normal;\">\n"

"<p style=\" margin-top:12px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-family:\'SimSun\'; font-size:16pt; font-weight:600;\">游戏目标</span><span style=\" font-family:\'SimSun\'; font-size:16pt;\">：<br />对角线数独是一款基于经典数独规则的游戏，玩家需要在一个 9x9 的网格中填写数字，使每一行、每一列以及每个 3x3 宫内都包含 1 到 9 的数字，且每个数字不能重复。此外，对角线数独还要求两条对角线上（从左上角到右下角和从右上角到左下角）每个数字也不能重复。</span></p>\n"

"<p style=\" margin-top:12px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-family:\'SimSun\'; font-size:16pt; font-weight:600;\">游戏模式</span><span style=\" font-family:\'SimSun\'; font-size:16pt;\">：</span></p>\n"

"<ul style=\"margin-top: 0px; margin-bottom: 0px; margin-left: 0px; margin-right: 0px; -qt-list-indent: 1;\"><li style=\" font-family:\'SimSun\'; font-size:16pt;\" style=\" margin-top:12px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-weight:600;\">SAT求解</span>：玩家可以上传一个由 CNF 格式文件表示的SAT问题。程序将解析题目，同时提供解题判定，判断是否存在解，并展示基础DPLL算法 和 优化后算法的求解对比。</li>\n"

"<li style=\" font-family:\'SimSun\'; font-size:16pt;\" style=\" margin-top:0px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-weight:600;\">随机出题</span>：系统将根据玩家选择的难度（不同的空白格数量）生成一个随机数独题目，玩家需要在限定时间内完成解题。</li></ul>\n"

"<p style=\" margin-top:12px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-family:\'SimSun\'; font-size:16pt; font-weight:600;\">交互功能</span><span style=\" font-family:\'SimSun\'; font-size:16pt;\">：</span></p>\n"

"<ul style=\"margin-top: 0px; margin-bottom: 0px; margin-left: 0px; margin-right: 0px; -qt-list-indent: 1;\"><li style=\" font-family:\'SimSun\'; font-size:16pt;\" style=\" margin-top:12px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-weight:600;\">检查</span>：当玩家完成数独后，点击“检查”按钮，程序将标记出玩家的正确和错误答案，正确答案以绿色显示，错误答案以红色显示，数秒后所有颜色恢复为默认颜色。</li>\n"

"<li style=\" font-family:\'SimSun\'; font-size:16pt;\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-weight:600;\">偷看答案</span>：玩家可在游戏过程中点击“偷看答案”按钮，系统将展示完整的答案，答案在十秒后消失，恢复玩家当前输入的状态。</li>\n"

"<li style=\" font-family:\'SimSun\'; font-size:16pt;\" style=\" margin-top:0px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-weight:600;\">显示答案</span>：玩家在解题过程中可以选择显示完整答案，程序将记录偷看次数，并统计游戏时间。游戏结束时将显示玩家的偷看次数和总用时。</li></ul>\n"

"<p style=\"-qt-paragraph-type:empty; margin-top:12px; margin-bottom:12px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><br /></p></body></html>"))

self.rules\_back.setText(\_translate("RulesWindow", "返回"))

**3 Ui\_start.py（调控初始页面）**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'c:\Users\Qiu\Desktop\杂\python\start.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1065, 678)

MainWindow.setMinimumSize(QtCore.QSize(824, 0))

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setMinimumSize(QtCore.QSize(824, 531))

self.centralwidget.setObjectName("centralwidget")

self.listWidget = QtWidgets.QListWidget(self.centralwidget)

self.listWidget.setGeometry(QtCore.QRect(0, -20, 1061, 671))

self.listWidget.setObjectName("listWidget")

self.title = QtWidgets.QLabel(self.centralwidget)

self.title.setGeometry(QtCore.QRect(130, -10, 901, 271))

font = QtGui.QFont()

font.setFamily("Arial Black")

font.setPointSize(72)

font.setBold(True)

font.setWeight(75)

self.title.setFont(font)

self.title.setObjectName("title")

self.writer = QtWidgets.QLabel(self.centralwidget)

self.writer.setGeometry(QtCore.QRect(670, 230, 431, 81))

font = QtGui.QFont()

font.setFamily("Bookman Old Style")

font.setPointSize(26)

self.writer.setFont(font)

self.writer.setObjectName("writer")

self.game = QtWidgets.QPushButton(self.centralwidget)

self.game.setGeometry(QtCore.QRect(120, 390, 371, 111))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(36)

self.game.setFont(font)

self.game.setObjectName("game")

self.rules = QtWidgets.QPushButton(self.centralwidget)

self.rules.setGeometry(QtCore.QRect(560, 390, 371, 101))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(36)

self.rules.setFont(font)

self.rules.setObjectName("rules")

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 1065, 26))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.title.setText(\_translate("MainWindow", "对角线数独游戏"))

self.writer.setText(\_translate("MainWindow", "设计者：邱月"))

self.game.setText(\_translate("MainWindow", "开始游戏"))

self.rules.setText(\_translate("MainWindow", "游戏规则"))

**4 Ui\_mode.py（调控选择模式页面）**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'c:\Users\Qiu\Desktop\mess\sudoku\_project\python\mode.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1092, 678)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.listWidget = QtWidgets.QListWidget(self.centralwidget)

self.listWidget.setGeometry(QtCore.QRect(0, 10, 1071, 641))

self.listWidget.setObjectName("listWidget")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(340, 180, 551, 141))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(60)

self.label.setFont(font)

self.label.setObjectName("label")

self.upload = QtWidgets.QPushButton(self.centralwidget)

self.upload.setGeometry(QtCore.QRect(120, 390, 371, 101))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(24)

self.upload.setFont(font)

self.upload.setObjectName("upload")

self.puzzle = QtWidgets.QPushButton(self.centralwidget)

self.puzzle.setGeometry(QtCore.QRect(580, 390, 361, 111))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(24)

self.puzzle.setFont(font)

self.puzzle.setObjectName("puzzle")

self.mode\_back = QtWidgets.QPushButton(self.centralwidget)

self.mode\_back.setGeometry(QtCore.QRect(110, 40, 251, 81))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(20)

self.mode\_back.setFont(font)

self.mode\_back.setObjectName("mode\_back")

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 1092, 26))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.label.setText(\_translate("MainWindow", "选择模式"))

self.upload.setText(\_translate("MainWindow", "SAT求解"))

self.puzzle.setText(\_translate("MainWindow", "随机出题"))

self.mode\_back.setText(\_translate("MainWindow", "返回"))

**5 Ui\_upload.py（调控SAT问题求解页面）**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'c:\Users\Qiu\Desktop\mess\sudoku\_project\python\upload.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1126, 775)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.listView = QtWidgets.QListView(self.centralwidget)

self.listView.setGeometry(QtCore.QRect(0, -20, 1101, 811))

font = QtGui.QFont()

font.setPointSize(11)

self.listView.setFont(font)

self.listView.setObjectName("listView")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(390, 20, 401, 91))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(48)

self.label.setFont(font)

self.label.setObjectName("label")

self.file\_choose = QtWidgets.QPushButton(self.centralwidget)

self.file\_choose.setGeometry(QtCore.QRect(400, 150, 141, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(16)

self.file\_choose.setFont(font)

self.file\_choose.setObjectName("file\_choose")

self.saved\_path\_label = QtWidgets.QLabel(self.centralwidget)

self.saved\_path\_label.setGeometry(QtCore.QRect(90, 300, 331, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.saved\_path\_label.setFont(font)

self.saved\_path\_label.setObjectName("saved\_path\_label")

self.judge\_label = QtWidgets.QLabel(self.centralwidget)

self.judge\_label.setGeometry(QtCore.QRect(170, 650, 141, 31))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.judge\_label.setFont(font)

self.judge\_label.setObjectName("judge\_label")

self.judge = QtWidgets.QLabel(self.centralwidget)

self.judge.setGeometry(QtCore.QRect(370, 640, 141, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.judge.setFont(font)

self.judge.setObjectName("judge")

self.dpll\_address\_label = QtWidgets.QLabel(self.centralwidget)

self.dpll\_address\_label.setGeometry(QtCore.QRect(70, 470, 191, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(12)

self.dpll\_address\_label.setFont(font)

self.dpll\_address\_label.setObjectName("dpll\_address\_label")

self.cdcl\_time\_label = QtWidgets.QLabel(self.centralwidget)

self.cdcl\_time\_label.setGeometry(QtCore.QRect(70, 530, 191, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.cdcl\_time\_label.setFont(font)

self.cdcl\_time\_label.setObjectName("cdcl\_time\_label")

self.cdcl\_time = QtWidgets.QLabel(self.centralwidget)

self.cdcl\_time.setGeometry(QtCore.QRect(290, 540, 131, 20))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.cdcl\_time.setFont(font)

