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**

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问题求解，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%)

* **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>

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic forProgramming, 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 InternationalSymposium 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] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

目 录

[**任 务 书 I**](#_Toc3115526)

[**1 引言 1**](#_Toc3115527)

[1.1 课题背景与意义 1](#_Toc3115528)

[1.2 问题简介 1](#_Toc3115529)

[1.3 国内外研究现状 4](#_Toc3115530)

[1.4 课程设计的主要研究工作 5](#_Toc3115531)

[**2 系统需求分析与总体设计 7**](#_Toc3115532)

[2.1 系统需求分析 7](#_Toc3115533)

[2.2 系统总体设计 8](#_Toc3115534)

[**3 系统详细设计 10**](#_Toc3115535)

[3.1 有关数据结构的定义 10](#_Toc3115536)

[3.2 主要算法设计 12](#_Toc3115537)

[3.2.1 DPLL算法策略 12](#_Toc3115538)

[3.2.2 有关数独的算法设计 13](#_Toc3115539)

[**4 系统实现与测试 14**](#_Toc3115540)

[4.1 系统实现 14](#_Toc3115541)

[4.2 系统测试 15](#_Toc3115542)

[4.2.1 主程序测试 15](#_Toc3115543)

[4.2.2 SAT问题求解测试 17](#_Toc3115544)

[4.2.3 数独游戏测试 20](#_Toc3115545)

[**5 总结与展望 23**](#_Toc3115546)

[5.1 全文总结 23](#_Toc3115547)

[5.2 工作展望 23](#_Toc3115548)

[**6 体会 24**](#_Toc3115549)

[**参考文献 26**](#_Toc3115550)

[**附录 程序使用手册 27**](#_Toc3115551)

1 引言

**1.1 课题背景与意义**

通过对C语言和数据结构的学习，我们已经对运用高级语言设计和编写程序有了深刻的认识，并且已经具备建立结构解决复杂问题的能力。该课题“基于SAT的数独游戏求解程序”是一个大型复杂的问题，要求我们自己从建立数据结构开始，通过自己的结构实现DPLL算法，从而求解SAT问题，同时也要求我们在解决SAT问题的基础上，将其转化为数独游戏求解。该课题综合性十分强，考察了我们对于数据结构的认识与运用，对于程序和算法的实现，然后进一步培养了我们对于问题转化实现实际问题的能力。这个课程设计是对我们学习C语言和数据结构课程的一次很好的实践与检验，也是对培养我们计算机编程思维能力的一次很大的提升。

1.2 问题简介

下面我们对于上述所提到的SAT问题和数独问题作一个简单的介绍。

* **概念介绍**
* **文字(literal)和子句(clause)**

对于任一布尔变元x，x与其非“¬x”称为**文字**。对于多个布尔变元，若干个文字的或运算l1∨l2∨…∨lk称为**子句**。

只含一个文字的子句称为**单子句**。不含任何文字的子句称为**空子句**。

* **单子句规则**

若字句S中有一个单子句L，则L一定取真，从S中删除所有有L的子句，得到S1，若其为空集，则S可满足。

如果此时S1不是空集，那么对于S1中的每一个子句，若有¬L，则去掉这个文字，得到S2。

S满足当且仅当S2满足。

* **SAT问题概述**

SAT问题即为对于给定布尔变元集合以及其子句集合，若对于其子句的合取范式(CNF范式)，判断是否有一组对于布尔变元的赋值使得该CNF范式为真。当为真的时候，即我们称该问题是可以满足的，即SAT，此时我们输出给布尔变元赋值的结果。

例如我们有CNF范式F=（¬a∨b∨c）∧（¬a∨¬b）∧（¬b∨c），当a=0，b=0，c=1时候，这个公式F取真，则我们称这个公式是可满足（satisfied）的，一组解是a=0，b=0，c=1。

