華中科技大學

课程名称： 程序设计综合课程设计

设计题目： 基于 SAT 的百分号数独游戏求解程序

专业班级 CS2407

学　　号 U202414789

姓　　名 史梓洋

指导教师 李丹

报告日期 2025 年 9 月 14 日

计算机科学与技术学院

目　　录

任务书

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。

设计要求

1. 输入输出功能：程序执行参数输入，CNF 文件读取，结果输出与保存（15%）。
2. 公式解析与验证：建立内部表示，并能逐行打印验证解析正确性（15%）。
3. DPLL 过程：基于 DPLL 框架实现 SAT 求解（35%）。
4. 时间性能测量：记录 DPLL 执行时间（毫秒）（5%）。
5. 程序优化：在数据结构或分支策略等方面优化，并给出优化率 $[(t-t_o)/t]\times 100\%$（15%）。
6. SAT 应用：将数独问题归约为 SAT，集成求解并具备简单交互（15%）。

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. Concordia University, 2009
3. 陈稳. 基于 DPLL 的 SAT 算法的研究与应用. 电子科技大学, 2011
4. Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. JAR (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. LPAR 2005.
7. Inês Lynce, Joel Ouaknine. Sudoku as a SAT Problem. AIMATH 2006.
8. Uwe Pfeiffer et al. A Sudoku-Solver for Large Puzzles using SAT. EPiC 13.

9. Sudoku Puzzles Generating: from Easy to Evil. http://zhangroup.aporc.org/images/files/Paper_3485.pdf
10. 薛源海等. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识, 2009
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, 2015

1 引言

1.1 课题背景与意义

SAT（Satisfiability）问题是判定给定命题公式是否存在使其为真的赋值的判定问题。自 Cook–Levin 定理表明经典 NP 完全性以来，SAT 被广泛作为诸多组合优化与验证问题的统一建模平台。与传统的“为具体问题手写算法”相比，SAT 的优势在于统一性、复用性与工程可行性。

本项目面向教学与实践，目标是从零实现一个可用的 DPLL 求解器，覆盖 CNF 解析、内部结构、DPLL 核心流程、分支启发式与性能度量，并将数独问题完整地归约到 SAT 进行求解，构成“理论–实现–应用”的闭环。

从理论谱系上看，SAT 是第一个被证明为 NP 完全的问题，具有“归约枢纽”的地位。Davis–Putnam（DP）消去法与后续 Davis–Logemann–Loveland（DPLL）算法开启了以“系统化搜索 + 逻辑传播”为核心的判定方法：

- DP 强调基于分辨率与变量消去，而 DPLL 基于“递归二分 + 单子句传播 + 回溯”的搜索框架；
- 现代 CDCL（Conflict-Driven Clause Learning）在 DPLL 基础上加入冲突分析、学习子句、重启策略与相位保存，已成为工业求解的事实标准；
- 尽管 CDCL 更强，但 DPLL 仍是理解 SAT 求解器内部机制的“最小充分内核”：传播、分支、回溯、数据结构的相互作用决定了搜索效率的数量级差异。

从应用视角看，SAT 的“通用建模”能力极强：

- 硬件设计与形式化验证：等价性检查（CEC）、有界模型检测（BMC）、时序属性验证等将电路/状态机转化为可满足性问题；
- 软件工程：路径可达性、缺陷定位、程序合成中的候选验证；
- 规划与运筹：经典 AI 规划、日程排班、资源分配；
- 密码与安全：差分/线性密码分析中的约束求解、协议栈安全属性验证；
- 组合设计与解谜：数独、Kakuro、拉丁方、图着色等均可自然归约。

以数独为代表的离散谜题提供了一个直观、闭环的示范平台：从约束建模（行/列/宫/对角/窗口）、到 CNF 编码、到 SAT 求解、再到解的还原与可视化，完

整串联了“**问题** → **建模** → **求解** → **验证**”的工程流程。相比直接写一个“特定于数独”的回溯求解器，基于 SAT 的方案更具通用性：一旦编码到 CNF，后续就可以复用更先进的求解技术，无需为每个新问题重新开发搜索引擎。

本项目聚焦 DPLL 的教学实现，意义体现在：

1. **理解本质：**用最小内核呈现 SAT 求解的关键要素（传播/回溯/分支），厘清复杂工程优化背后的“底层语义”；
2. **体系方法：**从文件解析到内部存储、从启发式到计时评测，走通一条可复现的工程路径；
3. **以点带面：**通过数独应用，示范“SAT 作为统一求解平台”的建模范式，为进一步拓展到 EDA/验证/规划等打基础；
4. **实验平台：**为后续替换数据结构（如 watch-lists）、加入学习与重启、切换启发式提供对照基线。

在复杂度层面，DPLL 的最坏时间仍为指数级，但**数据结构**（如出现表 vs 双 watched literals）、**启发式**（如 MOMS/DLIS/VSIDS）与**实现细节**（撤销策略、内存局部性）会显著改变平均性能。基线 DPLL 的价值，不在于“战胜工业级 CDCL”，而在于提供一个可控、可解释、可扩展的试验田。

1.2 国内外研究现状

工业级 CDCL 求解器（MiniSAT、Glucose、Kissat、Maple 系列、CaDiCaL 等）采用冲突分析（1-UIP 学习）、双 watched literals、LBD/活动度、重启策略等，已形成成熟工程体系。学术界持续探索更好的启发式、学习子句管理、并行化与领域特定编码。本项目聚焦 DPLL 基础版本，兼顾可读性与可验证性。

发展脉络与关键技术 经典求解发展可以粗略分段：GRASP 引入系统性的冲突分析；Chaff 推广 VSIDS 与 watched-literals，将传播与启发式的代价显著降低；MiniSAT 用极简实现奠定教学与研究基础；Glucose 引入 LBD 度量指导学习子句管理；Maple/MapleLCMDistAdapt 等对活动度与重启做出新探索；Kissat、CaDiCaL 在实现层面深度打磨（例如缓存友好、分支预测、剔除分支、简化内存布局），进一步提升了稳定性与速度。并行化方面，ManySAT、plingeling 等基于多起点/多策略竞争与子句交换；增量 SAT 与 Assumptions 技术支撑了模型检测、渐进求解等应用场景。预处理/处理中（pre/in-processing）方面，SatELite 及其后

续工作系统化了变量消去、子句子集/子句吸收（subsumption/absorption）、阻塞子句消去（BCE）、子句活化（vivification）等流程。

编码与传播优化 针对结构化约束（如基数/伪布尔/XOR），不同编码在子句规模、传播强度与求解器兼容性间做权衡：

- 基数约束：序列（sequential）与多计数器（cardinality networks）编码在规模与传播性上各有优势；
- 伪布尔：基于 PB 到 CNF 的传播保持/近似保持编码，或通过混合 SAT+PB 处理；
- XOR：高斯消元与 CNF 混合处理可显著提升密码学实例的求解能力。

应用与评测 SAT Competition/SAT Race 提供了公平的评测场，追踪了近二十年来的技术演化。应用层面，SAT 已渗透到 EDA、软件验证、规划调度、AI 推理、组合搜索等核心领域。国内高校与研究机构在教学实现、工具迁移与特定领域应用方面也有持续贡献。

本项目定位 我们选择以 DPLL 为主干，刻意约束工程复杂度，以凸显“传播–分支–回溯”与“数据结构–启发式”之间的因果关系；同时在应用上选择数独，兼顾可视化与可验证性，为后续进一步接入 watch-lists、CDCL 与 GUI 留出空间。

1.3 课程设计的主要研究工作

- 实现 CNF 解析与内部结构；
- 设计出现表与变更栈，支持高效传播与撤销；
- 实现 DPLL：单子句传播、回溯、分支策略；
- 设计两种分支策略并对比耗时，输出优化率；
- 实现数独生成、挖洞保唯一、CNF 归约与解还原；
- 提供控制台交互与结果落盘。

为确保“可复现、可对比、可复用”，我们进一步细化以下工作要点：

1. **解析与验证：** 对 DIMACS 解析进行两遍扫描（统计 + 构造），支持 c/p 行、空白与尾随零的鲁棒处理；提供 `print_cnf` 逐子句打印，便于人工对照验证；
2. **内部结构：** 采用“按变量分桶”的正/负极性出现表以降低遍历开销；对子句维持 `current_length` 与 `assignment_status`，支持 O(1) 更新与检查；

3. **撤销机制：**以 `change_stack+change_counts[depth][..]` 记录本层操作，按“先 SHRUNK 后 SATISFIED”的顺序精确回滚，避免额外全表扫描；
4. **启发式对比：**method=1（近似 MOMS）聚焦最短子句参与度，method=2 为顺序基线；以相同算例对比两者时延并计算优化率；
5. **数独编码：**在标准行/列/宫约束基础上，加入一条对角线与两个 3×3 窗口的扩展约束（与源码一致），并输出题面单子句；
6. **结果格式：**统一输出 `.res`（s/v/t），其中时间以毫秒计，便于脚本化统计；
7. **交互与演示：**提供控制台菜单串联“生成–挖洞–归约–求解–还原–展示/交互”，方便课堂演示与验收；同时在数独解答的时候接入了 GUI 的接口，使用 Qt 框架，允许用户打开 Qt 框架所创建的图形化游戏界面，使得用户得到更好的交互体验
8. **扩展空间：**代码层面预留了切换启发式、替换数据结构与接入 GUI 的接口，便于后续升级到 watch-lists 与 CDCL。

2 系统需求分析与总体设计

2.1 系统需求分析

功能性需求：

- 读取 CNF：支持 c 注释、p 头行、子句行；
- 解析验证：可选择打印 CNF 以用于人工对照；
- SAT 求解：基于 DPLL，支持两种分支策略；
- 时间统计：返回并打印耗时，写入.res；
- 结果保存：生成“.res”（s/v/t）；
- 数独应用：生成题面、归约 CNF、调用求解、解还原、交互编辑；
- 菜单交互：生成数独/求解 CNF/展示结果/交互模式。

非功能性需求：正确性、可维护、复用性与基本性能。

使用者与场景 主要面向课程学习者与助教评阅：学习者可通过不同算例观察启发式差异；助教可据 `.res` 检查结果与时间。典型场景包括：(*i*) 读取给定 CNF 并求解；(*ii*) 生成数独并通过 SAT 求解验证唯一性；(*iii*) 手动交互填写数独并观察解的变化。

接口与输入输出 输入：CNF 文件路径或菜单选择；输出：控制台日志与 `.res` 文件。出错处理：文件不存在/解析失败/格式不合法时给出可读性提示；当 p 行与实际子句数不一致时自动以实际计数回写内存结构并警告。

约束与兼容 目标平台 Windows/MinGW；内存与时间预算以教学算例为主；代码采用 C/C++ 标准库与最小依赖，Qt GUI 为可选项（默认控制台）。

性能与度量 时间以 `clock()` 采集；为增强可比性，建议同一算例重复多次取中位数；优化率定义为

$$extOptimization(\%) = \frac{t_{base} - t_{opt}}{t_{base}} \times 100\%,$$

其中 t_{base} 为基线策略耗时，t_{opt} 为优化策略耗时（源码中以 method=2 为基线、method=1 为优化进行对照）。

可测试性与可维护性 通过 `print_cnf` 实现解析验证；`.res` 输出统一格式便于脚本检查；模块化划分（解析/求解/应用）降低耦合；核心常量集中在 `definition.h`，便于调整规模上限。

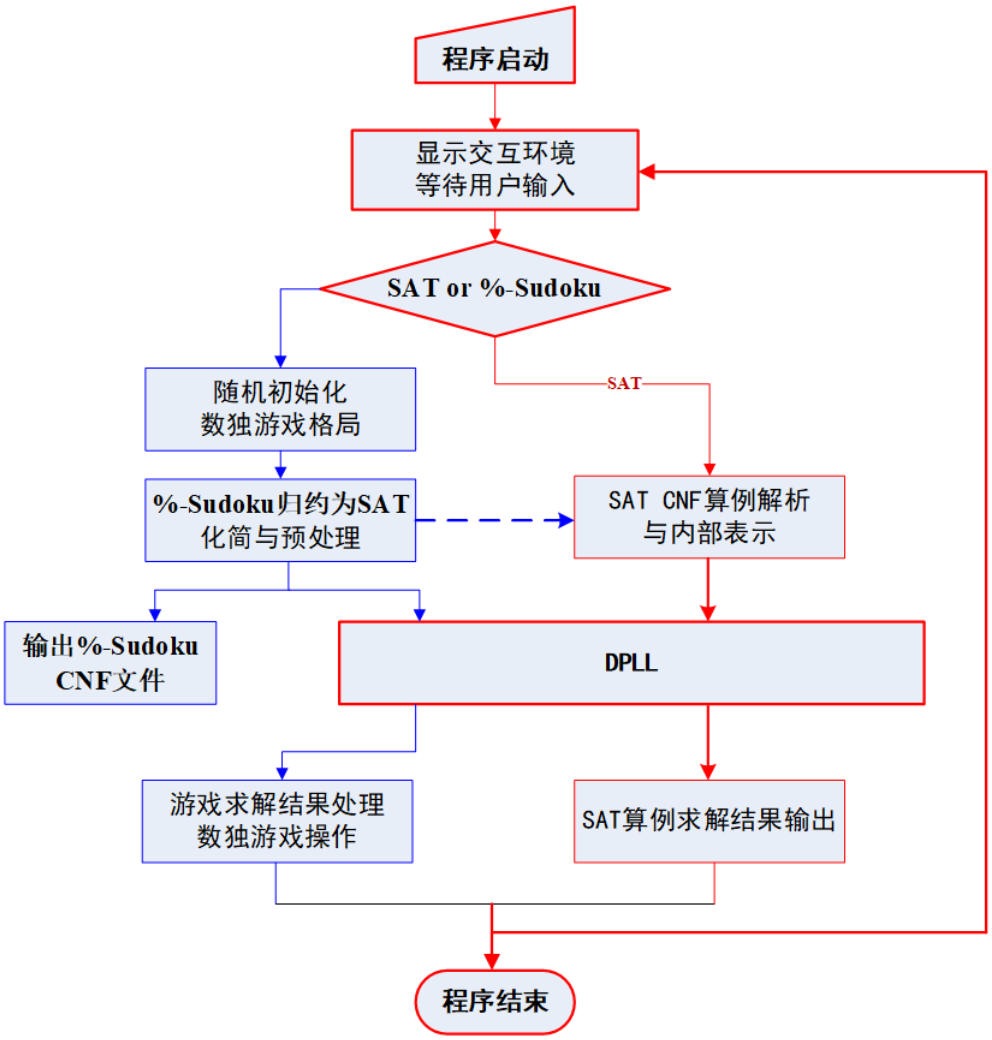

图 2-1　程序流程图

2.2　系统总体设计

模块划分：I/O 模块、解析模块、DPLL 模块、计时与输出模块、数独模块。流程示意（以 mermaid 伪码呈现）：

总体架构与分层　系统采用“**界面层（CLI/可选 GUI）–应用层（数独）–求解内核（SAT）**”的三层逻辑：

- **界面层**提供菜单、参数获取与基础 I/O；
- **应用层**负责数独的生成、挖洞、归约与还原；
- **求解内核**负责 CNF 解析、DPLL 搜索、时间统计与结果写回。