self.cdcl\_time.setObjectName("cdcl\_time")

self.cdcl\_address\_label = QtWidgets.QLabel(self.centralwidget)

self.cdcl\_address\_label.setGeometry(QtCore.QRect(70, 570, 181, 71))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(12)

self.cdcl\_address\_label.setFont(font)

self.cdcl\_address\_label.setObjectName("cdcl\_address\_label")

self.rate\_label = QtWidgets.QLabel(self.centralwidget)

self.rate\_label.setGeometry(QtCore.QRect(610, 640, 131, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(20)

self.rate\_label.setFont(font)

self.rate\_label.setObjectName("rate\_label")

self.optimization\_rate = QtWidgets.QLabel(self.centralwidget)

self.optimization\_rate.setGeometry(QtCore.QRect(790, 650, 171, 31))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(18)

self.optimization\_rate.setFont(font)

self.optimization\_rate.setObjectName("optimization\_rate")

self.upload\_back = QtWidgets.QPushButton(self.centralwidget)

self.upload\_back.setGeometry(QtCore.QRect(90, 30, 131, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(18)

self.upload\_back.setFont(font)

self.upload\_back.setObjectName("upload\_back")

self.label\_4 = QtWidgets.QLabel(self.centralwidget)

self.label\_4.setGeometry(QtCore.QRect(90, 150, 311, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.label\_4.setFont(font)

self.label\_4.setObjectName("label\_4")

self.cnf\_path = QtWidgets.QLineEdit(self.centralwidget)

self.cnf\_path.setGeometry(QtCore.QRect(90, 230, 921, 31))

self.cnf\_path.setFrame(True)

self.cnf\_path.setObjectName("cnf\_path")

self.save\_button = QtWidgets.QPushButton(self.centralwidget)

self.save\_button.setGeometry(QtCore.QRect(730, 290, 221, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.save\_button.setFont(font)

self.save\_button.setObjectName("save\_button")

self.saved\_path = QtWidgets.QLineEdit(self.centralwidget)

self.saved\_path.setGeometry(QtCore.QRect(80, 360, 931, 31))

self.saved\_path.setObjectName("saved\_path")

self.source\_button = QtWidgets.QPushButton(self.centralwidget)

self.source\_button.setGeometry(QtCore.QRect(470, 290, 211, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.source\_button.setFont(font)

self.source\_button.setObjectName("source\_button")

self.solve\_button = QtWidgets.QPushButton(self.centralwidget)

self.solve\_button.setGeometry(QtCore.QRect(560, 140, 151, 61))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(16)

self.solve\_button.setFont(font)

self.solve\_button.setObjectName("solve\_button")

self.dpll\_address = QtWidgets.QLineEdit(self.centralwidget)

self.dpll\_address.setGeometry(QtCore.QRect(250, 470, 771, 31))

self.dpll\_address.setObjectName("dpll\_address")

self.cdcl\_address = QtWidgets.QLineEdit(self.centralwidget)

self.cdcl\_address.setGeometry(QtCore.QRect(260, 590, 771, 31))

self.cdcl\_address.setObjectName("cdcl\_address")

self.dpll\_button = QtWidgets.QPushButton(self.centralwidget)

self.dpll\_button.setGeometry(QtCore.QRect(560, 420, 331, 41))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.dpll\_button.setFont(font)

self.dpll\_button.setObjectName("dpll\_button")

self.cdcl\_button = QtWidgets.QPushButton(self.centralwidget)

self.cdcl\_button.setGeometry(QtCore.QRect(560, 540, 331, 31))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.cdcl\_button.setFont(font)

self.cdcl\_button.setObjectName("cdcl\_button")

self.layoutWidget = QtWidgets.QWidget(self.centralwidget)

self.layoutWidget.setGeometry(QtCore.QRect(70, 410, 351, 51))

self.layoutWidget.setObjectName("layoutWidget")

self.horizontalLayout = QtWidgets.QHBoxLayout(self.layoutWidget)

self.horizontalLayout.setContentsMargins(0, 0, 0, 0)

self.horizontalLayout.setObjectName("horizontalLayout")

self.dpll\_time\_label = QtWidgets.QLabel(self.layoutWidget)

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.dpll\_time\_label.setFont(font)

self.dpll\_time\_label.setObjectName("dpll\_time\_label")

self.horizontalLayout.addWidget(self.dpll\_time\_label)

self.dpll\_time = QtWidgets.QLabel(self.layoutWidget)

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(14)

self.dpll\_time.setFont(font)

self.dpll\_time.setObjectName("dpll\_time")

self.horizontalLayout.addWidget(self.dpll\_time)

self.better\_solve\_button\_1 = QtWidgets.QPushButton(self.centralwidget)

self.better\_solve\_button\_1.setGeometry(QtCore.QRect(720, 130, 111, 71))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(18)

self.better\_solve\_button\_1.setFont(font)

self.better\_solve\_button\_1.setObjectName("better\_solve\_button\_1")

self.better\_solve\_button\_2 = QtWidgets.QPushButton(self.centralwidget)

self.better\_solve\_button\_2.setGeometry(QtCore.QRect(840, 130, 111, 71))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(18)

self.better\_solve\_button\_2.setFont(font)

self.better\_solve\_button\_2.setObjectName("better\_solve\_button\_2")

self.better\_solve\_button\_3 = QtWidgets.QPushButton(self.centralwidget)

self.better\_solve\_button\_3.setGeometry(QtCore.QRect(960, 130, 111, 71))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(18)

self.better\_solve\_button\_3.setFont(font)

self.better\_solve\_button\_3.setObjectName("better\_solve\_button\_3")

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 1126, 26))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

self.file\_choose.clicked.connect(self.cnf\_path.show) # type: ignore

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.label.setText(\_translate("MainWindow", "问题求解"))

self.file\_choose.setText(\_translate("MainWindow", "选择文件"))

self.saved\_path\_label.setText(\_translate("MainWindow", "解析成txt文件路径（供验证）："))

self.judge\_label.setText(\_translate("MainWindow", "结果判定为："))

self.judge.setText(\_translate("MainWindow", "未求解"))

self.dpll\_address\_label.setText(\_translate("MainWindow", "求解cnf文件地址："))

self.cdcl\_time\_label.setText(\_translate("MainWindow", "优化算法实现时间："))

self.cdcl\_time.setText(\_translate("MainWindow", "（未实现）"))

self.cdcl\_address\_label.setText(\_translate("MainWindow", "求解cnf文件地址："))

self.rate\_label.setText(\_translate("MainWindow", "优化率："))

self.optimization\_rate.setText(\_translate("MainWindow", "未完全求解"))

self.upload\_back.setText(\_translate("MainWindow", "返回"))

self.label\_4.setText(\_translate("MainWindow", "请输入文件路径（cnf格式）："))

self.save\_button.setText(\_translate("MainWindow", "打开解析文件"))

self.source\_button.setText(\_translate("MainWindow", "打开原文件"))

self.solve\_button.setText(\_translate("MainWindow", "基础求解"))

self.dpll\_button.setText(\_translate("MainWindow", "打开基础算法求解文件"))

self.cdcl\_button.setText(\_translate("MainWindow", "打开优化算法求解文件"))

self.dpll\_time\_label.setText(\_translate("MainWindow", "基础算法实现时间："))

self.dpll\_time.setText(\_translate("MainWindow", "（未实现）"))

self.better\_solve\_button\_1.setText(\_translate("MainWindow", "优化1"))

self.better\_solve\_button\_2.setText(\_translate("MainWindow", "优化2"))

self.better\_solve\_button\_3.setText(\_translate("MainWindow", "优化3"))

**6 Ui\_difficulty.py（调控选择难度页面）**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'c:\Users\Qiu\Desktop\杂\python\difficulty.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1175, 838)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.listView = QtWidgets.QListView(self.centralwidget)

self.listView.setGeometry(QtCore.QRect(10, -90, 1151, 841))

self.listView.setObjectName("listView")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(370, 90, 501, 201))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(60)

self.label.setFont(font)

self.label.setObjectName("label")

self.easy = QtWidgets.QPushButton(self.centralwidget)

self.easy.setGeometry(QtCore.QRect(280, 360, 231, 81))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.easy.setFont(font)

self.easy.setObjectName("easy")

self.simple = QtWidgets.QPushButton(self.centralwidget)

self.simple.setGeometry(QtCore.QRect(620, 360, 221, 91))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.simple.setFont(font)

self.simple.setObjectName("simple")

self.intermediate = QtWidgets.QPushButton(self.centralwidget)

self.intermediate.setGeometry(QtCore.QRect(280, 510, 231, 71))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.intermediate.setFont(font)

self.intermediate.setObjectName("intermediate")

self.expert = QtWidgets.QPushButton(self.centralwidget)

self.expert.setGeometry(QtCore.QRect(620, 500, 221, 81))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.expert.setFont(font)

self.expert.setObjectName("expert")

self.difficulty\_back = QtWidgets.QPushButton(self.centralwidget)

self.difficulty\_back.setGeometry(QtCore.QRect(100, 40, 171, 61))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(18)

self.difficulty\_back.setFont(font)

self.difficulty\_back.setObjectName("difficulty\_back")

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 1175, 26))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.label.setText(\_translate("MainWindow", "选择难度"))

self.easy.setText(\_translate("MainWindow", "简单"))

self.simple.setText(\_translate("MainWindow", "基础"))

self.intermediate.setText(\_translate("MainWindow", "中级"))

self.expert.setText(\_translate("MainWindow", "专家"))

self.difficulty\_back.setText(\_translate("MainWindow", "返回"))

**7 Ui\_game.py（调控游戏页面）**

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'c:\Users\Qiu\Desktop\mess\sudoku\_project\python\game.ui'

#

# Created by: PyQt5 UI code generator 5.15.9

#

# WARNING: Any manual changes made to this file will be lost when pyuic5 is

# run again. Do not edit this file unless you know what you are doing.

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(1140, 683)

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.listView = QtWidgets.QListView(self.centralwidget)

self.listView.setGeometry(QtCore.QRect(-10, 0, 1131, 661))

font = QtGui.QFont()

font.setKerning(False)

self.listView.setFont(font)

self.listView.setEditTriggers(QtWidgets.QAbstractItemView.AllEditTriggers)

self.listView.setObjectName("listView")

self.label = QtWidgets.QLabel(self.centralwidget)

self.label.setGeometry(QtCore.QRect(80, 40, 301, 91))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(24)

self.label.setFont(font)

self.label.setObjectName("label")

self.check\_button = QtWidgets.QPushButton(self.centralwidget)

self.check\_button.setGeometry(QtCore.QRect(880, 180, 161, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(16)

self.check\_button.setFont(font)

self.check\_button.setObjectName("check\_button")

self.peek\_button = QtWidgets.QPushButton(self.centralwidget)

self.peek\_button.setGeometry(QtCore.QRect(890, 360, 161, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(16)

self.peek\_button.setFont(font)

self.peek\_button.setObjectName("peek\_button")

self.show\_button = QtWidgets.QPushButton(self.centralwidget)

self.show\_button.setGeometry(QtCore.QRect(890, 440, 161, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(16)

self.show\_button.setFont(font)

self.show\_button.setObjectName("show\_button")

self.peek\_count = QtWidgets.QLabel(self.centralwidget)

self.peek\_count.setGeometry(QtCore.QRect(900, 550, 171, 31))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(12)

self.peek\_count.setFont(font)

self.peek\_count.setObjectName("peek\_count")

self.game\_time = QtWidgets.QLabel(self.centralwidget)

self.game\_time.setGeometry(QtCore.QRect(880, 30, 161, 121))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(12)

self.game\_time.setFont(font)

self.game\_time.setObjectName("game\_time")

self.check\_information = QtWidgets.QLineEdit(self.centralwidget)

self.check\_information.setGeometry(QtCore.QRect(220, 560, 591, 51))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(16)

self.check\_information.setFont(font)

self.check\_information.setObjectName("check\_information")

self.game\_back = QtWidgets.QPushButton(self.centralwidget)

self.game\_back.setGeometry(QtCore.QRect(40, 550, 151, 61))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.game\_back.setFont(font)

self.game\_back.setObjectName("game\_back")

self.sudoku\_table = QtWidgets.QTableWidget(self.centralwidget)

self.sudoku\_table.setEnabled(True)

self.sudoku\_table.setGeometry(QtCore.QRect(240, 30, 501, 501))

font = QtGui.QFont()

font.setFamily("Arial")

font.setPointSize(22)

self.sudoku\_table.setFont(font)

self.sudoku\_table.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAsNeeded)

self.sudoku\_table.setAutoScroll(True)

self.sudoku\_table.setEditTriggers(QtWidgets.QAbstractItemView.AllEditTriggers)

self.sudoku\_table.setTabKeyNavigation(True)

self.sudoku\_table.setProperty("showDropIndicator", True)

self.sudoku\_table.setShowGrid(True)

self.sudoku\_table.setGridStyle(QtCore.Qt.SolidLine)

self.sudoku\_table.setObjectName("sudoku\_table")

self.sudoku\_table.setColumnCount(9)

self.sudoku\_table.setRowCount(9)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(0, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(1, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(2, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(3, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(4, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(5, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(6, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(7, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setVerticalHeaderItem(8, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(0, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(1, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(2, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(3, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(4, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(5, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(6, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(7, item)

item = QtWidgets.QTableWidgetItem()

self.sudoku\_table.setHorizontalHeaderItem(8, item)

self.sudoku\_table.horizontalHeader().setVisible(False)

self.sudoku\_table.horizontalHeader().setCascadingSectionResizes(False)

self.sudoku\_table.horizontalHeader().setDefaultSectionSize(55)

self.sudoku\_table.horizontalHeader().setHighlightSections(False)

self.sudoku\_table.horizontalHeader().setMinimumSectionSize(55)

self.sudoku\_table.horizontalHeader().setSortIndicatorShown(False)

self.sudoku\_table.horizontalHeader().setStretchLastSection(False)

self.sudoku\_table.verticalHeader().setVisible(False)

self.sudoku\_table.verticalHeader().setDefaultSectionSize(55)

self.sudoku\_table.verticalHeader().setHighlightSections(False)

self.sudoku\_table.verticalHeader().setMinimumSectionSize(55)

MainWindow.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(MainWindow)

self.menubar.setGeometry(QtCore.QRect(0, 0, 1140, 26))

self.menubar.setObjectName("menubar")

MainWindow.setMenuBar(self.menubar)

self.statusbar = QtWidgets.QStatusBar(MainWindow)

self.statusbar.setObjectName("statusbar")

MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

self.label.setText(\_translate("MainWindow", "题目："))

self.check\_button.setText(\_translate("MainWindow", "检查"))

self.peek\_button.setText(\_translate("MainWindow", "偷看答案"))

self.show\_button.setText(\_translate("MainWindow", "显示答案"))

self.peek\_count.setText(\_translate("MainWindow", "偷看次数：0"))

self.game\_time.setText(\_translate("MainWindow", "游戏时间：00:00"))

self.check\_information.setText(\_translate("MainWindow", "这里会显示检查信息！"))

self.game\_back.setText(\_translate("MainWindow", "返回"))

**8 head.h**

// head.h

#ifndef HEAD\_H

#define HEAD\_H

#include <math.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

// 定义状态类型

typedef int status;

#define OK 1

#define ERROR 0

#define FOUND 1

#define NOTFOUND 0

#define UNASSIGNED 0

#define YES 1

#define NO 0

#define N 9 // 数独的大小

// 定义数据结构

typedef struct LiteralNode

{

int data;

struct LiteralNode \*right;

} LiteralNode;

typedef struct ClauseNode

{

struct LiteralNode \*right;

struct ClauseNode \*down;

} ClauseNode;

// 函数声明

ClauseNode \*AddClause(ClauseNode \*s, int var);

ClauseNode \*IsUnitClause(ClauseNode \*s);

ClauseNode \*CopyS(ClauseNode \*s);

ClauseNode \*CopyClause(ClauseNode \*s);

ClauseNode \*DeleteClause(ClauseNode \*s);

ClauseNode \*DeleteLiteral(ClauseNode \*s, int var);

status Print(ClauseNode \*s);

status IsEmptyClause(ClauseNode \*s);

status RecordTruth(ClauseNode \*s, int \*truth\_table);

status RemoveVar(ClauseNode \*s, int var);

int PickVar(ClauseNode \*s);

ClauseNode \*CnfParser(int \*literal\_num, char \*filename, int \*max\_var);

int PickVar\_1(ClauseNode \*s);

int PickVar\_2(ClauseNode \*s);

int PickVar\_3(ClauseNode \*s);

void clause\_print(ClauseNode \*root, FILE \*output\_file);

#endif // HEAD\_H

**9 cnf\_parser.c（解析cnf文件）**

// cnf\_parser.c

#include "head.h"

// 解析 CNF 文件

ClauseNode \*CnfParser(int \*literal\_num, const char \*filename, int \*max\_var)

{

FILE \*file = fopen(filename, "r");

if (!file)

{

printf("Error opening file: %s\n", filename);

return NULL;

}

ClauseNode \*cnf = (ClauseNode \*)malloc(sizeof(ClauseNode));

cnf->down = NULL;

cnf->right = NULL;

ClauseNode \*cnf\_tail = cnf; // 尾指针，保持对最后一个子句的引用

\*max\_var = 0;

char line[11000];

while (fgets(line, sizeof(line), file))