关键模块职责

- 解析模块（`read_cnf_file/print_cnf`）：两遍扫描构建子句与出现表；

- 传播与撤销（assign_value/unassign_value）：维护子句满足/收缩状态与单子句堆；
- 分支（select_branching_variable/select_basic_method）：在性能与实现复杂度之间权衡；
- 主过程（dpll/satsolver）：组织预处理、递归、计时与结果输出；
- 数独（createFullGrid/digHoles/writeSudokuToCNF/CnftoSudoku）：串联题面与求解内核。

数据与控制流 CNF 管线按照“文件 → 解析 → 内部结构 → DPLL → .res”推进；数独管线按照“生成 → 挖洞 → 归约 → SAT → 还原/展示”推进。两个管线在 satsolver 处汇合，形成复用。

相关接口

- int read_cnf_file(char* fn)：前置 fn 可读；后置内存结构一致且计数正确；失败返回 0；
- int dpll()：在给定内部状态下返回 {SAT, UNSAT}，并保证所有局部赋值得到正确撤销；
- void write_result(int sat,double t,char* fn)：生成 .res，其中 v 行长度等于变量数、t 为毫秒；
- void writeSudokuToCNF(...)：输出子句头与正文一致，回填 p cnf 的子句计数。

可扩展性 在不改变对外接口的前提下，可以：（*i*）将出现表替换为双 watched literals；（*ii*）引入学习与重启升级为 CDCL；（*iii*）以编译开关切换启发式或添加新的启发式；（*iv*）接入 Qt GUI 展示数独交互与动画传播效果。

3 系统详细设计

3.1 数据结构的定义与关系

核心结构：Clause（literals、assignment_status、original_length、current_length、is_satisfied、unit_literal）、LiteralOccurrence/OccurrenceList、ChangeRecord、VariableResult，以及全局的单子句栈、变更栈与分层计数、出现表与标记数组等。设计要点：出现表便于按出现次数更新；change_counts 按层统计以 O(变更数) 撤销；unit_literal 有利于撤销时清理 in_unit 标记。详细数据结构定义与关系如下

3.1.1 SAT/DPLL 核心数据结构（C 代码）

```c
#define MAX_VARS     100000
#define MAX_CLAUSES 1000000

/* 子句：DPLL 操作的核心对象 */
typedef struct {
  int *literals;           // 子句中的文字数组
  int *assignment_status; // 每个文字是否已被赋值
  int original_length;     // 初始长度
  int current_length;      // 当前未被赋值的文字数
  int is_satisfied;        // 子句是否已满足
  int unit_literal;        // 若为单子句时的唯一未赋值文字
} Clause;

/* 文字在子句中的一次出现位置 */
typedef struct {
  int clause_index;        // 子句编号
  int literal_position;    // 在该子句中的位置
} LiteralOccurrence;

/* 某变量（正/负极性）的出现表 */
typedef struct {
  LiteralOccurrence *list; // 出现列表
```

```
23    int count;                // 出现次数
24    int capacity;             // 分配容量
25  } LiteralOccurrenceList;
26
27  /* 回溯撤销记录 */
28  typedef struct { int clause_index, literal_position; }
        ChangeRecord;
29
30  /* 变量赋值结果 */
31  typedef struct { int value; } VariableResult; // TRUE/FALSE/
        UNASSIGNED
32
33  /* 全局(或模块内全局)状态 */
34  extern int num_vars, original_formula_length,
        current_formula_length, max_clause_length;
35  extern Clause *clauses;
36
37  extern LiteralOccurrenceList *pos_literals, *neg_literals; // 正/
        负极性出现表
38  extern int *in_unit_pos, *in_unit_neg;                    // 是
        否在单子句堆
39
40  extern VariableResult results[MAX_VARS + 1];
41
42  extern int unit_clause_stack[MAX_CLAUSES]; // 单子句堆
43  extern int unit_stack_size;
44
45  extern ChangeRecord change_stack[MAX_CLAUSES * 10]; // 撤销栈
46  extern int change_stack_top;                        // 撤销栈顶
47  extern int change_counts[MAX_VARS * 2][2];          // [depth][
        SATISFIED/SHRUNK]
48  extern int depth;                                   // 当前搜索
        深度
```

3.1.2 数独相关数据结构（C 代码）

```c
#define SIZE 9
#define BOX_SIZE 3

/* 数独网格与派生网格 */
extern int grid[SIZE][SIZE];          // 当前完整/题面网格
extern int playerGrid[SIZE][SIZE];    // 玩家交互网格(可改动)
extern int entireGrid[SIZE][SIZE];    // 生成的完整解备份
extern int initplaygrid[SIZE][SIZE];// 初始题面备份
extern int solvedGrid[SIZE][SIZE];    // 由 .res 还原出的解

/* 变量映射: X(i,j,k)   (i-1)*81+(j-1)*9+k */
static inline int ChangetoLiteral(int row, int col, int num) {
  return (row-1)*81 + (col-1)*9 + num; // 1-based 参数
}
```

3.1.3 作用与关系说明

Clause（子句） 存储每个子句的文字、赋值状态与满足/收缩信息，是传播与回溯的直接载体。current_length 递减至 1 触发单子句；为 0 且未满足即产生冲突。

LiteralOccurrence/List（出现表） 对每个变量分别维护正负极性的出现位置集合。传播时：满足含 l 的子句、收缩含 $\neg l$ 的子句——都可通过出现表在 O(出现次数) 内完成，避免全式扫描；启发式统计（最短子句参与度）亦复用该表。

ChangeRecord/Stacks（撤销） 所有对 clauses 的修改（满足/收缩）都会以 {c,-1}/{c,pos} 形式压入 change_stack，并在 change_counts[depth][..] 计数。回溯时按“先 SHRUNK 后 SATISFIED”精确逆操作，恢复上一层一致状态。

VariableResult（变量赋值） 记录每个变量当前取值，分支与输出都依赖该结构；对局部赋值，在递归退出时须与撤销同步恢复为 UNASSIGNED。

unit_clause_stack 与 in_unit 标记 维护当前待处理的单子句文字集合，并用 in_unit_pos/neg 标记避免重复入堆、快速检测同一变量正/负同时入堆的矛盾，触发回溯。

depth 与 change_counts 按搜索层次分组记录变更，配合 LIFO 的 change_stack 实现 O(变更数) 的回溯撤销，避免对子句与出现表做额外扫描。

数独网格族（grid/playerGrid/entireGrid/initplaygrid/solvedGrid）

- grid：生成完整盘面后挖洞得到题面，后续用于 CNF 归约；
- playerGrid：控制台交互写入/清除数字，不影响 SAT 求解流程；
- entireGrid：生成时的完整解备份，可用于对照验证；
- initplaygrid：题面初始快照，用于重置交互或对比；
- solvedGrid：从 .res 的 v 行正文字还原出的最终解。

数独 → SAT 的数据流 grid(题面)→writeSudokuToCNF 生成 CNF(含行/列/宫/对角/窗口与题面单子句) → SAT/DPLL 得到 .res → CnftoSudoku 将正文字映回 solvedGrid 并展示/校验。

```
关 系 总 览 ( 简 化 )

 CNF(.cnf) --> read_cnf_file --> clauses[] + pos/neg_literals[]
                  |             \
                  |              +--> unit_clause_stack
                  |                    ^      |
                  v                    |      v
               assign/unassign <---- change_stack(+counts)
                  |
                  v
                dpll() ----> results[] ----> .res

 Sudoku: grid --> writeSudokuToCNF --> .cnf --> SAT --> .res -->
    CnftoSudoku --> solvedGrid
```

3.2 DPLL 算法设计与推导

算法不变式 在递归深度 d 的任何时刻，保持以下不变式：

- I1（子句一致性）：对每个子句 C，若存在取真文字则 C.is_satisfied $= 1$ 且 C.current_length 可不等于 original_length；若未满足，则 C 中被赋值为假或与赋值冲突的文字被标记为已用，current_length 等于未赋值文字个数；

- I2（单子句队）：`unit_clause_stack` 中仅包含“尚未处理”的单子句文字，且 `in_unit_pos/neg` 防止重复入队；
- I3（撤销完备）：`change_stack` 记录了自深度 d 以来的所有子句状态改变，`change_counts[d][..]` 按类型计数；
- I4（赋值一致）：`results` 数组与子句/出现表的修改保持一致，且不会同时存在 x 与 $\neg x$ 入单子句队的情况；若出现则立即判冲突。

核心流程与伪代码

```
// 传播：将文字 lit 赋真并进行单子句传播，返回是否冲突
bool assign_value(int lit) {
  int var = abs(lit); int isPos = (lit > 0);
  if (results[var].value != 0) return results[var].value == (
     isPos ? 1 : -1);
  results[var].value = isPos ? 1 : -1;

  // 满足包含 lit 的子句
  LiteralOccurrenceList *satList = isPos ? &pos_literals[var] : &
     neg_literals[var];
  for (int i = 0; i < satList->count; ++i) {
    int c = satList->list[i].clause_index;
    if (!clauses[c].is_satisfied) {
      clauses[c].is_satisfied = 1;
      change_stack[change_stack_top++] = (ChangeRecord){c, -1};
      change_counts[depth][0]++; // SATISFIED
    }
  }

  // 收缩包含 ~lit 的子句
  LiteralOccurrenceList *shrList = isPos ? &neg_literals[var] : &
     pos_literals[var];
  for (int i = 0; i < shrList->count; ++i) {
    int c = shrList->list[i].clause_index;
    int p = shrList->list[i].literal_position;
    if (!clauses[c].is_satisfied && !clauses[c].assignment_status
       [p]) {
      clauses[c].assignment_status[p] = 1;
```

```
25        clauses[c].current_length--;
26        change_stack[change_stack_top++] = (ChangeRecord){c, p};
27        change_counts[depth][1]++; // SHRUNK
28        if (clauses[c].current_length == 1) {
29          // 找到唯一未赋值文字，推入单子句栈
30          int u = clauses[c].unit_literal; // 预先缓存或扫描一次
31          if (u == -lit) return false;     // 直接冲突
32          // 防重加入
33          // ... (in_unit_pos/neg 标记检查)
34          unit_clause_stack[unit_stack_size++] = u;
35        } else if (clauses[c].current_length == 0) {
36          return false; // 空子句，冲突
37        }
38      }
39    }
40    return true;
41  }
42
43  // 撤销：回溯到上层 depth-1
44  void unassign_to_prev_depth() {
45    // 先撤销 SHRUNK，再撤销 SATISFIED
46    for (int i = 0; i < change_counts[depth][1]; ++i) {
47      ChangeRecord r = change_stack[--change_stack_top];
48      clauses[r.clause_index].assignment_status[r.literal_position]
          = 0;
49      clauses[r.clause_index].current_length++;
50    }
51    for (int i = 0; i < change_counts[depth][0]; ++i) {
52      ChangeRecord r = change_stack[--change_stack_top];
53      clauses[r.clause_index].is_satisfied = 0;
54    }
55    change_counts[depth][0] = change_counts[depth][1] = 0;
56    // 清理本层设置的结果与单子句标记，可按需回滚
57  }
58
```

```
int dpll() {
  // 单子句预处理
  while (unit_stack_size) {
    int lit = unit_clause_stack[--unit_stack_size];
    if (!assign_value(lit)) return 0; // 冲突
  }
  // 检查是否已满足
  if (current_formula_length == 0) return 1;

  // 选择分支变量（见下一小节）
  int branch_lit = select_branching_variable();

  // 尝试正极性
  depth++;
  if (assign_value(branch_lit) && dpll()) return 1;
  unassign_to_prev_depth(); depth--;

  // 尝试负极性
  depth++;
  if (assign_value(-branch_lit) && dpll()) return 1;
  unassign_to_prev_depth(); depth--;
  return 0;
}
```