{

if (line[0] == 'c' || line[0] == 'p') // 跳过注释和头信息

{

continue;

}

ClauseNode \*new\_clause = (ClauseNode \*)malloc(sizeof(ClauseNode));

new\_clause->down = NULL;

new\_clause->right = NULL;

LiteralNode \*literal\_tail = NULL; // 文字链表的尾指针

char \*token = strtok(line, " ");

while (token)

{

int lit = atoi(token);

if (lit == 0)

break; // 结束符为0

if (abs(lit) > \*max\_var)

\*max\_var = abs(lit); // 更新最大变元编号

LiteralNode \*literal = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal->data = lit;

literal->right = NULL; // 新的文字节点的next为NULL

// 将新文字插入到子句链表的尾部

if (new\_clause->right == NULL)

{

new\_clause->right = literal;

literal\_tail = literal; // 初始化尾指针

}

else

{

literal\_tail->right = literal;

literal\_tail = literal; // 更新尾指针

}

token = strtok(NULL, " ");

}

// 将新子句插入到CNF链表的尾部

cnf\_tail->down = new\_clause;

cnf\_tail = new\_clause;

}

fclose(file);

return cnf;

}

// 打印 CNF 公式

void clause\_print(ClauseNode \*root, FILE \*output\_file)

{

ClauseNode \*current\_clause = root->down; // 跳过头结点

while (current\_clause)

{

LiteralNode \*literal = current\_clause->right;

while (literal)

{

fprintf(output\_file, "%d ", literal->data);

literal = literal->right;

}

fprintf(output\_file, "0\n"); // 每个子句结束以0结尾

current\_clause = current\_clause->down;

}

}

// 主程序

int main(int argc, char \*argv[])

{

if (argc != 3)

{

fprintf(stderr, "Usage: %s <cnf file path> <output file path>\n", argv[0]);

return 1;

}

const char \*cnf\_path = argv[1];

const char \*output\_path = argv[2];

int literal\_num = 0;

int max\_var = 0;

// 解析 CNF 文件

ClauseNode \*cnf = CnfParser(&literal\_num, cnf\_path, &max\_var);

if (!cnf)

{

return 1;

}

// 打印 CNF 公式到输出文件

FILE \*output\_file = fopen(output\_path, "w");

if (!output\_file)

{

fprintf(stderr, "Cannot open the output file: %s\n", output\_path);

return 1;

}

clause\_print(cnf, output\_file);

fclose(output\_file);

return 0;

}

**10 solver.c（利用DPLL算法求解SAT问题）**

// solver.c

#include "head.h"

// DPLL 求解器

status DpllSolver(ClauseNode \*s, int \*truth\_table)

{

ClauseNode \*s\_tmp = s;

ClauseNode \*unit\_clause = IsUnitClause(s\_tmp);

while (unit\_clause)

{

RecordTruth(unit\_clause, truth\_table);

int var = unit\_clause->right->data;

RemoveVar(s, var);

if (s->down == NULL)

return FOUND;

else if (IsEmptyClause(s))

return NOTFOUND;

s\_tmp = s;

unit\_clause = IsUnitClause(s\_tmp);

}

int var = PickVar(s);

if (DpllSolver(AddClause(CopyS(s), var), truth\_table))

return FOUND;

return DpllSolver(AddClause(s, -var), truth\_table);

}

// 添加子句到子句集

ClauseNode \*AddClause(ClauseNode \*s, int var)

{

ClauseNode \*c\_tmp = (ClauseNode \*)malloc(sizeof(ClauseNode));

c\_tmp->right = (LiteralNode \*)malloc(sizeof(LiteralNode));

c\_tmp->right->data = var;

c\_tmp->right->right = NULL;

c\_tmp->down = s->down;

s->down = c\_tmp;

return s;

}

// 判断是否存在单子句

ClauseNode \*IsUnitClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s;

while (c\_tmp)

{

if (c\_tmp->right && !c\_tmp->right->right)

return c\_tmp;

c\_tmp = c\_tmp->down;

}

return NULL;

}

// 复制子句集

ClauseNode \*CopyS(ClauseNode \*s)

{

ClauseNode \*s\_new, \*c\_tmp, \*s\_origin = s->down;

s\_new = (ClauseNode \*)malloc(sizeof(ClauseNode));

s\_new->right = NULL;

c\_tmp = s\_new;

while (s\_origin)

{

c\_tmp->down = CopyClause(s\_origin);

c\_tmp = c\_tmp->down;

s\_origin = s\_origin->down;

}

return s\_new;

}

// 复制单个子句

ClauseNode \*CopyClause(ClauseNode \*s)

{

ClauseNode \*clause\_tmp = (ClauseNode \*)malloc(sizeof(ClauseNode));

LiteralNode \*literal\_tmp, \*s\_tmp = s->right;

clause\_tmp->down = NULL;

clause\_tmp->right = NULL;

if (s\_tmp)

{

literal\_tmp = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal\_tmp->data = s\_tmp->data;

literal\_tmp->right = NULL;

s\_tmp = s\_tmp->right;

clause\_tmp->right = literal\_tmp;

}

while (s\_tmp)

{

literal\_tmp->right = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal\_tmp->right->data = s\_tmp->data;

literal\_tmp->right->right = NULL;

literal\_tmp = literal\_tmp->right;

s\_tmp = s\_tmp->right;

}

return clause\_tmp;

}

// 删除子句

ClauseNode \*DeleteClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s;

LiteralNode \*l;

while (s->right)

{

l = s->right;

s->right = s->right->right;

free(l);

}

s = s->down;

free(c\_tmp);

return s;

}

// 删除负变元

ClauseNode \*DeleteLiteral(ClauseNode \*s, int var)

{

LiteralNode \*l = s->right, \*l\_tmp;

if (l->data == -var)

{

l\_tmp = s->right->right;

free(l);

s->right = l\_tmp;

return s;

}

else if (l->data == var)

{

return NULL;

}

while (l)

{

if (l->right && l->right->data == -var)

{

l\_tmp = l->right;

l->right = l->right->right;

free(l\_tmp);

}

else if (l->right && l->right->data == var)

{

return NULL;

}

l = l->right;

}

return s;

}

// 判断是否存在空子句

status IsEmptyClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s->down;

while (c\_tmp)

{

if (!c\_tmp->right)

return YES;

c\_tmp = c\_tmp->down;

}

return NO;

}

// 记录单子句中元素的真值

status RecordTruth(ClauseNode \*s, int \*truth\_table)

{

int idx = abs(s->right->data) - 1;

if (s->right->data > 0)

truth\_table[idx] = 1;

else

truth\_table[idx] = 0;

return OK;

}

// 删除变元所在子句与负变元

status RemoveVar(ClauseNode \*s, int var)

{

ClauseNode \*c\_tmp = s->down, \*last\_c\_tmp = s;

while (c\_tmp)

{

c\_tmp = DeleteLiteral(c\_tmp, var);

if (c\_tmp == NULL)

{

c\_tmp = DeleteClause(last\_c\_tmp->down);

last\_c\_tmp->down = c\_tmp;

continue;

}

if (c\_tmp == NULL)

break;

last\_c\_tmp = c\_tmp;

c\_tmp = c\_tmp->down;

}

return OK;

}

// 选取一个变元

int PickVar(ClauseNode \*s)

{

int var = s->down->right->data;

return var;

}

// 打印子句集

status Print(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s->down;

LiteralNode \*l\_tmp;

while (c\_tmp)

{

l\_tmp = c\_tmp->right;

while (l\_tmp)

{

printf("%d ", l\_tmp->data);

l\_tmp = l\_tmp->right;

}

if (c\_tmp->right)

printf("\n");

else

printf("\n");

c\_tmp = c\_tmp->down;

}

return OK;

}

// 解析 CNF 文件

ClauseNode \*CnfParser(int \*literal\_num, char \*filename, int \*max\_var)

{

FILE \*file = fopen(filename, "r");

if (!file)

{

printf("Error opening file: %s\n", filename);

return NULL;

}

ClauseNode \*cnf = (ClauseNode \*)malloc(sizeof(ClauseNode));

cnf->down = NULL;

cnf->right = NULL;

\*max\_var = 0;

char line[11000];

while (fgets(line, sizeof(line), file))

{

if (line[0] == 'c' || line[0] == 'p')

{

continue;

}

ClauseNode \*new\_clause = (ClauseNode \*)malloc(sizeof(ClauseNode));

new\_clause->down = NULL;

new\_clause->right = NULL;

char \*token = strtok(line, " ");

while (token)

{

int lit = atoi(token);

if (lit == 0)

break;

if (abs(lit) > \*max\_var)

\*max\_var = abs(lit); // 记录最大变元编号

LiteralNode \*literal = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal->data = lit;

literal->right = new\_clause->right;

new\_clause->right = literal;

token = strtok(NULL, " ");