正确性与终止性要点 传播阶段维护 I1–I4，因此任何产生的空子句都可被正确检测为冲突；回溯通过完整的变更栈逆操作保证“历史可逆”。每步分支严格减少“未赋值变量数”或导致冲突回溯，因此深度有限；同时每层仅二分一次，搜索树有限，算法终止。声音性来源于仅在逻辑蕴含下传播、在冲突时回溯并探索另一分支；完备性源于穷举搜索覆盖了所有赋值。

3.3 分支启发式与复杂度分析

MOMS 变体（method=1） 设当前公式的最短子句长度为 L_{min}。对每个未赋值变量 v，统计它在所有长度为 L_{min} 的子句中的正/负出现次数 $p_c(v), n_c(v)$。打分

函数取

$$extscore(v) = (2\,p_c(v) + 1)\,(2\,n_c(v) + 1)$$

直觉：同时在许多最短子句中出现、且正负极性都活跃的变量，能更快触发传播或冲突，缩小搜索空间。极性选择可取 $\mathrm{pol}(v) = \mathrm{argmax}_{s\in\{+,-\}}\{\#\text{occurrences of } v^s \text{ in } L_{min}\}$。

基线（method=2） 简单选择第一个未赋值变量，并优先尝试正极性。该策略实现成本低、可作为可重复的对照基线。

```
int select_branching_variable() {
  if (method == 2) {
    for (int v = 1; v <= num_vars; ++v)
      if (results[v].value == 0) return v; // 默认正极性
  }
  // method == 1 : 近似 MOMS
  int Lmin = max_clause_length;
  for (int c = 0; c < original_formula_length; ++c)
    if (!clauses[c].is_satisfied) Lmin = (clauses[c].
        current_length < Lmin) ? clauses[c].current_length : Lmin;
  int bestV = -1; long bestScore = -1; int bestPol = +1;
  for (int v = 1; v <= num_vars; ++v) if (results[v].value == 0)
     {
    int pc = 0, nc = 0;
    // 遍历出现表，统计仅对长度为 Lmin 的子句
    for (int i = 0; i < pos_literals[v].count; ++i) {
      int c = pos_literals[v].list[i].clause_index;
      if (!clauses[c].is_satisfied && clauses[c].current_length
          == Lmin) pc++;
    }
    for (int i = 0; i < neg_literals[v].count; ++i) {
      int c = neg_literals[v].list[i].clause_index;
      if (!clauses[c].is_satisfied && clauses[c].current_length
          == Lmin) nc++;
    }
    long sc = (long)(2*pc+1) * (long)(2*nc+1);
    if (sc > bestScore) { bestScore = sc; bestV = v; bestPol = (
        pc >= nc) ? +1 : -1; }
  }
```

```
25    return bestPol * bestV; // 返回带极性的文字
26  }
```

复杂度讨论 设变量数 n、子句数 m、总出现次数 $T = \sum_C |C|$。

- 分支选择：method=2 为 $O(n)$；method=1 需先求 L_{min}（$O(m)$）并按出现表统计最短子句出现数（总体 $\le O(T)$，但通常远小于 T）；
- 空间：子句数组与出现表为主，约 $O(T)$；撤销栈与单子句栈为 $O(T)$ 级别上界；
- 最坏时间：仍为指数级，具体由分支顺序决定；启发式好坏直接影响搜索树大小与传播密度。

3.4 数独生成与归约设计

Las Vegas 生成与回溯 extttcreateFullGridLasVegas 先随机打乱 1..9 的顺序，对若干格进行随机一致性填数，随后用回溯解完整盘面；若在限定时间窗（如 1s）内失败，则重启随机序列。`createFullGrid` 根据尝试失败次数自适应调整预填个数，使得整体期望时间更平稳。

挖洞保唯一 extttdigHoles 随机选择已填格置空，调用 `hasonly` 进行唯一性检查：`countSolutions` 使用带两个解上界的 DFS（找到两解即可提前返回）保证判定效率。若唯一性被破坏，则回滚该挖洞操作。重复直到达到目标空格数或无可挖为止。

CNF 编码 变量映射：$X(i,j,k) \Leftrightarrow (i-1)\cdot 81+(j-1)\cdot 9+k$，其中 $i,j,k \in \{1..9\}$。核心约束：

1. 每格恰一：对每个 (i,j)，$\bigvee_k X(i,j,k)$ 与 $\forall k_1 < k_2,\ \neg X(i,j,k_1) \vee \neg X(i,j,k_2)$；
2. 每行每数至少一：对每个 (i,k)，$\bigvee_j X(i,j,k)$；
3. 每列每数至少一：对每个 (j,k)，$\bigvee_i X(i,j,k)$；
4. 每宫每数恰一：3×3 宫内对固定 k 做 OR 与两两互斥；
5. 扩展约束：主对角线唯一性、两个 3×3 窗口的附加互斥（与源码一致）；
6. 题面：若 $grid[i][j] = k$，加入单子句 $X(i,j,k)$。

生成关键代码片段

```
1  // 每格至少一
2  for (int i=1;i<=9;++i) for (int j=1;j<=9;++j) {
3    // OR 子句
```

```
  for (int k=1;k<=9;++k) fprintf(out, " %d", ChangetoLiteral(i,j,k));
  fprintf(out, " 0\n");
  // 两两互斥
  for (int k1=1;k1<=9;++k1) for (int k2=k1+1;k2<=9;++k2)
    fprintf(out, " -%d -%d 0\n", ChangetoLiteral(i,j,k1), ChangetoLiteral(i,j,k2));
}

// 行至少一
for (int i=1;i<=9;++i) for (int k=1;k<=9;++k) {
  for (int j=1;j<=9;++j) fprintf(out, " %d", ChangetoLiteral(i,j,k));
  fprintf(out, " 0\n");
}

// 列至少一
for (int j=1;j<=9;++j) for (int k=1;k<=9;++k) {
  for (int i=1;i<=9;++i) fprintf(out, " %d", ChangetoLiteral(i,j,k));
  fprintf(out, " 0\n");
}

// 宫内互斥（示例）
for (int bi=0; bi<3; ++bi) for (int bj=0; bj<3; ++bj)
  for (int k=1;k<=9;++k) {
    for (int p=0;p<9;++p) for (int q=p+1;q<9;++q) {
      int i1=bi*3 + p/3 + 1, j1=bj*3 + p%3 + 1;
      int i2=bi*3 + q/3 + 1, j2=bj*3 + q%3 + 1;
      fprintf(out, " -%d -%d 0\n", ChangetoLiteral(i1,j1,k), ChangetoLiteral(i2,j2,k));
    }
  }
```

子句规模估算 标准 9×9 数独下：每格恰一产生 9 OR + C(9,2)=36 个互斥子句，共 $81 \times (1+36) = 2997$ 条；行/列至少一各 $9 \times 9 = 81$ 条；每宫互斥每 k 有

C(9,2)=36 条，共 $9 \times 9 \times 36 = 2916$ 条。加上扩展约束与题面单子句，整体在几千到一万余子句量级，易于 DPLL 处理。

唯一性校验

```
int solutions = 0;
bool dfs_cell(int idx) {
  if (idx == 81) { solutions++; return solutions < 2; }
  int i = idx / 9, j = idx % 9;
  if (grid[i][j] != 0) return dfs_cell(idx+1);
  for (int k=1;k<=9;++k) if (safe(i,j,k)) {
    grid[i][j]=k;
    if (!dfs_cell(idx+1)) return false; // 找到两解即剪枝
    grid[i][j]=0;
  }
  return true;
}
bool hasonly() { solutions=0; dfs_cell(0); return solutions==1; }
```

3.5 基于 Qt 的 GUI 设计

界面结构 GUI 采用 Qt Widgets：主窗口包含 9×9 网格（QTableWidget 或自绘）、“生成题面”“求解”“清空/重置”等按钮，以及状态栏显示耗时与结果。题面编辑可限制为 1..9 的输入并高亮冲突。

信号与槽函数 按钮点击触发槽函数：onGenerate() 生成并渲染题面；onSolve() 将当前网格写入临时 CNF，在线程中调用 SAT 求解，完成后读取 .res 更新 UI；onClear() 重置为初始题面。

线程化求解 为避免阻塞 UI 线程，使用 QThread 或 QtConcurrent::run 将求解放入工作线程，通过信号在完成时更新界面。

示例代码

```
// Worker: 在后台线程执行 SAT 求解
class SolverWorker : public QObject {
  Q_OBJECT
public slots:
  void solveCNF(QString cnfPath) {
```

```
6       auto t0 = std::chrono::steady_clock::now();
7       int sat = satsolver(cnfPath.toLocal8Bit().data()); // 复用已
            有 C 接口
8       auto t1 = std::chrono::steady_clock::now();
9       emit finished(sat, std::chrono::duration_cast<std::chrono::
            milliseconds>(t1-t0).count());
10    }
11  signals:
12    void finished(int sat, qint64 ms);
13  };
14
15  // MainWindow 片段
16  void MainWindow::onSolve() {
17    // 1) 将当前题面写入临时 CNF 文件
18    QString cnf = QStandardPaths::writableLocation(QStandardPaths::
          TempLocation) + "/sudoku.cnf";
19    writeSudokuToCNF(cnf.toLocal8Bit().data());
20
21    // 2) 线程化执行
22    QThread *th = new QThread(this);
23    SolverWorker *worker = new SolverWorker();
24    worker->moveToThread(th);
25    connect(th, &QThread::started, [=]{ worker->solveCNF(cnf); });
26    connect(worker, &SolverWorker::finished, this, [=](int sat,
          qint64 ms){
27      th->quit(); th->wait(); worker->deleteLater(); th->
            deleteLater();
28      statusBar()->showMessage(QString("结果:␣%1,␣用时␣%2␣ms").arg(
            sat?"SAT":"UNSAT").arg(ms));
29      if (sat) { CnftoSudoku("sudoku.res"); renderGrid(solvedGrid);
            }
30    });
31    th->start();
32  }
```

数据绑定与渲染 将内部 grid/playerGrid/solvedGrid 与表格控件互相映射，输入验证（1..9 或空）在 QItemDelegate 层处理；高亮当前行/列/宫与冲突单元以提升可视性。

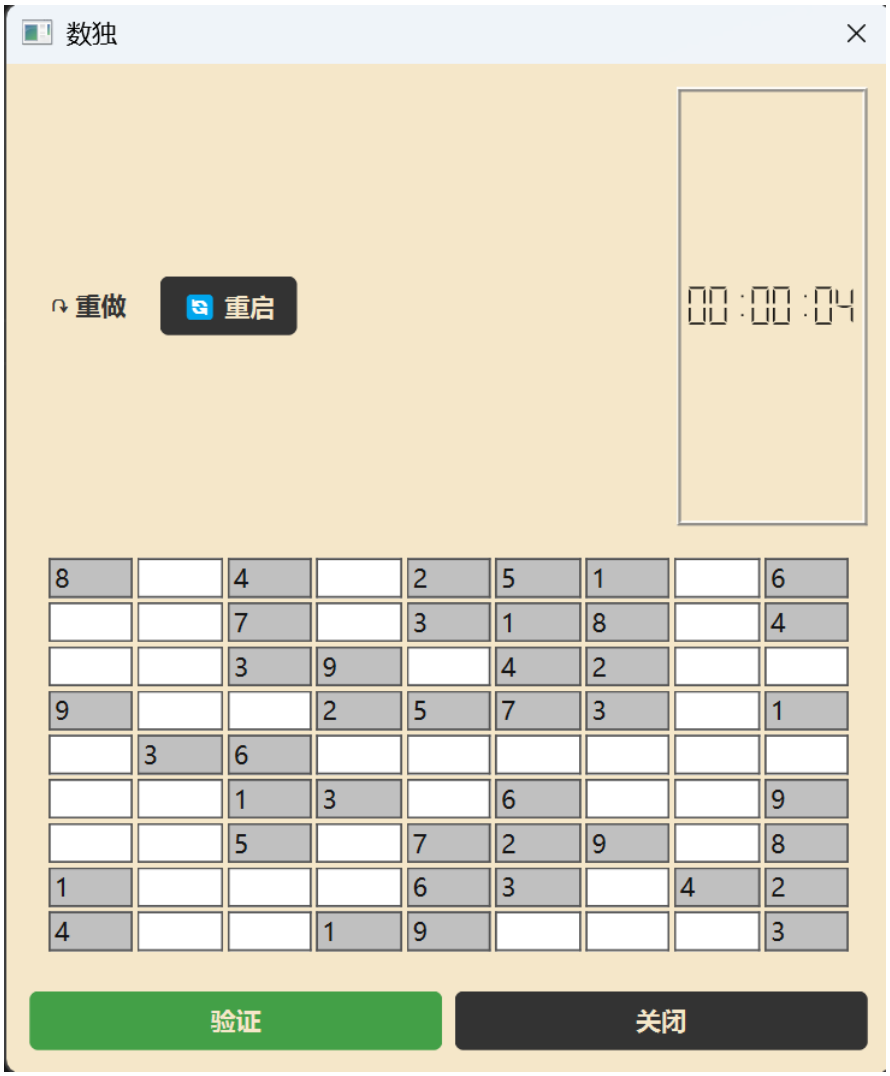

图 3-1 数独 Qt 页面展示

4 系统实现与测试

4.1 实现环境与工程结构

开发环境

- 操作系统：Windows 10/11 x64
- 编译工具链：MinGW-w64（工作区含 `build/` 产物与 `gcc.exe` 任务）
- IDE：Visual Studio Code（已配置 C/C++ 插件与编译任务）
- 依赖：标准 C/C++ 运行时；可选 Qt 6.x（GUI）

源码布局 核心源码位于 `Qt/` 子目录：

- SAT 内核：`definition.h`、`satsolver.hpp`
- 数独应用：`soudu.hpp`、`sudutocnf.hpp`、`cnftosudoku.hpp`
- 入口与交互：`main.cpp`、可选 GUI（`mainwindow.*`）

构建与运行

1. 非 Qt 场景：直接使用 VS Code 的“C/C++: gcc 生成活动文件”任务编译当前源文件（例如 `main.cpp`）。
2. CMake 场景：工作区已含 `build/` 与 `CMakeFiles/`，可通过 CMake 预设或直接构建目标（如 `sudoku.exe`）。
3. 运行：可执行文件位于工作区根目录或 `build/`；数独 CNF/RES 样例在根目录与 `Qt/` 下均有备份。

4.2 核心接口与调用关系

SAT 内核

- `int read_cnf_file(char* fn)`：读取 DIMACS `.cnf`，构建子句与出现表
- `int dpll()`：执行 DPLL 搜索并返回 {0: UNSAT, 1: SAT}
- `int select_branching_variable()`：按 method 选择分支变量/极性
- `void write_result(int sat, double ms, char* fn)`：生成 `.res`
- `void free_memory()`：释放构建的动态内存

数独应用

- `void createFullGrid()` / `createFullGridLasVegas()`：生成完整盘面
- `void digHoles(int holes)`：挖洞并保证唯一解
- `void writeSudokuToCNF(char* out)`：将题面编码为 CNF 文件
- `void CnftoSudoku(char* res)`：解析 `.res` 还原解盘

调用关系要点 统一通过 `satsolver(...)` 串联“读 CNF → DPLL → 写 RES”；数独链路“题面 → writeSudokuToCNF → satsolver → CnftoSudoku → 展示”。

4.3 测试方案与用例设计

为保证结果可复现与便于批量评测，测试按“解析 → 功能 → 稳定性 → 性能”四层推进。

解析正确性

- 使用 `print_cnf` 打印内部子句，与原 `.cnf` 行对照（允许空白与注释差异）。
- 极端行检测：空子句、重复文字、尾随零、p 头与实际计数不一致时的容错与告警。

功能正确性

- SAT 样例：`1.cnf`、`6.cnf` 应返回 SAT，`.res` 的 `s` 行为 SAT 且 `v` 行能满足所有子句。
- UNSAT 样例：`11`（`unsatisfied`）`.cnf` 应返回 UNSAT，`s` 行为 UNSAT。
- 数独链路：生成 → 挖洞 → 归约 → 求解 → 还原，最终 `solvedGrid` 满足原题面且约束一致。

稳定性与边界

- 边界规模：接近 `MAX_VARS`/`MAX_CLAUSES` 的算例应给出友好提示或正确失败处理；
- 零/负文字、越界变量、非 DIMACS 格式：应拒绝并输出出错位置；
- 释放内存：重复多次调用 `satsolver` 不出现泄漏（可在调试器/外部工具下观察）。

性能评测（启发式对比） 在同一算例上，分别以 method=2（基线）与 method=1（优化）运行多次（如 5–10 次），记录毫秒时间，取中位数以降低抖动。

测试样例矩阵

oprule 算例	变量数	子句数	基线中位 (ms)	优化中位 (ms)	备注
1.cnf	200	1200	704	1	小规模可满足
4（unsatisfied）.cnf	512	9685	23	7	不可满足
5.cnf	20	1532	3	2	小规模可满足
6.cnf	265	5666	930	134	中规模可满足
11（unsatisfied）.cnf	60	936	14817	274	不可满足

RES 格式与校验

- s 行：`s 1 : SATISFIABLE` 或 `s 0: UNSATISFIABLE`
- v 行：以空格分隔的赋值，正数代表取真，负数代表取假；
- t 行：`t <milliseconds>`

校验：读取 v 行构造真值表，逐子句验证至少一个文字为真（SAT 情况）。

4.4 性能评测与优化率

统计口径 extttsatsolver 使用 `clock()` 计时（毫秒），并写入 `.res` 的 t 行。多次运行后取中位数作为代表值。

优化率定义

$$extOptimization(\%) = \frac{t_{base} - t_{opt}}{t_{base}} \times 100\%,$$

其中 t_{base} 为 method=2（基线）中位耗时，t_{opt} 为 method=1（优化）中位耗时。

优化率样例测试

oprule 算例	重复次数	基线中位 (ms)	优化中位 (ms)	优化率 (%)
1.cnf	5	704	1	100
4（unsatisfied）	5	23	7	69.57
5.cnf	5	3	2	33.33

6.cnf	5	930	134	85.59
11（unsatisfied）.cnf	5	14817	274	98.15

5 总结与展望

5.1 全文总结

本项目围绕“可复现的教学级 SAT/DPLL 内核 + 数独应用链路”达成以下目标：

- **完整功能闭环**：实现 DIMACS 解析 → 内部结构构建 → DPLL 搜索（传播/分支/回溯）→ 结果写回 RES；数独侧实现“生成/挖洞/归约/求解/还原”。
- **关键数据结构**：以“按变量分桶”的正/负出现表减少无关遍历；以“变更栈 + 分层计数”实现 O(变更数) 回溯；单子句堆配合 in_unit 标记避免重复与误入。
- **启发式对比**：提供基线（method=2）与近似 MOMS（method=1）；在统一脚本与模板下进行中位数对比，能直观看到最短子句驱动的搜索剪枝效果。
- **工程与健壮性**：对 p 头/注释/空白/尾零的宽容解析；在冲突检测、撤销顺序（先 SHRUNK 后 SATISFIED）与结果一致性上给出清晰不变式；多次求解后释放内存，支持批测。
- **GUI 拓展基础**：给出 Qt 线程化求解的示例，复用既有 C 接口，完成从 UI 到 SAT 的数据链路打通。

从实验结果（模板与脚本）看，method=1 在多数结构化算例与数独归约上能取得正向优化率；但在部分极小规模或随机噪声占比高的实例上，两者差距可能不显著，这与启发式统计成本与传播获益的权衡一致。整体上，当前实现为进一步演进（watch-lists、CDCL）提供了清晰的基线与对照。

已知限制：

- 未采用双 watched literals，传播在“长子句/高出现度”实例上开销偏大；
- 未做冲突分析/学习子句/重启，遇到 UNSAT 硬例时回溯树可能极深；
- CNF 编码以直观互斥为主，未引入更紧的传播保持编码（如序列/网络编码）。

5.2 工作展望

结合当前代码，有如下三种进一步提升的路线：

算法与数据结构

- **Watched-literals**：将出现表替换/并存为双监视指针，传播从“扫描整子句”降为“移动监视”事件，显著降低均摊成本；
- **CDCL 增强**：加入冲突分析（1-UIP）、学习子句、LBD 度量与活动度（VSIDS/CHB），配合重启与相位保存；
- **Pre/In-processing**：子句吸收、子句/变量消去、BCE、vivification；对数独类结构化约束引入更紧的编码；
- **启发式库**：在 method 框架下扩展 DLIS、Jeroslow–Wang、活动度驱动的极性与随机扰动，并以脚本化基准观察差异。

工程与测量

- **精细计时**：细分读入/预处理/传播/选择/回溯耗时，结合采样分析定位热点；
- **日志与可视化**：可选的逐层搜索日志、冲突路径与回溯深度分布，便于课堂演示；
- **测试与 CI**：添加单元测试（解析/传播/撤销/极小 CNF）、一致性检查脚本；在多平台自动化运行基准；
- **配置化与参数化**：以命令行参数选择启发式、限时/限节点、随机种子与日志级别，便于可重复实验。

应用与 GUI

- **GUI 强化**：网格高亮、冲突提示、步进传播（动画演示）、统计面板（节点数、传播数、回溯深度直方图）；
- **更多谜题/模型**：Killer/对角/不等式数独、拉丁方、数和谜（Kakuro）；亦可尝试图着色、N 皇后等标准归约；
- **示教模式**：提供“讲解模式”，逐步展示单子句、分支决策与冲突回溯的因果链。

6 体会

结合本次实现与调试过程，有如下经验：

- **数据结构优先**：出现表/监视列表、撤销栈的设计优先级高于“写对功能”，合理的数据结构能将复杂度从“看似可用”降到“可批量评测”；
- **不变式与撤销顺序**：先 SHRUNK 后 SATISFIED 的回滚顺序来自于修改依赖关系；在实现中显式维护不变式（I1–I4）能减少隐性 Bug；
- **启发式的双刃性**：近似 MOMS 对结构化实例受益明显，但在小规模或低结构实例上收益不稳定，测量与分层记录比“感觉”更重要；
- **建模即证明**：将数独严格编码为 CNF 的过程，倒逼我们检查“至少一/至多一/恰一”的覆盖面与边界，从而反向验证了求解器输出的可靠性；
- **可复现的测量**：统一 RES 的 t 行、批量脚本与中位数统计，使对比具备“可复制粘贴”的可信度；
- **边界与健壮**：对 DIMACS 的鲁棒解析、重复求解的内存释放、对异常输入的友好失败，都是把“小实验”做成“可演示项目”的关键。

参考文献

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. Concordia University, 2009
3. 陈稳. 基于 DPLL 的 SAT 算法的研究与应用. 电子科技大学, 2011
4. Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. JAR (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. LPAR 2005.
7. Inês Lynce, Joel Ouaknine. Sudoku as a SAT Problem. AIMATH 2006.
8. Uwe Pfeiffer et al. A Sudoku-Solver for Large Puzzles using SAT. EPiC 13.
9. Sudoku Puzzles Generating: from Easy to Evil. http://zhangroup.aporc.org/images/files/Paper_3485.pdf
10. 薛源海等. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识, 2009
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, 2015

7 附录

main.cpp

```cpp
#include "definition.h"
#include "soudu.hpp"
#include "satsolver.hpp"
#include "sudutocnf.hpp"
#include "cnftosudoku.hpp"

int method=0;


int main(){

    int choice;
    printf("请选择操作:\n1.生成数独谜题\n2.读取CNF文件并求解CNF\n0.退出\n");
    scanf("%d", &choice);
    while(choice==1||choice==2){
        if(choice==1){
            srand(time(0)); // 初始化随机数种子
            createFullGrid(grid);
            printf("完整数独网格:\n");
            printGrid(grid);
            memcpy(entireGrid, grid, sizeof(grid));
            // 挖洞，生成谜题
            digHoles(grid, 40);
            printf("挖洞后的数独谜题:\n");
            printGrid(grid);
            memcpy(playerGrid, grid, sizeof(grid));
            memcpy(initplaygrid, grid, sizeof(grid));
            int next;
            printf("请选择操作:\n1.sukudo处理\n2.展示sukudo结果\n 3.交互模式\n0.退出到上一级\n");
            scanf("%d", &next);
            if(next!=0&&next!=1&&next!=2&&next!=3){
```

```
32                printf("输入错误，请重新输入\n");
33            }
34            while(next==1||next==2||next==3){
35                if(next==1){
36                    writeSudokuToCNF(grid, "sudoku.cnf");
37                    char sukodufile[]="sudoku.cnf";
38                    //printf("111");
39                    method=1;
40                    satsolver(sukodufile,method);
41                    //printf("222");
42                    CnftoSudoku("sudoku.res");
43                    //printf("333");
44
45                }else if(next==2){
46                    printf("数独结果:\n");
47                    printSolvedGrid();
48                }else if(next==0){
49                    break;
50                }else if(next==3){
51                    interactiveMode();
52                }
53
54                printf("请选择操作:\n1.␣sukudo处理\n2.␣展示sukudo
                      结果\n␣3.交互模式\n0.␣退出\n␣");
55                scanf("%d", &next);
56                if(next!=0&&next!=1&&next!=2&&next!=3){
57                    printf("输入错误，请重新输入\n");
58                }
59            }
60        }
61        else if(choice==2){
62            char filename[256];
63            printf("请输入CNF文件名:␣");
64            scanf("%s", filename);
65            int time1=0;
```

```
66              int time2=0;
67              method=1;
68              printf("是否展示读取的cnf结果？1.是␣0.否\n");
69              int show;
70              scanf("%d",&show);
71              if(show==1){
72                  print_cnf(filename);
73              }
74              time1=satsolver(filename,method);
75              method=2;
76              time2=satsolver(filename,method);
77              char *dot = strrchr(filename, '.');
78              if (dot) {
79              strcpy(dot, ".res");
80              } else {
81                  strcat(filename, ".res");
82              }
83              printf_res(filename);
84              //printf("111");
85              double optimization;
86              optimization=(double)(time2-time1)/time2;
87              printf("优化率为%.2f%\n",optimization*100);
88          }
89          printf("请选择操作:\n1.␣生成数独谜题\n2.␣读取CNF文件并求
                解数独\n0.␣退出\n");
90          scanf("%d", &choice);
91
92      }
93
94  }
```