}

new\_clause->down = cnf->down;

cnf->down = new\_clause;

}

fclose(file);

return cnf;

}

// 主程序入口

int main(int argc, char \*argv[])

{

if (argc != 2)

{

fprintf(stderr, "Usage: %s <cnf file path>\n", argv[0]);

return 1;

}

char cnf\_path[256];

strcpy(cnf\_path, argv[1]);

int truth\_table[5000] = {0};

int max\_var = 0;

ClauseNode \*cnf = CnfParser(truth\_table, cnf\_path, &max\_var);

if (!cnf)

{

fprintf(stderr, "Error reading CNF file: %s\n", cnf\_path);

return 1;

}

clock\_t start\_time = clock();

status result = DpllSolver(cnf, truth\_table);

clock\_t end\_time = clock();

double elapsed\_time = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC \* 1000;

char result\_path[1024];

snprintf(result\_path, sizeof(result\_path), "%s.res", cnf\_path);

FILE \*output\_file = fopen(result\_path, "w");

if (!output\_file)

{

fprintf(stderr, "Cannot open output file.\n");

return 1;

}

if (result == FOUND)

{

fprintf(output\_file, "s 1\nv ");

for (int i = 0; i < max\_var; i++)

{

if (truth\_table[i] == 1)

fprintf(output\_file, "%d ", i + 1);

else if (truth\_table[i] == 0)

fprintf(output\_file, "-%d ", i + 1);

}

fprintf(output\_file, "\n");

}

else

{

fprintf(output\_file, "s 0\n");

}

fprintf(output\_file, "t %.2f\n", elapsed\_time);

fclose(output\_file);

Print(cnf);

return 0;

}

**11 better\_solver.c（优化算法求解`SAT问题）**

// better\_solver.c

#include "head.h"

// 修改后的 DPLL 求解器

status DpllSolver(ClauseNode \*s, int \*truth\_table, int pickvar\_choice)

{

ClauseNode \*s\_tmp = s;

ClauseNode \*unit\_clause = IsUnitClause(s\_tmp);

while (unit\_clause)

{

RecordTruth(unit\_clause, truth\_table);

int var = unit\_clause->right->data;

RemoveVar(s, var);

if (s->down == NULL)

return FOUND;

else if (IsEmptyClause(s))

return NOTFOUND;

s\_tmp = s;

unit\_clause = IsUnitClause(s\_tmp);

}

// 根据参数选择不同的 PickVar 函数

int var;

if (pickvar\_choice == 1)

var = PickVar\_1(s);

else if (pickvar\_choice == 2)

var = PickVar\_2(s);

else

var = PickVar\_3(s);

if (DpllSolver(AddClause(CopyS(s), var), truth\_table, pickvar\_choice))

return FOUND;

return DpllSolver(AddClause(s, -var), truth\_table, pickvar\_choice);

}

// 添加子句到子句集

ClauseNode \*AddClause(ClauseNode \*s, int var)

{

ClauseNode \*c\_tmp = (ClauseNode \*)malloc(sizeof(ClauseNode));

c\_tmp->right = (LiteralNode \*)malloc(sizeof(LiteralNode));

c\_tmp->right->data = var;

c\_tmp->right->right = NULL;

c\_tmp->down = s->down;

s->down = c\_tmp;

return s;

}

// 判断是否存在单子句

ClauseNode \*IsUnitClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s;

while (c\_tmp)

{

if (c\_tmp->right && !c\_tmp->right->right)

return c\_tmp;

c\_tmp = c\_tmp->down;

}

return NULL;

}

// 复制子句集

ClauseNode \*CopyS(ClauseNode \*s)

{

ClauseNode \*s\_new, \*c\_tmp, \*s\_origin = s->down;

s\_new = (ClauseNode \*)malloc(sizeof(ClauseNode));

s\_new->right = NULL;

c\_tmp = s\_new;

while (s\_origin)

{

c\_tmp->down = CopyClause(s\_origin);

c\_tmp = c\_tmp->down;

s\_origin = s\_origin->down;

}

return s\_new;

}

// 复制单个子句

ClauseNode \*CopyClause(ClauseNode \*s)

{

ClauseNode \*clause\_tmp = (ClauseNode \*)malloc(sizeof(ClauseNode));

LiteralNode \*literal\_tmp, \*s\_tmp = s->right;

clause\_tmp->down = NULL;

clause\_tmp->right = NULL;

if (s\_tmp)

{

literal\_tmp = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal\_tmp->data = s\_tmp->data;

literal\_tmp->right = NULL;

s\_tmp = s\_tmp->right;

clause\_tmp->right = literal\_tmp;

}

while (s\_tmp)

{

literal\_tmp->right = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal\_tmp->right->data = s\_tmp->data;

literal\_tmp->right->right = NULL;

literal\_tmp = literal\_tmp->right;

s\_tmp = s\_tmp->right;

}

return clause\_tmp;

}

// 删除子句

ClauseNode \*DeleteClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s;

LiteralNode \*l;

while (s->right)

{

l = s->right;

s->right = s->right->right;

free(l);

}

s = s->down;

free(c\_tmp);

return s;

}

// 删除负变元

ClauseNode \*DeleteLiteral(ClauseNode \*s, int var)

{

LiteralNode \*l = s->right, \*l\_tmp;

if (l->data == -var)

{

l\_tmp = s->right->right;

free(l);

s->right = l\_tmp;

return s;

}

else if (l->data == var)

{

return NULL;

}

while (l)

{

if (l->right && l->right->data == -var)

{

l\_tmp = l->right;

l->right = l->right->right;

free(l\_tmp);

}

else if (l->right && l->right->data == var)

{

return NULL;

}

l = l->right;

}

return s;

}

// 判断是否存在空子句

status IsEmptyClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s->down;

while (c\_tmp)

{

if (!c\_tmp->right)

return YES;

c\_tmp = c\_tmp->down;

}

return NO;

}

// 记录单子句中元素的真值

status RecordTruth(ClauseNode \*s, int \*truth\_table)

{

int idx = abs(s->right->data) - 1;

if (s->right->data > 0)

truth\_table[idx] = 1;

else

truth\_table[idx] = 0;

return OK;

}

// 删除变元所在子句与负变元

status RemoveVar(ClauseNode \*s, int var)

{

ClauseNode \*c\_tmp = s->down, \*last\_c\_tmp = s;

while (c\_tmp)

{

c\_tmp = DeleteLiteral(c\_tmp, var);

if (c\_tmp == NULL)

{

c\_tmp = DeleteClause(last\_c\_tmp->down);

last\_c\_tmp->down = c\_tmp;

continue;

}

if (c\_tmp == NULL)

break;

last\_c\_tmp = c\_tmp;

c\_tmp = c\_tmp->down;

}

return OK;

}

int PickVar\_1(ClauseNode \*s) // 选择出现次数最多的变元

{

int max\_var = 0;

ClauseNode \*c\_tmp = s->down;

while (c\_tmp)

{

LiteralNode \*l\_tmp = c\_tmp->right;

while (l\_tmp)

{

if (abs(l\_tmp->data) > max\_var)

max\_var = abs(l\_tmp->data);

l\_tmp = l\_tmp->right;

}

c\_tmp = c\_tmp->down;

}

int \*count = (int \*)malloc(sizeof(int) \* (2 \* max\_var + 1));

for (int i = 0; i <= 2 \* max\_var; i++)

count[i] = 0;

c\_tmp = s->down;

while (c\_tmp)

{

LiteralNode \*l\_tmp = c\_tmp->right;

while (l\_tmp)

{

int lit = l\_tmp->data;

if (lit > 0)

count[lit]++;

else

count[max\_var - lit]++;

l\_tmp = l\_tmp->right;

}

c\_tmp = c\_tmp->down;

}

int selected\_var = 1;

int max\_count = 0;

for (int i = 1; i <= max\_var; i++)

{

if (count[i] > max\_count)

{

max\_count = count[i];

selected\_var = i;

}

}

if (max\_count == 0)

{

for (int i = max\_var + 1; i <= 2 \* max\_var; i++)

{

if (count[i] > max\_count)

{

max\_count = count[i];

selected\_var = max\_var - i;

}

}

}

free(count);

return selected\_var;

}

int PickVar\_2(ClauseNode \*s) // 选择下一个变元

{

int var = s->down->right->data;

return var;

}

int PickVar\_3(ClauseNode \*s) // 为所在子句较短的变元分配较大权重

{

int max\_var = 0;

ClauseNode \*c\_tmp = s->down;

while (c\_tmp)

{

LiteralNode \*l\_tmp = c\_tmp->right;

while (l\_tmp)

{

if (abs(l\_tmp->data) > max\_var)

max\_var = abs(l\_tmp->data);

l\_tmp = l\_tmp->right;