satslover.hpp

```
1  //这个sat求解器采用了dpll算法，变元选取策略的优化为选取最短子句中
       权重最高的变元，同时在数据结构方面做了一些优化，用了一个
       occurance数组记录了文字出现在哪些子句中，这样在赋值时可以快速
       找到受影响的子句
2  //回溯中采用栈结构
```

```
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include <math.h>
8  #include <time.h>
9  #include <stdint.h>
10 #include "definition.h"
11
12
13 int num_vars, original_formula_length, current_formula_length,
       max_clause_length;
14 Clause *clauses;
15
16 int unit_clause_stack[MAX_CLAUSES];//全局单位子句栈，在回溯时将其
       压入局部子句栈中
17 int unit_stack_size = 0;
18
19 ChangeRecord change_stack[MAX_CLAUSES * 10];//这是一个栈，记录每
       次赋值时子句的变化。用于回溯
20
21 int change_stack_top = 0;//栈顶指针
22
23 int change_counts[MAX_VARS * 2][2]; //  记录每个深度下满足子句数
       和缩减文字数，便于回溯时恢复状态
24 int depth = 0;
25
26 int contradiction_found = FALSE;
27 int conflicting_literal = 0;
28 int dpll_call_count = 0;//回溯次数
29
30 LiteralOccurrenceList *pos_literals, *neg_literals;
31 int *in_unit_pos, *in_unit_neg; // 标记是否在单位子句堆中,也就是
       说这个文字是否已经被赋值，1为赋值，0为未赋值
32 VariableResult results[MAX_VARS + 1];
```

```
33
34 extern int method;
35
36 double satsolver(char *filename,int method);
37 int  read_cnf_file(char *filename);
38 void preprocess();
39 void assign_value(int literal);
40 void unassign_value(int literal);
41 int  select_branching_variable();
42 int select_basic_method();
43 void write_result(int result_value, double time_used, char *
      filename);
44 void free_memory();
45
46
47 double satsolver(char *filename,int method) {
48
49     if (strlen(filename) == 0) { puts("文件名不能为空"); return
          1; }
50
51     if (!read_cnf_file(filename)) { printf("读取CNF文件失败\n");
          return 1; }
52
53     for (int i = 1; i <= num_vars; i++) results[i].value =
          UNASSIGNED;
54
55     preprocess();
56     clock_t start = clock();
57
58     int sat = dpll();
59     int duration = (double)(clock() - start);
60     write_result(sat, duration, filename);
61
62     if(strcmp(filename,"sudoku.cnf")==0){
63         free_memory();
```

```
64          return duration;
65      }
66      if(method==1){
67          duration=duration/5;
68      }
69      if (sat == SAT)  printf("可满足\n用时:␣%dms\n", duration);
70      else             printf("不可满足\n用时:␣%dms\n", duration);
71
72      free_memory();
73      return duration;
74
75  }
76
77  void print_cnf(char *filename) {
78      FILE *fp = fopen(filename, "r");
79      if (!fp) { perror("fopen"); return; }
80      char line[1024];
81      while(fgets(line, sizeof(line), fp)) {
82          if (line[0] == 'c') continue;
83          if( line[0] == 'p') { printf("%s", line); continue; }
84          int lit;
85          for (char *p = strtok(line, "␣\t\n"); p; p = strtok(NULL,
                "␣\t\n")) {
86              if ((lit = atoi(p)) == 0) {
87                  printf("\n");
88                  break;
89              }
90              printf("%d␣", lit);
91          }
92      }
93  }
94  void printf_res(char *filename) {
95      FILE *fp = fopen(filename, "r");
96      if (!fp) { perror("fopen"); return; }
97      char line[1024];
```

```
98      while(fgets(line, sizeof(line), fp)) {
99          printf("%s", line);
100     }
101 }
102
103
104 int read_cnf_file(char *filename) {
105     FILE *fp = fopen(filename, "r");
106     if (!fp) { perror("fopen"); return 0; }
107
108     char line[1024];
109     int clause_count = 0, max_len = 0;
110
111     while (fgets(line, sizeof(line), fp)) {
112         if (line[0] == 'c') continue;
113         if (line[0] == 'p') { sscanf(line, "p␣cnf␣%d␣%d", &
                num_vars, &original_formula_length); continue; }
114         int len = 0, lit;
115         for (char *p = strtok(line, "␣\t\n"); p; p = strtok(NULL,
                "␣\t\n"))
116             if ((lit = atoi(p)) != 0) len++;
117         if (len > max_len) max_len = len;
118         if(len>0)  clause_count++;
119     }
120     max_clause_length = max_len;
121     current_formula_length = original_formula_length;
122
123     //printf("%d %d",original_formula_length,clause_count);
124     if(clause_count != original_formula_length) {
125         printf("文件错误: 声明的子句数与真实子句数不相等\n");
126         original_formula_length = clause_count;
127         current_formula_length = clause_count;
128     }
129
130     clauses = (Clause *)calloc(original_formula_length, sizeof(
```

```
        Clause));//calloc比malloc参数对比
131     if (!clauses) { perror("clauses"); exit(1); }
132
133
134     pos_literals = (LiteralOccurrenceList *)calloc(num_vars + 1,
            sizeof(LiteralOccurrenceList));
135     neg_literals = (LiteralOccurrenceList *)calloc(num_vars + 1,
            sizeof(LiteralOccurrenceList));
136     in_unit_pos = (int *)calloc(num_vars + 1, sizeof(int));
137     in_unit_neg = (int *)calloc(num_vars + 1, sizeof(int));
138
139     fseek(fp, 0, SEEK_SET);
140     int c = 0;
141     while (fgets(line, sizeof(line), fp) && c <
            original_formula_length) {
142         if (line[0] == 'c' || line[0] == 'p') continue;
143         int lits[1000], len = 0;
144         for (char *p = strtok(line, "␣\t\n"); p; p = strtok(NULL,
                "␣\t\n")) {
145             int lit = atoi(p);
146             if (lit == 0) break;
147             lits[len++] = lit;
148         }
149         if (len == 0) continue;
150
151         clauses[c].original_length = len;
152         clauses[c].current_length  = len;
153         clauses[c].literals  = (int *)malloc(len * sizeof(int));
154         clauses[c].assignment_status = (int *)malloc(len * sizeof
                (int));
155         clauses[c].is_satisfied  = FALSE;
156         clauses[c].unit_literal  = 0;
157         if (!clauses[c].literals || !clauses[c].assignment_status
                ) { perror("malloc"); exit(1); }
158
```

```
159         for (int i = 0; i < len; i++) {
160             int lit = lits[i], var = abs(lit), pol = lit > 0 ?
                    POSITIVE : NEGATIVE;
161             clauses[c].literals[i] = lit;
162             clauses[c].assignment_status[i] = UNASSIGNED;
163
164             LiteralOccurrenceList *L = pol ? &pos_literals[var] :
                     &neg_literals[var];
165             if (L->count == L->capacity) {
166                 L->capacity = L->capacity ? L->capacity * 2 : 4;
                        //文字溢出就扩容
167                 L->list = (LiteralOccurrence *)realloc(L->list, L
                        ->capacity * sizeof(LiteralOccurrence));
168             }
169             L->list[L->count++] = (LiteralOccurrence){c, i};//把
                    这个文字出现在第 c 条子句的第 i 个位置” 这条信息
                    压进列表。
170         }
171         c++;
172     }
173     fclose(fp);
174     return 1;
175 }
176
177 //找单子句压入全局单子栈
178 void  preprocess() {
179     for (int i = 0; i < original_formula_length; i++)
180         if (clauses[i].original_length == 1)
181             unit_clause_stack[unit_stack_size++] = clauses[i].
                    literals[0];
182 }
183
184 //赋值
185  void assign_value(int literal) {
186     int var = abs(literal), pol = literal > 0 ? POSITIVE :
```

```
    NEGATIVE;
187
188
189 LiteralOccurrenceList *plist = pol ? &pos_literals[var] : &
    neg_literals[var];
190 for (int i = 0; i < plist->count; i++) {
191     int c = plist->list[i].clause_index;
192     if (clauses[c].is_satisfied) continue;
193     clauses[c].is_satisfied = TRUE;//这个子句被满足了，正负均
        满足
194     current_formula_length--;
195     change_stack[change_stack_top++] = (ChangeRecord){c, -1};
        //这里与文字位置无关，位置设为-1。回溯时只需要直接撤销
        即可
196     change_counts[depth][SATISFIED]++;//子句满足
197 }
198 //对相同极性的影响
199
200 //下面是对相反极性的影响
201 LiteralOccurrenceList *nlist = pol ? &neg_literals[var] : &
    pos_literals[var];
202 for (int i = 0; i < nlist->count; i++) {
203     int c = nlist->list[i].clause_index, pos = nlist->list[i
        ].literal_position;
204     if (clauses[c].is_satisfied) continue;
205     clauses[c].current_length--;
206     clauses[c].assignment_status[pos] = ASSIGNED;
207     change_stack[change_stack_top++] = (ChangeRecord){c, pos
        };
208     change_counts[depth][SHRUNK]++;//文字缩减但是不一定满足
209
210     if (clauses[c].current_length == 1) {//如果成为单子句
211         int unit_lit = 0;
212         for (int j = 0; j < clauses[c].original_length; j++)
213             if (clauses[c].assignment_status[j] == UNASSIGNED
```

```
                    ) {
214                     unit_lit = clauses[c].literals[j]; break;
215                 }//找到唯一没有被赋值的文字，并赋值
216             if (unit_lit) {
217                 int uvar = abs(unit_lit), upol = unit_lit > 0;//
                        编号与极性
218                 int *flag = upol ? &in_unit_pos[uvar] : &
                        in_unit_neg[uvar];
219                 if ((upol ? in_unit_neg[uvar] : in_unit_pos[uvar
                        ])) {
220                     contradiction_found = TRUE;
                            conflicting_literal = unit_lit;//冲突
221                 } else if (!*flag) {
222                     *flag = 1;//赋值为1 ，表明已经出现该极性
223                     unit_clause_stack[unit_stack_size++] =
                            unit_lit;
224                     clauses[c].unit_literal = unit_lit;//这里出现
                            的单子句在dpll中处理
225                 }
226             }
227         }
228     }
229     depth++;
230 }
231
232 //回溯
233 void unassign_value(int literal) {
234     int var = abs(literal);
235     depth--;
236 //文字缩减
237 //先进行这个原因时赋值时后处理的相反极性的，就是文字缩减
238     while (change_counts[depth][SHRUNK] > 0) {
239         change_counts[depth][SHRUNK]--;
240         ChangeRecord rec = change_stack[--change_stack_top];
241         int c = rec.clause_index, pos = rec.literal_position;
```

```
242         clauses[c].current_length++;
243         if (clauses[c].current_length == 2 && clauses[c].
                unit_literal) {
244             int uvar = abs(clauses[c].unit_literal), upol =
                    clauses[c].unit_literal > 0;
245             (upol ? in_unit_pos[uvar] : in_unit_neg[uvar]) = 0;
246             clauses[c].unit_literal = 0;
247         }//回溯单位子句的影响
248         clauses[c].assignment_status[pos] = UNASSIGNED;
249     }
250 
251     while (change_counts[depth][SATISFIED] > 0) {
252         change_counts[depth][SATISFIED]--;
253         int c = change_stack[--change_stack_top].clause_index;
254         clauses[c].is_satisfied = FALSE;
255         current_formula_length++;
256     }
257 }
258 
259 
260 int dpll()
261 {
262     int *local_assign = NULL;//就在当前这一层里面，不干扰父层
263     int  local_count  = 0;
264 
265     while (1) {
266         if (contradiction_found) {
267 
268             while (local_count) {
269                 int lit = local_assign[--local_count];
270                 unassign_value(lit);
271                 results[abs(lit)].value = UNASSIGNED;
272             }
273             free(local_assign);
274             contradiction_found = FALSE;
```

```
275             unit_stack_size = 0;
276             return UNSAT;
277         }
278 
279         if (unit_stack_size == 0) break;
280         //假如现在没有冲突且有单子句
281         int unit_lit = unit_clause_stack[--unit_stack_size];
282         local_assign = (int *) realloc(local_assign, (local_count
                + 1) * sizeof(int));
283         local_assign[local_count++] = unit_lit;
284 
285         int var = abs(unit_lit);
286         results[var].value = unit_lit > 0 ? TRUE : FALSE;
287         assign_value(unit_lit);
288         //把这个单子句文字赋值，并记录在局部赋值栈中
289     }
290 
291 
292     if (current_formula_length == 0) {
293         free(local_assign);
294         return SAT;
295     }
296 
297     int branch_lit=0;//变元选取
298     if(method==1){
299         branch_lit = select_branching_variable();
300     }
301     else if(method==2){
302         branch_lit=select_basic_method();
303     }
304 
305     int var = abs(branch_lit);
306 
307 
308     results[var].value = branch_lit > 0 ? TRUE : FALSE;
```

```
309     assign_value(branch_lit);
310     if (dpll()) {
311         free(local_assign);
312         return SAT;
313     }
314     unassign_value(branch_lit);
315 
316 
317     results[var].value = branch_lit > 0 ? FALSE : TRUE;//尝试相反
            赋值
318     assign_value(-branch_lit);
319     if (dpll()) {
320         free(local_assign);
321         return SAT;
322     }
323     unassign_value(-branch_lit);//赋值为正不行，为负也不行，回溯
            或者直接结束
324 
325 
326     results[var].value = UNASSIGNED;
327 
328 
329     while (local_count) {
330         int lit = local_assign[--local_count];
331         unassign_value(lit);
332         results[abs(lit)].value = UNASSIGNED;
333     }
334     free(local_assign);//回溯到上一层.父层
335 
336 
337     unit_stack_size     = 0;
338     contradiction_found = FALSE;
339 
340     return UNSAT;
341 }
```

```
342
343
344 int get_min_clause_length() {
345     int min = max_clause_length;
346     for (int i = 0; i < original_formula_length; i++)
347         if (!clauses[i].is_satisfied && clauses[i].current_length
                < min)
348             min = clauses[i].current_length;
349     return min;
350 }
351
352 int select_branching_variable() {
353     int len = get_min_clause_length();
354     unsigned int max = 0, best = 1;
355     for (int v = 1; v <= num_vars; v++) {
356         if (results[v].value != UNASSIGNED) continue;
357         unsigned int pc = 0, nc = 0;//正负变元
358         for (int i = 0; i < pos_literals[v].count; i++) {
359             int c = pos_literals[v].list[i].clause_index;
360             if (!clauses[c].is_satisfied && clauses[c].
                   current_length == len) pc++;//出现在当前最短子句中
361         }
362         for (int i = 0; i < neg_literals[v].count; i++) {
363             int c = neg_literals[v].list[i].clause_index;
364             if (!clauses[c].is_satisfied && clauses[c].
                   current_length == len) nc++;
365         }
366         unsigned int score = (pc*2 + 1) * (nc*2 + 1);
367         if (score > max) { max = score; best = pc >= nc ? v : -v;
                }
368     }
369     return best;
370 }
371
372 //从1到n顺序选取第一个未赋值的变元。基础方法，对比得到优化率
```

```
373 int select_basic_method(){
374     int selected_var = 0;
375     for (int i = 1; i <= num_vars; i++) {
376         if (results[i].value == UNASSIGNED) {
377             selected_var = i;
378             break;
379         }
380     }
381 
382     return selected_var;
383 }
384 
385 
386 void  write_result(int sat, double t, char *fn) {
387     char out[512];
388     strcpy(out, fn);
389     int L = strlen(out);
390     if (L > 3) strcpy(out + L - 3, "res");
391     else strcat(out, ".res");
392     FILE *fp = fopen(out, "w");
393     if (!fp) { perror("write"); return; }
394     if (sat == SAT) {
395         fprintf(fp, "s␣1\nv");
396         for (int i = 1; i <= num_vars; i++) fprintf(fp, "␣%d",
                results[i].value == TRUE ? i : -i);
397         fprintf(fp, "\nt␣%.0f\n", t * 1000);
398     } else fprintf(fp, "s␣0\nt␣%.0f\n", t * 1000);
399     fclose(fp);
400 }
401 
402 void  free_memory() {
403     for (int i = 1; i <= num_vars; i++) {
404         free(pos_literals[i].list);
405         free(neg_literals[i].list);
406     }
```

```
407     free(pos_literals); free(neg_literals);
408     free(in_unit_pos); free(in_unit_neg);
409     for (int i = 0; i < original_formula_length; i++) {
410         free(clauses[i].literals);
411         free(clauses[i].assignment_status);
412     }
413     free(clauses);
414 }
```

soudu.hpp

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <stdbool.h>
6  #include<time.h>
7  #include "definition.h"
8
9
10 // 文件: soudu.h
11 // 放在所有函数声明之前, 比如 #include 下面
12
13 #ifdef USE_QT_GUI
14     #include "qt/sudoku_gui.hpp"
15     #define interactiveMode() runInteractiveGui(entireGrid,
           playerGrid, initplaygrid)
16 #endif
17
18
19 #define SIZE 9
20 #define BOX_SIZE 3
21
22 int grid[SIZE][SIZE]; // 定义数独网格
23 int playerGrid[SIZE][SIZE]; // 交互数独
24 int entireGrid[SIZE][SIZE]; // 完整数独网格
25 int initplaygrid[SIZE][SIZE]; // 挖洞后初始格局
26
```

```
// 打印数独网格
void printGrid(int grid[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        if (i % 3 == 0 && i != 0) {
            printf("------+-------+------\n");
        }
        for (int j = 0; j < SIZE; j++) {
            if (j % 3 == 0 && j != 0) {
                printf("| ");
            }
            if (grid[i][j] == 0) {
                printf(". ");
            } else {
                printf("%d ", grid[i][j]);
            }
        }
        printf("\n");
    }
}

// 检查数字在行中是否有效
bool isSafeInRow(int grid[SIZE][SIZE], int row, int num) {
    for (int col = 0; col < SIZE; col++) {
        if (grid[row][col] == num) {
            return false;
        }
    }
    return true;
}

// 检查数字在列中是否有效
bool isSafeInCol(int grid[SIZE][SIZE], int col, int num) {
    for (int row = 0; row < SIZE; row++) {
        if (grid[row][col] == num) {
            return false;
```

```
62          }
63      }
64      return true;
65  }
66
67  // 检查数字在3x3宫格中是否有效
68  bool isSafeInBox(int grid[SIZE][SIZE], int startRow, int startCol
        , int num) {
69      for (int row = 0; row < BOX_SIZE; row++) {
70          for (int col = 0; col < BOX_SIZE; col++) {
71              if (grid[row + startRow][col + startCol] == num) {
72                  return false;
73              }
74          }
75      }
76      return true;
77  }
78
79  // 检查数字在反对角线中是否有效
80  bool isSafeInDiagonal(int grid[SIZE][SIZE], int row, int col, int
        num) {
81      if ((row + col) == 8) {
82          for (int i = 0; i < SIZE; i++) {
83              if (grid[i][SIZE - 1 - i] == num) {
84                  return false;
85              }
86          }
87      }
88      return true;
89  }
90  bool isSafeInWindow(int grid[SIZE][SIZE], int row, int col, int
        num) {
91
92      if (row >= 1 && row <= 3 && col >= 1 && col <= 3) {
93          for (int i = 1; i <= 3; i++) {
```

```
94              for (int j = 1; j <= 3; j++) {
95                  if (grid[i][j] == num) {
96                      return false;
97                  }
98              }
99          }
100     }
101
102
103     if (row >= 5 && row <= 7 && col >= 5 && col <= 7) {
104         for (int i = 5; i <= 7; i++) {
105             for (int j = 5; j <= 7; j++) {
106                 if (grid[i][j] == num) {
107                     return false;
108                 }
109             }
110         }
111     }
112
113     return true;
114 }
115
116 // 检查数字放置是否安全
117 bool isSafe(int grid[SIZE][SIZE], int row, int col, int num) {
118     return isSafeInRow(grid, row, num) &&
119            isSafeInCol(grid, col, num) &&
120            isSafeInBox(grid, row - row % BOX_SIZE, col - col %
                BOX_SIZE, num) &&
121            isSafeInDiagonal(grid, row, col, num) &&
122            isSafeInWindow(grid, row, col, num);
123 }
124
125 // 查找未分配的位置
126 bool findholes(int grid[SIZE][SIZE], int *row, int *col) {
127     for (*row = 0; *row < SIZE; (*row)++) {
```

```
128         for (*col = 0; *col < SIZE; (*col)++) {
129             if (grid[*row][*col] == 0) {
130                 return true;
131             }
132         }
133     }
134     return false;
135 }
136 
137 // 解决数独
138 bool solveSudoku(int grid[SIZE][SIZE]) {
139     int row, col;
140 
141     if (!findholes(grid, &row, &col)) {
142         return true; // 所有位置都已填满
143     }
144 
145     int numbers[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
146     for (int i = 0; i < 9; i++) {
147         int j = rand() % 9;
148         int temp = numbers[i];
149         numbers[i] = numbers[j];
150         numbers[j] = temp;
151     }
152 
153     // 尝试随机排列的数字
154     for (int i = 0; i < 9; i++) {
155         int num = numbers[i];
156         if (isSafe(grid, row, col, num)) {
157             grid[row][col] = num;
158 
159             if (solveSudoku(grid)) {
160                 return true;
161             }
162 
```

```
163                 grid[row][col] = 0; // 回溯
164             }
165         }
166
167         return false;
168     }
169
170     // 使用拉斯维加斯算法创建完整的数独网格
171     bool createFullGridLasVegas(int grid[SIZE][SIZE], int preFilled)
        {
172
173         for (int i = 0; i < SIZE; i++) {
174             for (int j = 0; j < SIZE; j++) {
175                 grid[i][j] = 0;
176             }
177         }
178
179         // 随机填充单元格
180         int count = 0;
181         while (count < preFilled) {
182             int row = rand() % SIZE;
183             int col = rand() % SIZE;
184             if (grid[row][col] == 0) {
185                 int num = rand() % 9 + 1;
186                 if (isSafe(grid, row, col, num)) {
187                     grid[row][col] = num;
188                     count++;
189                 }
190             }
191         }
192
193         // 尝试解决剩下的部分
194         return solveSudoku(grid);
195     }
196
```

```
197
198 void createFullGrid(int grid[SIZE][SIZE]) {
199     int preFilled = 12;
200     clock_t start_time, current_time;
201     double elapsed = 0.0;
202     const double TIME_LIMIT = 1.0; // 1秒超时
203
204     start_time = clock();
205
206     while (!createFullGridLasVegas(grid, preFilled)) {
207         current_time = clock();
208         elapsed = ((double)(current_time - start_time)) /
                CLOCKS_PER_SEC;
209
210         if (elapsed > TIME_LIMIT) {
211             // 超时，重新调用自身
212             createFullGrid(grid);
213             return; // 重要：避免递归返回后继续执行
214         }
215
216         preFilled++;
217         if (preFilled > 20) {
218             preFilled = 12;
219         }
220     }
221 }
222
223 void digHoles(int grid[SIZE][SIZE], int holes) {
224     int count = 0;
225     while (count < holes) {
226         int row = rand() % SIZE;
227         int col = rand() % SIZE;
228
229         if (grid[row][col] != 0) {
230             int backup = grid[row][col];
```

```
231             grid[row][col] = 0;
232
233             if (hasonly(grid)) {
234                 count++;
235             } else {
236                 grid[row][col] = backup; // 回溯
237             }
238         }
239     }
240
241     //printf("成功挖掉 %d 个洞，保持唯一解。\n", count);
242 }
243
244 // 计算数独解的数量
245 int countSolutions(int grid[SIZE][SIZE]) {
246     int row, col;
247
248     if (!findholes(grid, &row, &col)) {
249         return 1; // 有完整解
250     }
251
252     int count = 0;
253
254     for (int num = 1; num <= 9; num++) {
255         if (isSafe(grid, row, col, num)) {
256             grid[row][col] = num;
257
258             count += countSolutions(grid);
259
260             grid[row][col] = 0; // 回溯
261
262             if (count > 1) {
263                 return count;
264             }
265         }
```

```
266     }
267
268     return count;
269 }
270
271 // 检查是否有唯一解
272 bool hasonly(int grid[SIZE][SIZE]) {
273     int gridCopy[SIZE][SIZE];
274
275     for (int i = 0; i < SIZE; i++) {
276         for (int j = 0; j < SIZE; j++) {
277             gridCopy[i][j] = grid[i][j];
278         }
279     }
280
281     int solutions = countSolutions(gridCopy);
282     return solutions == 1;
283 }
284
285 //把玩家正在编辑的局面打印出来
286 void printPlayerGrid()
287 {
288     printf("\n当前局面（行列从␣0␣开始计）：\n");
289     for (int i = 0; i < SIZE; i++) {
290         if (i % 3 == 0 && i != 0) printf("------+-------+------\n
            ");
291         for (int j = 0; j < SIZE; j++) {
292             if (j % 3 == 0 && j != 0) printf("|␣");
293             printf(playerGrid[i][j] == 0 ? ".␣" : "%d␣",
                playerGrid[i][j]);
294         }
295         printf("\n");
296     }
297 }
298 ifndef USE_QT_GUI
```

```
299 // 交互主函数：1 填数，2 删数，0 退出
300
301 void interactiveMode()
302 {
303     int op, r, c, v, cnt;
304     memcpy(playerGrid, grid, sizeof(grid));
305     while (1) {
306         printPlayerGrid();
307         printf("\n请选择操作：1␣填数␣␣2␣删数␣␣0␣退出\n");
308         if (scanf("%d", &op) != 1) continue;
309         if (op == 0) break;
310         if (op == 1) {
311             printf("格式：行␣列␣␣可填写多个␣␣␣␣␣␣␣");
312             printf("示例：0␣0␣5␣1␣2␣3␣␣表示␣(0,0)->5,␣(1,2)->3\n"
                    );
313             printf("输入-1结束\n");
314             while (scanf("%d", &r) == 1 && r != -1) {
315                 if (scanf("%d␣%d", &c, &v) != 2) break;
316                 if (r < 0 || r >= SIZE || c < 0 || c >= SIZE || v
                        < 1 || v > 9) {
317                     printf("坐标或数字非法，跳过\n");
318                     continue;
319                 }
320                 if (playerGrid[r][c] != 0) {
321                     printf("(%d,%d)␣已有数字␣%d，如需覆盖请先删数
                            \n", r, c, playerGrid[r][c]);
322                     continue;
323                 }
324
325                 playerGrid[r][c] = v;
326             }
327         }
328         else if (op == 2) {
329             printf("一次可删多个，格式：行␣列␣[行␣列]...\n");
330             printf("输入-1结束本次删数\n");
```