}

c\_tmp = c\_tmp->down;

}

double \*weight = (double \*)malloc(sizeof(double) \* (2 \* max\_var + 1));

for (int i = 0; i <= 2 \* max\_var; i++)

weight[i] = 0.0;

c\_tmp = s->down;

while (c\_tmp)

{

int clause\_length = 0;

LiteralNode \*l\_tmp = c\_tmp->right;

while (l\_tmp)

{

clause\_length++;

l\_tmp = l\_tmp->right;

}

l\_tmp = c\_tmp->right;

while (l\_tmp)

{

int lit = l\_tmp->data;

double factor = pow(2.0, (double)(-clause\_length));

if (lit > 0)

weight[lit] += factor;

else

weight[max\_var - lit] += factor;

l\_tmp = l\_tmp->right;

}

c\_tmp = c\_tmp->down;

}

double max\_weight = 0.0;

int selected\_var = 1;

for (int i = 1; i <= max\_var; i++)

{

if (weight[i] + weight[i + max\_var] > max\_weight)

{

max\_weight = weight[i] + weight[i + max\_var];

selected\_var = i;

}

}

if (weight[selected\_var] < weight[selected\_var + max\_var])

selected\_var = -selected\_var;

free(weight);

return selected\_var;

}

// 打印子句集

status Print(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s->down;

LiteralNode \*l\_tmp;

printf("\n-----Start-----\n");

while (c\_tmp)

{

l\_tmp = c\_tmp->right;

while (l\_tmp)

{

printf("%d ", l\_tmp->data);

l\_tmp = l\_tmp->right;

}

if (c\_tmp->right)

printf("\n");

else

printf("空\n");

c\_tmp = c\_tmp->down;

}

printf("------End------\n");

return OK;

}

// 解析 CNF 文件

ClauseNode \*CnfParser(int \*literal\_num, char \*filename, int \*max\_var)

{

FILE \*file = fopen(filename, "r");

if (!file)

{

printf("Error opening file: %s\n", filename);

return NULL;

}

ClauseNode \*cnf = (ClauseNode \*)malloc(sizeof(ClauseNode));

cnf->down = NULL;

cnf->right = NULL;

\*max\_var = 0;

char line[11000];

while (fgets(line, sizeof(line), file))

{

if (line[0] == 'c' || line[0] == 'p')

{

continue;

}

ClauseNode \*new\_clause = (ClauseNode \*)malloc(sizeof(ClauseNode));

new\_clause->down = NULL;

new\_clause->right = NULL;

char \*token = strtok(line, " ");

while (token)

{

int lit = atoi(token);

if (lit == 0)

break;

if (abs(lit) > \*max\_var)

\*max\_var = abs(lit);

LiteralNode \*literal = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal->data = lit;

literal->right = new\_clause->right;

new\_clause->right = literal;

token = strtok(NULL, " ");

}

new\_clause->down = cnf->down;

cnf->down = new\_clause;

}

fclose(file);

return cnf;

}

// 修改后的主程序入口

int main(int argc, char \*argv[])

{

if (argc != 3)

{

fprintf(stderr, "Usage: %s <cnf file path> <pickvar choice>\n", argv[0]);

return 1;

}

char cnf\_path[256];

strcpy(cnf\_path, argv[1]);

int pickvar\_choice = atoi(argv[2]);

int truth\_table[5000] = {0};

int max\_var = 0;

ClauseNode \*cnf = CnfParser(truth\_table, cnf\_path, &max\_var);

if (!cnf)

{

fprintf(stderr, "Error reading CNF file: %s\n", cnf\_path);

return 1;

}

clock\_t start\_time = clock();

status result = DpllSolver(cnf, truth\_table, pickvar\_choice);

clock\_t end\_time = clock();

double elapsed\_time = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC \* 1000;

char result\_path[1024];

snprintf(result\_path, sizeof(result\_path), "%s\_faster.res", cnf\_path);

FILE \*output\_file = fopen(result\_path, "w");

if (!output\_file)

{

fprintf(stderr, "Cannot open output file.\n");

return 1;

}

if (result == FOUND)

{

fprintf(output\_file, "s 1\nv ");

for (int i = 0; i < max\_var; i++)

{

if (truth\_table[i] == 1)

fprintf(output\_file, "%d ", i + 1);

else if (truth\_table[i] == 0)

fprintf(output\_file, "-%d ", i + 1);

}

fprintf(output\_file, "\n");

}

else

{

fprintf(output\_file, "s 0\n");

}

fprintf(output\_file, "t %.2f\n", elapsed\_time);

fclose(output\_file);

Print(cnf);

return 0;

}

**12 generate\_diagonal\_sudoku.c（生成数独题目）**

// generate\_diagonal\_sudoku.c

#include "head.h"

int solutionCount; // 全局变量来计数解的数量

// 检查数字 num 能否放置在 grid[row][col]

int isSafe(int grid[N][N], int row, int col, int num)

{

int x, y;

for (x = 0; x < N; x++)

if (grid[row][x] == num || grid[x][col] == num)

return 0;

int startRow = row - row % 3, startCol = col - col % 3;

for (x = 0; x < 3; x++)

for (y = 0; y < 3; y++)

if (grid[x + startRow][y + startCol] == num)

return 0;

// 检查对角线

if (row == col) // 主对角线

for (x = 0; x < N; x++)

if (grid[x][x] == num)

return 0;

if (row + col == N - 1) // 次对角线

for (x = 0; x < N; x++)

if (grid[x][N - x - 1] == num)

return 0;

return 1;

}

// 递归填充数独元素

int fillSudoku(int grid[N][N], int row, int col)

{

if (row == N - 1 && col == N)

return 1;

if (col == N)

{

row++;

col = 0;

}

if (grid[row][col] != UNASSIGNED)

return fillSudoku(grid, row, col + 1);

int num;

for (num = 1; num <= N; num++)

{

if (isSafe(grid, row, col, num))

{

grid[row][col] = num;

if (fillSudoku(grid, row, col + 1))

return 1;

}

grid[row][col] = UNASSIGNED;

}

return 0;

}

// 计算数独的解的数量

void solveWithCount(int grid[N][N], int row, int col)

{

if (row == N - 1 && col == N)

{

solutionCount++;

return;

}

if (col == N)

{

row++;

col = 0;

}

if (grid[row][col] != UNASSIGNED)

{

solveWithCount(grid, row, col + 1);

return;

}

int num;

for (num = 1; num <= N && solutionCount < 2; num++) // 只计算到两个解即可

{

if (isSafe(grid, row, col, num))

{

grid[row][col] = num;

solveWithCount(grid, row, col + 1);

}

grid[row][col] = UNASSIGNED;

}

}

// 验证当前数独是否只有一个解

int isUniqueSolution(int grid[N][N])

{

solutionCount = 0;

solveWithCount(grid, 0, 0);

return solutionCount == 1;

}

// 随机生成第一行

void randomFirstRow(int grid[N][N])

{

int i, j;

int row[N];

srand((unsigned)time(NULL));

for (i = 0; i < N; i++)

{

row[i] = rand() % 9 + 1;

j = 0;

while (j < i)

{

if (row[i] == row[j])

{

row[i] = rand() % 9 + 1;

j = 0;

}

else

j++;

}

}

for (i = 0; i < N; i++)

grid[0][i] = row[i];

}

// 随机生成初盘，确保唯一解

void createStartingGrid(int grid[N][N], int holes)

{

int x, y;

int attempts = 0; // 增加尝试次数计数

while (holes > 0) // 限制最大尝试次数为100

{

x = rand() % N;

y = rand() % N;

if (grid[x][y] == UNASSIGNED)

{

attempts++;

continue; // 如果该位置已经是空格，则跳过

}

int backup = grid[x][y];

grid[x][y] = UNASSIGNED;

if (!isUniqueSolution(grid))

{

grid[x][y] = backup; // 如果不是唯一解，撤销操作

attempts++; // 增加尝试次数

}

else

{

holes--;

}

}

}

// 将语义编码转换为自然编码

int semanticToNatural(int i, int j, int k)

{

return ((i - 1) \* 81 + (j - 1) \* 9 + k);

}

// 将自然编码转换为语义编码

void naturalToSemantic(int code, int \*i, int \*j, int \*k)

{

// 计算 k

\*k = (code % 9 == 0) ? 9 : code % 9;

// 计算 i

\*i = (code - \*k) / 81 + 1;

// 计算 j

\*j = ((code - (\*i - 1) \* 81 - \*k) / 9) + 1;

}

// 生成单元格约束

void addCellConstraints(FILE \*file\_nat)

{

for (int i = 1; i <= 9; i++)

{

for (int j = 1; j <= 9; j++)

{

// 至少填入一个数字的子句

for (int k = 1; k <= 9; k++)

{

fprintf(file\_nat, "%d ", semanticToNatural(i, j, k));