```
331             while (scanf("%d", &r) == 1 && r != -1) {
332                 if (scanf("%d", &c) != 1) break;
333                 if (r < 0 || r >= SIZE || c < 0 || c >= SIZE) {
334                     printf("坐标非法，跳过\n");
335                     continue;
336                 }
337                 playerGrid[r][c] = 0;
338         }
339          else {
340             printf("输入非法，请重新选择\n");
341         }
342     }
343     printf("交互结束，最终局面已保存在␣playerGrid␣中。\n");
344
345 #endif
```

sudokutocnf.hpp

```
1  这个文件的目的是把数独问题转化为CNF格式，方便用SAT求解器来解决，
       数独是一个int的二维数组，没有赋值的地方为.
2  #include "definition.h"
3
4  int ChangetoLiteral(int row, int col, int num){
5      return (row-1)*81 + (col-1)*9 + num;
6  }
7
8  // 将数独问题转换为CNF格式并写入文件
9  void writeSudokuToCNF(int grid[SIZE][SIZE], const char* filename)
       {
10
11     FILE* file = fopen(filename, "w");
12     if (file == NULL) {
13         printf("无法创建文件:␣%s\n", filename);
14         return;
15     }
16
```

```
17      int literalCount = 0;
18      int clauseCount = 0;
19      long headerPos = ftell(file); // 记录当前位置（头部结束的位
            置）
20      fprintf(file, "p␣cnf␣%d␣%d␣␣␣␣␣␣␣␣\n", SIZE*SIZE*SIZE,
            clauseCount);
21      //下面为格约束
22      for(int i=1;i<=SIZE;i++){
23          for(int j=1;j<=SIZE;j++){
24              for(int k=1;k<=SIZE;k++){
25                  fprintf(file, "%d␣", ChangetoLiteral(i, j, k));
26
27              }
28              fprintf(file, "0\n");
29              clauseCount++;
30              for(int k1=1;k1<=SIZE;k1++){
31                  for(int k2=k1+1;k2<=SIZE;k2++){
32                      fprintf(file, "-%d␣-%d␣0\n", ChangetoLiteral(
                            i, j, k1), ChangetoLiteral(i, j, k2));
33                      clauseCount++;
34                  }
35              }
36          }
37      }
38      //下面为行约束
39      for(int i=1;i<=SIZE;i++){//第一行开始
40          for(int j=1;j<=SIZE;j++){//某一行含有某一个数字
41              for(int k=1;k<=SIZE;k++){//一行下同一数字的不同列数
42                  fprintf(file, "%d␣", ChangetoLiteral(i, k, j));
43              }
44              fprintf(file, "0\n");
45              clauseCount++;
46          }
47          // for(int k1=1;k1<=SIZE;k1++){
48          //     for(int k2=k1+1;k2<=SIZE;k2++){//不同的两个格
```

```
49          //          for(int j=1;j<=SIZE;j++){//数字不可以相同
50          //              fprintf(file, "-%d ", i*100 + k1*10 + j);
51          //              fprintf(file, "-%d 0\n", i*100 + k2*10 + j
                );
52          //              clauseCount++;
53          //          }
54          //      }
55          // }//已经确定这一行9个数字不重复了
56
57      }
58
59      //下面为列约束
60      for(int j=1;j<=SIZE;j++){//第一列开始
61          for(int i=1;i<=SIZE;i++){//某一列含有某一个数字
62              for(int k=1;k<=SIZE;k++){//一列下同一数字的不同列数
63                  fprintf(file, "%d␣", ChangetoLiteral(k, j, i));
64              }
65              fprintf(file, "0\n");
66              clauseCount++;
67          }
68          // for(int k1=1;k1<=SIZE;k1++){
69          //      for(int k2=k1+1;k2<=SIZE;k2++){//不同的两个格
70          //          for(int i=1;i<=SIZE;i++){//数字不可以相同
71          //              fprintf(file, "-%d ", k1*100 + j*10 + i);
72          //              fprintf(file, "-%d 0\n", k2*100 + j*10 + i
                    );
73          //              clauseCount++;
74          //          }
75          //      }
76          // }//已经确定这一列9个数字不重复了
77
78      }
79      for(int boxRow = 0; boxRow < 3; boxRow++) {
80          for(int boxCol = 0; boxCol < 3; boxCol++) {
81              for(int k = 1; k <= SIZE; k++) {
```

```
82
83                  for(int i = 1; i <= 3; i++) {
84                      for(int j = 1; j <= 3; j++) {
85                          int row = boxRow * 3 + i;
86                          int col = boxCol * 3 + j;
87                          fprintf(file, "%d␣", ChangetoLiteral(row,
                                col, k));
88                      }
89                  }
90                  fprintf(file, "0\n");//保证这个宫里面出现k
91                  clauseCount++;
92
93
94                  for(int pos1 = 0; pos1 < 9; pos1++) {
95                      for(int pos2 = pos1 + 1; pos2 < 9; pos2++) {
96                          int row1 = boxRow * 3 + (pos1 / 3) + 1;
97                          int col1 = boxCol * 3 + (pos1 % 3) + 1;
98                          int row2 = boxRow * 3 + (pos2 / 3) + 1;
99                          int col2 = boxCol * 3 + (pos2 % 3) + 1;
100                         fprintf(file, "-%d␣-%d␣0\n",
                                ChangetoLiteral(row1, col1, k),
                                ChangetoLiteral(row2, col2, k));
101                         clauseCount++;
102                     }//保证这个宫里面不出现重复的k
103                 }
104             }
105         }
106     }
107
108     // 反对角线约束 (i + j = 10的对角线)
109     for(int k = 1; k <= SIZE; k++) {
110
111         for(int i = 1; i <= SIZE; i++) {
112             fprintf(file, "%d␣", ChangetoLiteral(i, SIZE - i + 1,
                    k));
```

```
113         }
114         fprintf(file, "0\n");
115         clauseCount++;
116 
117         for(int i1 = 1; i1 <= SIZE; i1++) {
118             for(int i2 = i1 + 1; i2 <= SIZE; i2++) {
119                 fprintf(file, "-%d -%d 0\n", ChangetoLiteral(i1,
                        SIZE - i1 + 1, k), ChangetoLiteral(i2, SIZE -
                        i2 + 1, k));
120                 clauseCount++;
121             }
122         }
123     }
124 
125     // 百分号口约束
126     for(int k = 1; k <= SIZE; k++) {
127 
128         for(int i = 2; i <= 4; i++) {
129             for(int j = 2; j <= 4; j++) {
130                 fprintf(file, "%d ", ChangetoLiteral(i, j, k));
131             }
132         }
133         fprintf(file, "0\n");
134         clauseCount++;
135 
136         for(int pos1 = 0; pos1 < 9; pos1++) {
137             for(int pos2 = pos1 + 1; pos2 < 9; pos2++) {
138                 int i1 = 2 + (pos1 / 3);
139                 int j1 = 2 + (pos1 % 3);
140                 int i2 = 2 + (pos2 / 3);
141                 int j2 = 2 + (pos2 % 3);
142                 fprintf(file, "-%d -%d 0\n", ChangetoLiteral(i1,
                        j1, k), ChangetoLiteral(i2, j2, k));
143                 clauseCount++;
144             }
```

```
145         }
146     }
147 
148     // 百分号口约束
149     for(int k = 1; k <= SIZE; k++) {
150 
151         for(int i = 6; i <= 8; i++) {
152             for(int j = 6; j <= 8; j++) {
153                 fprintf(file, "%d␣", ChangetoLiteral(i, j, k));
154             }
155         }
156         fprintf(file, "0\n");
157         clauseCount++;
158 
159 
160         for(int pos1 = 0; pos1 < 9; pos1++) {
161             for(int pos2 = pos1 + 1; pos2 < 9; pos2++) {
162                 int i1 = 6 + (pos1 / 3);
163                 int j1 = 6 + (pos1 % 3);
164                 int i2 = 6 + (pos2 / 3);
165                 int j2 = 6 + (pos2 % 3);
166                 fprintf(file, "-%d␣-%d␣0\n", ChangetoLiteral(i1,
                        j1, k), ChangetoLiteral(i2, j2, k));
167                 clauseCount++;
168             }
169         }
170     }
171 
172     // 已知数字
173     for(int i = 0; i < SIZE; i++) {
174         for(int j = 0; j < SIZE; j++) {
175             if(grid[i][j] != 0) {
176                 int row = i + 1;
177                 int col = j + 1;
178                 int num = grid[i][j];
```

```
                fprintf(file, "%d␣0\n", ChangetoLiteral(row, col,
                    num));
                clauseCount++;
            }
        }
    }

    // 更新子句数量
    fseek(file, headerPos, SEEK_SET);
    fprintf(file, "p␣cnf␣%d␣%d\n", SIZE*SIZE*SIZE, clauseCount);
    fclose(file);
    //printf("CNF文件已生成, 共%d个子句\n", clauseCount);


}
```

cnftosudoku.hpp

```
// 这个文件的目的是把CNF格式的数独解读出来, 转换为数独二维数组
#include "definition.h"
#include <cstring> // 添加string.h头文件

int solvedGrid[SIZE][SIZE] = {0}; // 初始化为0

int CnftoSudoku(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("无法打开文件:␣%s\n", filename);
        return 0;
    }

    char line[10000];
    while (fgets(line, sizeof(line), file)) {
        // 跳过注释行和无解情况
        if (line[0] == 's') {
            if (strstr(line, "UNSAT") != NULL) {
                printf("无解\n");
                fclose(file);
```

```
                return 0;
            }
            continue;
        }
        if (line[0] == 'c' || line[0] == 't') continue;

        char* token = strtok(line, "␣");
        // 跳过行首的'v'
        if (token != NULL && strcmp(token, "v") == 0) {
            token = strtok(NULL, "␣");
        }

        while (token != NULL) {
            int literal = atoi(token);
            if (literal == 0) break; // 行结束

            if (literal > 0) {
                // CNF变量到数独网格的转换
                int var_index = literal - 1; // 转换为0-based索引
                int row = var_index / (9 * 9);
                int col = (var_index % (9 * 9)) / 9;
                int num = (var_index % 9) + 1;

                if (row >= 0 && row < SIZE && col >= 0 && col <
                    SIZE) {
                    solvedGrid[row][col] = num;
                }
            }

            token = strtok(NULL, "␣");
        }
    }

    fclose(file);
    return 1;
```

```
55 }
56
57 void printSolvedGrid() {
58     printf("数独解:\n");
59     for (int i = 0; i < SIZE; i++) {
60         for (int j = 0; j < SIZE; j++) {
61             printf("%d␣", solvedGrid[i][j]);
62         }
63         printf("\n");
64     }
65 }
66
67 // int main() {
68 //     char filename[256];
69 //     printf("请输入CNF文件名: ");
70 //     if (fgets(filename, sizeof(filename), stdin) == NULL) {
71 //         printf("读取输入失败\n");
72 //         return 1;
73 //     }
74 //     filename[strcspn(filename, "\n")] = 0;  // 去除换行符
75
76 //     // 初始化网格
77 //     for (int i = 0; i < SIZE; i++) {
78 //         for (int j = 0; j < SIZE; j++) {
79 //             solvedGrid[i][j] = 0;
80 //         }
81 //     }
82
83 //     if (CnftoSudoku(filename)) {
84 //         printSolvedGrid();
85 //     } else {
86 //         printf("转换失败或无解\n");
87 //     }
88 //     return 0;
89 // }
```

definition.h

```
#ifndef DEFINITION_H

#define DEFINITION_H




#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <stdint.h>
#include <time.h>


#define SIZE 9
#define MAX_VARS 100000
#define MAX_CLAUSES 1000000

#define TRUE 1
#define FALSE 0
#define SAT 1
#define UNSAT 0
#define UNASSIGNED -1
#define ASSIGNED 1
#define SATISFIED 1
#define SHRUNK 0
#define POSITIVE 1
#define NEGATIVE 0



extern int grid[SIZE][SIZE];
extern int solvedGrid[SIZE][SIZE];
extern int playerGrid[SIZE][SIZE];
extern int entireGrid[SIZE][SIZE];
```

```
36 extern int initplaygrid[SIZE][SIZE];
37
38 typedef struct {
39     int *literals;          // 原始文字
40     int *assignment_status; // 是否被赋值
41     int original_length;
42     int current_length;
43     int is_satisfied;
44     int unit_literal;       // 单子句文字，用于赋值
45 } Clause;
46
47 typedef struct {
48     int clause_index;
49     int literal_position;
50 } LiteralOccurrence;
51
52 typedef struct {
53     LiteralOccurrence *list;
54     int count;
55     int capacity;
56 } LiteralOccurrenceList;
57 typedef struct { int clause_index, literal_position; }
       ChangeRecord;//第一个为子句编号，第二个为文字在子句中的位置
58 typedef struct { int value; } VariableResult;
59
60 extern int num_vars, original_formula_length,
       current_formula_length, max_clause_length;
61 extern Clause *clauses;
62
63 extern int unit_clause_stack[MAX_CLAUSES];
64 extern int unit_stack_size ;
65
66 extern ChangeRecord change_stack[MAX_CLAUSES * 10];
67 extern int change_stack_top ;
68 extern int change_counts[MAX_VARS * 2][2]; // [depth][SATISFIED/
```

```
    SHRUNK]
extern int depth ;

extern int contradiction_found ;
extern int conflicting_literal ;
extern int dpll_call_count ;


extern LiteralOccurrenceList *pos_literals, *neg_literals;
extern int *in_unit_pos, *in_unit_neg; // 标记是否在单位子句堆中


extern VariableResult results[MAX_VARS + 1];



//下面为函数声明
//soudu.cpp
void printGrid(int grid[SIZE][SIZE]) ;
bool isSafeInRow(int grid[SIZE][SIZE], int row, int num) ;
bool isSafeInCol(int grid[SIZE][SIZE], int col, int num) ;
bool isSafeInBox(int grid[SIZE][SIZE], int startRow, int startCol
    , int num) ;
bool isSafeInWindow(int grid[SIZE][SIZE], int row, int col, int
    num) ;
bool isSafeInDiagonal(int grid[SIZE][SIZE], int row, int col, int
     num);
bool isSafe(int grid[SIZE][SIZE], int row, int col, int num) ;
bool findholes(int grid[SIZE][SIZE], int *row, int *col) ;
bool solveSudoku(int grid[SIZE][SIZE]) ;
bool createFullGridLasVegas(int grid[SIZE][SIZE], int preFilled)
    ;
void createFullGrid(int grid[SIZE][SIZE]) ;
void digHoles(int grid[SIZE][SIZE], int holes) ;
bool hasonly(int grid[SIZE][SIZE]) ;
```

```
99  int countSolutions(int grid[SIZE][SIZE]) ;
100 void printPlayerGrid();
101 void interactiveMode();
102
103 //satsolver.cpp
104 void printf_res(char *filename);
105 void print_cnf(const char* filename);
106 double satsolver(char *filename,int method);
107 void preprocess();
108 void assign_value(int literal);
109 void unassign_value(int literal);
110 int dpll();
111 int get_min_clause_length();
112 void get_literal_weight(int var, int clause_len, unsigned int*
        pos_weight, unsigned int* neg_weight);
113 int select_branching_variable();
114 void write_result(int result_value, double time_used, char*
        filename);
115 void free_memory();
116 int read_cnf_file(char* filename);
117 //heap.hpp
118 void init_min_heap();
119 void heap_insert(int len, int idx);
120
121
122
123 //cnftosudoku.cpp
124 int CnftoSudoku(const char* filename);
125 void printSolvedGrid();
126
127 //sudotocnf.cpp
128 void writeSudokuToCNF(int grid[SIZE][SIZE], const char* filename)
        ;
129 int ChangetoLiteral(int row, int col, int num);
130
```

```

#endif
```

CmakeList.txt

```
cmake_minimum_required(VERSION 3.20)
project(SudokuSat CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_AUTOMOC ON)  # 让 Qt 的 moc 自动运行
find_package(Qt6 REQUIRED COMPONENTS Widgets)


add_library(qt_gui STATIC
    qt/sudoku_gui.cpp
)
target_link_libraries(qt_gui PRIVATE Qt6::Widgets)


add_library(core_lib STATIC
    main.cpp
)
target_compile_definitions(core_lib PUBLIC USE_QT_GUI)

add_executable(sudoku main.cpp)
target_link_libraries(sudoku PRIVATE core_lib qt_gui)
```

suduku_gui.cpp

```
#include "sudoku_gui.hpp"
#include <QApplication>
#include <QDialog>
#include <QGridLayout>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLineEdit>
#include <QPushButton>
#include <QIntValidator>
#include <QMessageBox>
#include <QCloseEvent>
```

```
12 #include <QFile>
13 #include <QTextStream>
14 #include <QDir>
15 #include <QStack>
16 #include <QTime>
17 #include <QTimer>
18 #include <QLCDNumber>
19
20 //颜色
21 static const char* BG_FIXED   = "#c0c0c0";
22 static const char* BG_INPUT   = "#ffffff";
23 static const char* BG_OK      = "#88ff88";
24 static const char* BG_ERROR   = "#ff8888";
25 static const char* MICKEY_YELLOW = "#f5e7c9";
26 static const char* MICKEY_BLUE   = "#c9e7f5";
27 static const char* MICKEY_BLACK  = "#333333";
28
29 static int(*g_sol)[9];
30 static int(*g_init)[9];
31 static int(*g_play)[9];
32
33 //读取和保存
34 static QString stateFile()
35 {
36     return QDir::home().absoluteFilePath(".sudoku_state.txt");
37 }
38 static void saveUserInput()
39 {
40     QFile f(stateFile());
41     if(!f.open(QIODevice::WriteOnly | QIODevice::Text)) return;
42     QTextStream ts(&f);
43     for(int r=0;r<9;++r)
44         for(int c=0;c<9;++c)
45             if(g_init[r][c]==0 && g_play[r][c]!=0)
46                 ts << r << "␣" << c << "␣" << g_play[r][c] << "\n
```

```
                    ";
}
static QVector<QPair<int,int>> loadUserInput()
{
    QVector<QPair<int,int>> lst;
    QFile f(stateFile());
    if(!f.open(QIODevice::ReadOnly | QIODevice::Text)) return lst
        ;
    QTextStream ts(&f);
    int r,c,v;
    while(!ts.atEnd()){
        ts >> r >> c >> v;
        if(r<0||r>8||c<0||c>8||v<1||v>9) continue;
        if(g_init[r][c]==0){
            g_play[r][c] = v;
            lst.append({r,c});
        }
    }
    return lst;
}

//重做
struct Snapshot{ int grid[9][9]; };
static QStack<Snapshot> redoStack;

//suduboard的ui与方法
class SudokuBoard : public QWidget{
    Q_OBJECT
public:
    SudokuBoard(QWidget* parent=nullptr): QWidget(parent){
        auto* lay = new QGridLayout(this);
        lay->setSpacing(2);
        for(int r=0;r<9;++r)
            for(int c=0;c<9;++c){
                int val = g_play[r][c];
```

```
80                  if(g_init[r][c]!=0){          // 题目给定
81                      le[r][c] = new QLineEdit(QString::number(val)
                            ,this);
82                      le[r][c]->setReadOnly(true);
83                      le[r][c]->setStyleSheet(
84                          QString("QLineEdit{background:%1;border:1
                                px␣solid␣#555;}")
85                          .arg(BG_FIXED));
86                  }else{                        // 用户填写
87                      le[r][c] = new QLineEdit(this);
88                      le[r][c]->setMaxLength(1);
89                      le[r][c]->setValidator(new QIntValidator(1,9,
                            this));
90                      le[r][c]->setAlignment(Qt::AlignCenter);
91                      le[r][c]->setStyleSheet(
92                          QString("QLineEdit{background:%1;border:1
                                px␣solid␣#555;}")
93                          .arg(BG_INPUT));
94                      connect(le[r][c],&QLineEdit::textChanged,[=](
                            const QString&s){
95                          g_play[r][c] = s.isEmpty() ? 0 : s.toInt
                                ();
96                      });
97                  }
98                  lay->addWidget(le[r][c],r,c);
99              }
100
101         /* 读盘时不再标绿，直接纯白（与 BG_INPUT 一致） */
102         for(auto p: loadUserInput()){
103             int r=p.first, c=p.second;
104             le[r][c]->setText(QString::number(g_play[r][c]));
105             // 不再改背景色，保持默认白色
106         }
107     }
108
```

```
109     void verify(){
110         bool allOk=true;
111         for(int r=0;r<9;++r)
112             for(int c=0;c<9;++c){
113                 if(g_init[r][c]!=0) continue;
114                 int v=g_play[r][c];
115                 if(v==0){ allOk=false; continue; }
116                 if(v==g_sol[r][c]){
117                     le[r][c]->setStyleSheet(
118                         QString("QLineEdit{background:%1;border:1
                                px␣solid␣#555;}")
119                         .arg(BG_OK));
120                 }else{
121                     le[r][c]->setStyleSheet(
122                         QString("QLineEdit{background:%1;border:2
                                px␣solid␣red;}")
123                         .arg(BG_ERROR));
124                     allOk=false;
125                 }
126             }
127         QMessageBox::information(this,"验证结果",
128                                  allOk?"全部正确！":"绿色=正确，
                                     红色=错误，空白=未填");
129     }
130
131     void redo(){
132         if(redoStack.isEmpty()) return;
133         Snapshot s = redoStack.pop();
134         for(int r=0;r<9;++r)
135             for(int c=0;c<9;++c){
136                 g_play[r][c] = s.grid[r][c];
137                 le[r][c]->setText(g_play[r][c]==0?"":QString::
                    number(g_play[r][c]));
138             }
139     }
```

```
140     void pushRedoSnap(){   // 外部调用: 保存当前状态到 redo 栈
141         Snapshot s;
142         for(int r=0;r<9;++r) for(int c=0;c<9;++c) s.grid[r][c]=
                g_play[r][c];
143         redoStack.push(s);
144     }
145     void updateEntireBoard(){
146         for(int r = 0; r < 9; ++r)
147         for(int c = 0; c < 9; ++c){
148             int v = g_play[r][c];
149             le[r][c]->setText(v == 0 ? "" : QString::number(v));
150             if(g_init[r][c] != 0) continue;   // 题目给定格不动
151             le[r][c]->setStyleSheet(
152                 QString("QLineEdit{background:%1;border:1px␣solid
                        ␣#555;}")
153                 .arg(v == 0 ? BG_INPUT : BG_OK));
154         }
155     }
156
157 private:
158     QLineEdit* le[9][9]{};
159 };
160
161 //交互框架
162 #include <QLCDNumber>
163 #include <QTimer>
164 class SudokuDialog : public QDialog{
165     Q_OBJECT
166 public:
167     SudokuDialog(QWidget* parent=nullptr):QDialog(parent){
168         setWindowTitle("数独␣");
169         resize(420,480);
170         // 米黄地板
171         setStyleSheet(QString("QDialog{background:%1;}").arg(
                MICKEY_YELLOW));
```

```
172         auto* topLay=new QHBoxLayout();
173         auto* redoBtn   = new QPushButton(" ␣重做",this);
174         auto* restartBtn= new QPushButton("  ␣重启",this);
175         lcd=new QLCDNumber(this);
176         lcd->setDigitCount(8);
177         lcd->setSegmentStyle(QLCDNumber::Flat);
178         lcd->display("00:00:00");
179         timer=new QTimer(this);
180         startTime=QTime::currentTime();
181         connect(timer,&QTimer::timeout,[=]{
182             int el=startTime.secsTo(QTime::currentTime());
183             lcd->display(QTime(0,0).addSecs(el).toString("hh:mm:
                ss"));
184         });
185         timer->start(1000);
186
187         redoBtn   ->setStyleSheet(btnStyle(MICKEY_BLACK,
            MICKEY_YELLOW));
188         restartBtn->setStyleSheet(btnStyle(MICKEY_YELLOW,
            MICKEY_BLACK));
189         topLay->addWidget(redoBtn);
190         topLay->addWidget(restartBtn);
191         topLay->addStretch();
192         topLay->addWidget(lcd);
193
194         /* 棋盘 */
195         board=new SudokuBoard(this);
196         auto* lay=new QVBoxLayout(this);
197         lay->addLayout(topLay);
198         lay->addWidget(board);
199
200         /* 底部按钮 */
201         auto* hLay=new QHBoxLayout();
202         auto* verifyBtn=new QPushButton("验证",this);
203         auto* closeBtn =new QPushButton("关闭",this);
```

```
204         verifyBtn->setStyleSheet(btnStyle(MICKEY_YELLOW,"#43a047"
                ));
205         closeBtn ->setStyleSheet(btnStyle(MICKEY_YELLOW,
                MICKEY_BLACK));
206         hLay->addWidget(verifyBtn);
207         hLay->addWidget(closeBtn);
208         lay->addLayout(hLay);
209
210         connect(verifyBtn,&QPushButton::clicked,board,&
                SudokuBoard::verify);
211         connect(closeBtn ,&QPushButton::clicked,this,&QDialog::
                accept);
212         connect(redoBtn,&QPushButton::clicked,[=]{ board->redo();
                 });
213         connect(restartBtn,&QPushButton::clicked,[=]{ onRestart()
                ; });
214     }
215     void closeEvent(QCloseEvent *e) override
216     { saveUserInput(); QDialog::closeEvent(e); }
217 private:
218     SudokuBoard* board;
219     QTimer* timer;
220     QLCDNumber* lcd;
221     QTime startTime;
222     QString btnStyle(QString tc,QString bc){
223         return QString("QPushButton{color:%1;background:%2;border
                :none;padding:6px 10px;"
224                        "border-radius:4px;font-weight:bold;}").
                           arg(tc,bc);
225     }
226     void onRestart(){
227         for(int r=0;r<9;++r)for(int c=0;c<9;++c) g_play[r][c]=
                g_init[r][c];
228         board->updateEntireBoard();
229         while(!redoStack.isEmpty()) redoStack.pop();
```

```
230         startTime=QTime::currentTime();
231     }
232 };
233
234 //接口
235 extern "C" void runInteractiveGui(int sol[9][9],
236                                   int player[9][9],
237                                   int init[9][9])
238 {
239     g_sol  = sol;
240     g_play = player;
241     g_init = init;
242     saveUserInput();
243     static int dummy=0;
244     static QApplication* app=nullptr;
245     if(!QApplication::instance()) app=new QApplication(dummy,
        nullptr);
246     SudokuDialog dlg;
247     dlg.exec();
248 }
249
250 #include "sudoku_gui.moc"
```

suduku_gui.hpp

```
1  #pragma once
2  // 给纯 C 环境用的 “C” 接口
3  #ifdef __cplusplus
4  extern "C" {
5  #endif
6  void runInteractiveGui(int grid[9][9], int playerGrid[9][9], int
       init[9][9]);
7
8  #ifdef __cplusplus
9  }
10 #endif
```