}

fprintf(file\_nat, "0\n");

// 互斥性子句，确保每个单元格只能填一个数字

for (int k1 = 1; k1 <= 8; k1++)

{

for (int k2 = k1 + 1; k2 <= 9; k2++)

{

fprintf(file\_nat, "-%d -%d 0\n", semanticToNatural(i, j, k1), semanticToNatural(i, j, k2));

}

}

}

}

}

// 生成行、列和3x3子盒约束

void addRowColumnBoxConstraints(FILE \*file\_nat)

{

// 行约束和列约束

for (int num = 1; num <= 9; num++)

{

for (int i = 1; i <= 9; i++)

{

// 行约束：每行包含1到9

for (int j = 1; j <= 9; j++)

{

fprintf(file\_nat, "%d ", semanticToNatural(i, j, num));

}

fprintf(file\_nat, "0\n");

// 列约束：每列包含1到9

for (int j = 1; j <= 9; j++)

{

fprintf(file\_nat, "%d ", semanticToNatural(j, i, num));

}

fprintf(file\_nat, "0\n");

// 互斥性约束：行和列中每个数字不重复

for (int j1 = 1; j1 <= 8; j1++)

{

for (int j2 = j1 + 1; j2 <= 9; j2++)

{

fprintf(file\_nat, "-%d -%d 0\n", semanticToNatural(i, j1, num), semanticToNatural(i, j2, num)); // 行互斥

fprintf(file\_nat, "-%d -%d 0\n", semanticToNatural(j1, i, num), semanticToNatural(j2, i, num)); // 列互斥

}

}

}

}

// 3x3子盒约束

for (int num = 1; num <= 9; num++)

{

for (int boxRow = 0; boxRow < 3; boxRow++)

{

for (int boxCol = 0; boxCol < 3; boxCol++)

{

for (int i = 1; i <= 3; i++)

{

for (int j = 1; j <= 3; j++)

{

fprintf(file\_nat, "%d ", semanticToNatural(boxRow \* 3 + i, boxCol \* 3 + j, num));

}

}

fprintf(file\_nat, "0\n");

// 3x3 子盒的唯一性约束

for (int i1 = 1; i1 <= 3; i1++)

{

for (int j1 = 1; j1 <= 3; j1++)

{

for (int i2 = i1; i2 <= 3; i2++)

{

for (int j2 = (i1 == i2 ? j1 + 1 : 1); j2 <= 3; j2++)

{

fprintf(file\_nat, "-%d -%d 0\n", semanticToNatural(boxRow \* 3 + i1, boxCol \* 3 + j1, num), semanticToNatural(boxRow \* 3 + i2, boxCol \* 3 + j2, num));

}

}

}

}

}

}

}

}

// 生成对角线约束子句

void addDiagonalConstraints(FILE \*file\_nat)

{

// 主对角线约束

for (int num = 1; num <= 9; num++)

{

for (int i = 1; i <= 9; i++)

{

fprintf(file\_nat, "%d ", semanticToNatural(i, i, num));

}

fprintf(file\_nat, "0\n");

// 主对角线互斥性约束

for (int i = 1; i <= 9; i++)

{

for (int j = i + 1; j <= 9; j++)

{

fprintf(file\_nat, "-%d -%d 0\n", semanticToNatural(i, i, num), semanticToNatural(j, j, num));

}

}

}

// 次对角线约束

for (int num = 1; num <= 9; num++)

{

for (int i = 1; i <= 9; i++)

{

fprintf(file\_nat, "%d ", semanticToNatural(i, 10 - i, num));

}

fprintf(file\_nat, "0\n");

// 次对角线互斥性约束

for (int i = 1; i <= 9; i++)

{

for (int j = i + 1; j <= 9; j++)

{

fprintf(file\_nat, "-%d -%d 0\n", semanticToNatural(i, 10 - i, num), semanticToNatural(j, 10 - j, num));

}

}

}

}

void createSudokuToCNF(const char \*semanticFile, const char \*naturalFile, int holes)

{

int grid[N][N] = {0};

randomFirstRow(grid); // 随机生成第一行

fillSudoku(grid, 1, 0); // 递归生成后续行

printf("Initial Sudoku Grid after filling:\n");

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

printf("%d ", grid[i][j]);

}

printf("\n");

}

// 生成初盘，确保唯一解

createStartingGrid(grid, holes);

printf("Sudoku Grid after creating starting grid with holes:\n");

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

printf("%d ", grid[i][j]);

}

printf("\n");

}

// 输出为语义编码的CNF文件

FILE \*file\_sem = fopen(semanticFile, "w");

if (!file\_sem)

{

printf("无法创建语义输出文件: %s\n", semanticFile);

return;

}

// 输出为自然编码的CNF文件

FILE \*file\_nat = fopen(naturalFile, "w");

if (!file\_nat)

{

printf("无法创建自然输出文件: %s\n", naturalFile);

return;

}

for (int row = 0; row < N; row++)

{

for (int col = 0; col < N; col++)

{

if (grid[row][col] != UNASSIGNED)

{

fprintf(file\_sem, "%d%d%d 0\n", row + 1, col + 1, grid[row][col]);

int natural\_encoded = semanticToNatural(row + 1, col + 1, grid[row][col]);

fprintf(file\_nat, "%d 0\n", natural\_encoded);

}

}

}

// 添加数独约束子句

addCellConstraints(file\_nat);

addRowColumnBoxConstraints(file\_nat);

addDiagonalConstraints(file\_nat);

fclose(file\_sem);

fclose(file\_nat);

printf("数独已生成并保存至: %s 和 %s\n", semanticFile, naturalFile);

}

int main(int argc, char \*argv[])

{

if (argc != 4)

{

printf("Usage: %s <difficulty\_level> <semantic\_output.cnf> <natural\_output.cnf>\n", argv[0]);

return 1;

}

int difficulty = atoi(argv[1]);

const char \*semanticFile = argv[2];

const char \*naturalFile = argv[3];

// 根据难度生成挖洞数目

int maxHoles[] = {20, 30, 40, 50};

int maxRange = 5;

if (difficulty < 1 || difficulty > 4)

{

printf("Difficulty level should be between 1 and 4.\n");

return 1;

}

srand((unsigned)time(NULL));

int holes = maxHoles[difficulty - 1] - (rand() % (maxRange \* 2 + 1) - maxRange);

createSudokuToCNF(semanticFile, naturalFile, holes);

return 0;

}

**13 sudoku\_solver.c（利用DPLL算法求解数独）**

// sudoku\_solver.c

#include "head.h"

// DPLL 求解器

status DpllSolver(ClauseNode \*s, int \*truth\_table)

{

ClauseNode \*s\_tmp = s;

ClauseNode \*unit\_clause = IsUnitClause(s\_tmp);

printf("Starting DPLL solver...\n");

while (unit\_clause)

{

RecordTruth(unit\_clause, truth\_table);

int var = unit\_clause->right->data;

printf("Found unit clause with variable %d\n", var);

RemoveVar(s, var);

if (s->down == NULL)

{

printf("Solution FOUND\n");

return FOUND;

}

else if (IsEmptyClause(s))

{

printf("Empty clause detected, returning NOTFOUND\n");

return NOTFOUND;

}

s\_tmp = s;

unit\_clause = IsUnitClause(s\_tmp);

}

int var = PickVar(s);

printf("Picking variable %d for branching...\n", var);

if (DpllSolver(AddClause(CopyS(s), var), truth\_table))

{

printf("Branch with variable %d found a solution.\n", var);

return FOUND;

}

printf("Trying branch with variable %d negated...\n", var);

return DpllSolver(AddClause(s, -var), truth\_table);

}

// 添加子句至子句集

ClauseNode \*AddClause(ClauseNode \*s, int var)

{

ClauseNode \*c\_tmp = (ClauseNode \*)malloc(sizeof(ClauseNode));

c\_tmp->right = (LiteralNode \*)malloc(sizeof(LiteralNode));

c\_tmp->right->data = var;

c\_tmp->right->right = NULL;

c\_tmp->down = s->down;

s->down = c\_tmp;

return s;

}

// 判断是否存在单子句

ClauseNode \*IsUnitClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s;

while (c\_tmp)

{

if (c\_tmp->right && !c\_tmp->right->right)

return c\_tmp;

c\_tmp = c\_tmp->down;

}

return NULL;

}

// 复制子句集

ClauseNode \*CopyS(ClauseNode \*s)

{

ClauseNode \*s\_new, \*c\_tmp, \*s\_origin = s->down;

s\_new = (ClauseNode \*)malloc(sizeof(ClauseNode));

s\_new->right = NULL;

c\_tmp = s\_new;

while (s\_origin)

{

c\_tmp->down = CopyClause(s\_origin);

c\_tmp = c\_tmp->down;

s\_origin = s\_origin->down;

}

return s\_new;

}

// 复制单个子句

ClauseNode \*CopyClause(ClauseNode \*s)

{

ClauseNode \*clause\_tmp = (ClauseNode \*)malloc(sizeof(ClauseNode));

LiteralNode \*literal\_tmp, \*s\_tmp = s->right;

clause\_tmp->down = NULL;

clause\_tmp->right = NULL;

if (s\_tmp)

{

literal\_tmp = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal\_tmp->data = s\_tmp->data;

literal\_tmp->right = NULL;

s\_tmp = s\_tmp->right;

clause\_tmp->right = literal\_tmp;

}

while (s\_tmp)

{

literal\_tmp->right = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal\_tmp->right->data = s\_tmp->data;

literal\_tmp->right->right = NULL;

literal\_tmp = literal\_tmp->right;

s\_tmp = s\_tmp->right;

}

return clause\_tmp;

}

// 删除子句

ClauseNode \*DeleteClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s;

LiteralNode \*l;

while (s->right)

{

l = s->right;

s->right = s->right->right;

free(l);

}

s = s->down;

free(c\_tmp);

return s;

}

// 删除负变元

ClauseNode \*DeleteLiteral(ClauseNode \*s, int var)

{

LiteralNode \*l = s->right, \*l\_tmp;

if (l->data == -var)

{

l\_tmp = s->right->right;

free(l);

s->right = l\_tmp;

return s;

}

else if (l->data == var)

{

return NULL;

}

while (l)

{

if (l->right && l->right->data == -var)

{

l\_tmp = l->right;

l->right = l->right->right;

free(l\_tmp);

}

else if (l->right && l->right->data == var)

{

return NULL;

}

l = l->right;

}

return s;

}

// 判断是否存在空子句

status IsEmptyClause(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s->down;

while (c\_tmp)

{

if (!c\_tmp->right)

return YES;

c\_tmp = c\_tmp->down;

}

return NO;

}

// 记录单子句中元素的真值

status RecordTruth(ClauseNode \*s, int \*truth\_table)

{

int idx = abs(s->right->data) - 1;

if (s->right->data > 0)

truth\_table[idx] = 1;

else

truth\_table[idx] = 0;

return OK;

}

// 删除变元所在子句与负变元

status RemoveVar(ClauseNode \*s, int var)

{

ClauseNode \*c\_tmp = s->down, \*last\_c\_tmp = s;

while (c\_tmp)

{

c\_tmp = DeleteLiteral(c\_tmp, var);

if (c\_tmp == NULL)

{

c\_tmp = DeleteClause(last\_c\_tmp->down);

last\_c\_tmp->down = c\_tmp;

continue;

}

if (c\_tmp == NULL)

break;

last\_c\_tmp = c\_tmp;

c\_tmp = c\_tmp->down;

}

return OK;

}

// 选取一个变元

int PickVar(ClauseNode \*s)

{

int var = s->down->right->data;

return var;

}

// 打印子句集

status Print(ClauseNode \*s)

{

ClauseNode \*c\_tmp = s->down;

LiteralNode \*l\_tmp;

printf("\n-----Start-----\n");

while (c\_tmp)

{

l\_tmp = c\_tmp->right;

while (l\_tmp)

{

printf("%d ", l\_tmp->data);

l\_tmp = l\_tmp->right;

}

if (c\_tmp->right)

printf("\n");

else

printf("空\n");

c\_tmp = c\_tmp->down;

}

printf("------End------\n");

return OK;

}

// 解析 CNF 文件

ClauseNode \*CnfParser(int \*literal\_num, char \*filename, int \*max\_var)

{

FILE \*file = fopen(filename, "r");

if (!file)

{

printf("Error opening file: %s\n", filename);

return NULL;

}

ClauseNode \*cnf = (ClauseNode \*)malloc(sizeof(ClauseNode));

cnf->down = NULL;

cnf->right = NULL;

\*max\_var = 0;

char line[11000];

while (fgets(line, sizeof(line), file))

{

if (line[0] == 'c' || line[0] == 'p')

{

continue;

}

ClauseNode \*new\_clause = (ClauseNode \*)malloc(sizeof(ClauseNode));

new\_clause->down = NULL;

new\_clause->right = NULL;

char \*token = strtok(line, " ");

while (token)

{

int lit = atoi(token);

if (lit == 0)

break;

if (abs(lit) > \*max\_var)

\*max\_var = abs(lit); // 记录最大变元编号

LiteralNode \*literal = (LiteralNode \*)malloc(sizeof(LiteralNode));

literal->data = lit;

literal->right = new\_clause->right;

new\_clause->right = literal;

token = strtok(NULL, " ");

}

new\_clause->down = cnf->down;

cnf->down = new\_clause;

}

fclose(file);

return cnf;

}

// 将自然编码转换为语义编码

void naturalToSemantic(int code, int \*i, int \*j, int \*k)

{

// 计算 k

\*k = (code % 9 == 0) ? 9 : code % 9;

// 计算 i

\*i = (code - \*k) / 81 + 1;

// 计算 j

\*j = ((code - (\*i - 1) \* 81 - \*k) / 9) + 1;

}

// 主程序入口

int main(int argc, char \*argv[])

{

if (argc != 2)

{

fprintf(stderr, "Usage: %s <cnf file path>\n", argv[0]);

return 1;

}

char cnf\_path[256];

strcpy(cnf\_path, argv[1]);

int truth\_table[N \* N \* N] = {0}; // 调整为适应9x9数独

int max\_var = 0;

ClauseNode \*cnf = CnfParser(truth\_table, cnf\_path, &max\_var);

if (!cnf)

{

fprintf(stderr, "Error reading CNF file: %s\n", cnf\_path);

return 1;

}

status result = DpllSolver(cnf, truth\_table);

// 打开语义编码的输出文件

FILE \*output\_file = fopen("solution.cnf", "w"); // 固定输出文件路径

if (!output\_file)

{

fprintf(stderr, "Cannot open output file.\n");

return 1;

}

// 打开自然编码的输出文件

FILE \*natural\_output\_file = fopen("natural\_solution.cnf", "w");

if (!natural\_output\_file)

{

fprintf(stderr, "Cannot open natural output file.\n");

fclose(output\_file);

return 1;

}

if (result == FOUND)

{

for (int i = 0; i < N \* N \* N; i++)

{

if (truth\_table[i] == 1)

{

// 将自然编码转换为语义编码

int row, col, num;

naturalToSemantic(i + 1, &row, &col, &num);

// 写入语义编码文件

fprintf(output\_file, "%d%d%d 0\n", row, col, num);

// 写入自然编码文件

fprintf(natural\_output\_file, "%d 0\n", i + 1);

}

}

}

else

{

fprintf(output\_file, "s 0\n");

fprintf(natural\_output\_file, "s 0\n");

}

fclose(output\_file);

fclose(natural\_output\_file);

Print(cnf);

return 0;

}

**14 cnf\_to\_grid.c（将cnf文件转换为数独格局）**

//cnf\_to\_grid.c

#include "head.h"

// 初始化数独网格

void initializeGrid(int grid[N][N])

{

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

grid[i][j] = 0;

}

// 从CNF文件读取并转换为数独网格

int readCNFToGrid(const char \*cnfFile, int grid[N][N])

{

FILE \*file = fopen(cnfFile, "r");

if (!file)

{

fprintf(stderr, "Error opening CNF file: %s\n", cnfFile);

return 1;

}

char line[256];

while (fgets(line, sizeof(line), file))

{

if (line[0] == 'c' || line[0] == 'p')

continue; // 跳过注释和头信息行

int lit;

sscanf(line, "%d", &lit);

if (lit > 0)

{ // 确保是正文字

int row = (lit / 100) - 1;

int col = ((lit / 10) % 10) - 1;

int num = (lit % 10);

grid[row][col] = num;

}

}

fclose(file);

return 0;

}

// 打印数独网格到文件

void printGridToFile(int grid[N][N], const char \*outputFile)

{

FILE \*file = fopen(outputFile, "w");

if (!file)

{

fprintf(stderr, "Error creating output file: %s\n", outputFile);

return;

}

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

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

fprintf(file, "0 ");

else

fprintf(file, "%d ", grid[i][j]);

}

fprintf(file, "\n");

}

fclose(file);

}

int main(int argc, char \*argv[])

{

if (argc != 3)

{

fprintf(stderr, "Usage: %s <input\_file.cnf> <output\_file.txt>\n", argv[0]);

return 1;

}

const char \*cnfFile = argv[1];

const char \*outputFile = argv[2];

int grid[N][N];

initializeGrid(grid);

if (readCNFToGrid(cnfFile, grid) != 0)

{

return 1;

}

printGridToFile(grid, outputFile);

printf("Puzzle parsed and grid generated, saved to: %s\n", outputFile);

return 0;

}