* **DPLL算法综述**

DPLL算法是解决SAT问题的一个经典算法，我们可以简要叙述其算法如下。

* 输入一个子句集S，使得其进入DPLL算法，若满足返回true，不满足返回为false。
* 接下来是一个循环。从S中选择单子句L，利用上述提到的单子句规则来化简L。
* 如果此时化简结果是S为空集，证明可满足，返回true。
* 如果此时S不为空集但是S中有子句是空子句，证明不可满足（无论取怎样的布尔值组都不可满足）。
* 若是其他情况，继续循环，寻找S中的单子句，重复上面三步。
* 直到S中没有单子句，此时从S中选择一个v使其成为单子句，加入S递归调用DPLL，若返回值不为true，则v变为¬v，递归调用DPLL。
* **数独游戏综述**

数独是一个在9×9的网格里面，每个单元填入1-9中的一个数字，使得其满足下列条件：

* 每行，每列，每个3×3的网格内必须都有1到9这9个数字
* 每行，每列，每个3×3的网格内1到9这9个数字只能出现一次。
* 具体格局参见下图1（其中黑色的是题目，粉色的为所填的答案）

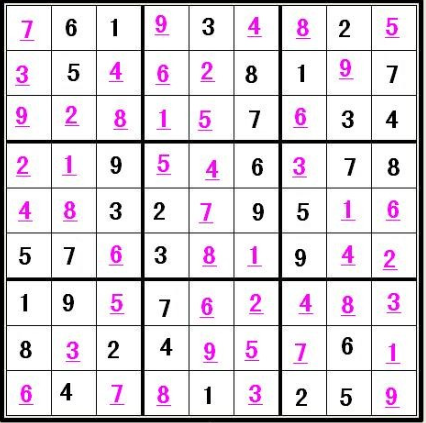


图1-1 一个完整的数独及其解答

可以看到，这个数独格格局符合以上两个条件。

* **数独转换成SAT基本思路**

将数独转换成SAT是这个工程中很重要的一步，我们可以根据以上提到的数独的两个条件进行转换。首先，我们可以把数独在某行某列填入某数转换成一个literal，比如第一行第二列的数字三，可以写成123。这样我们可以转换成SAT来解决。

如下图2，我们用这个数独演示一下数独转换成SAT的全过程：

1. **对于已经存在的数字比如2行1列的4，3行2列的1等等，创造子句：**

214 0

321 0

338 0

……

1. **对于每一行，一定要包含1到9，且每个数字只出现一次，构造子句如下：**

**第一行：**

存在1： 111 121 131 141 151 161 171 181 191 0

存在2： 112 122 132 142 152 162 172 182 192 0

……

**不存在超过一个1：**

-111 -121 0

-111 -131 0

……

**不存在超过一个2：**

-112 -122 0

-112 -132 0

……

**第二行：**

**存在1:** 211 221 231 241 251 261 271 281 291 0

……

**不存在超过一个1**：

-211 -221 0

-211 -231 0

……

1. **对于每一列，每一个3×3的块也是如此，存在某个数就将可能其位置出现写入一个子句，数的唯一性靠两两互异实现（取非写入一个子句）。**

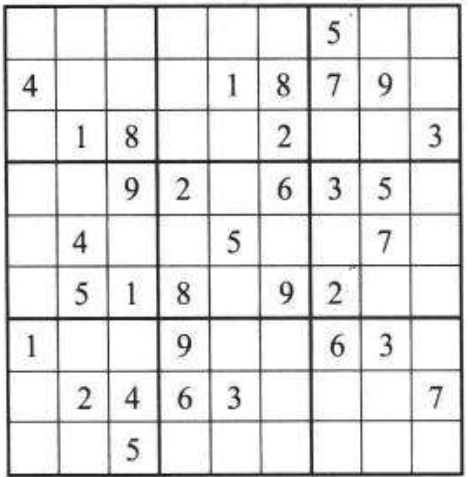


图1-2 数独转SAT演示图

1.3 国内外研究现状

运用DPLL解决SAT问题是一个很经典的算法问题，许多论文都论述了如何解决DPLL问题，从递归到算法到迭代算法，到不同数据结构的建立，大家似乎都很热衷于优化这一算法。这也为我们解决这个问题提供了许多思路。

1.4 课程设计的主要研究工作

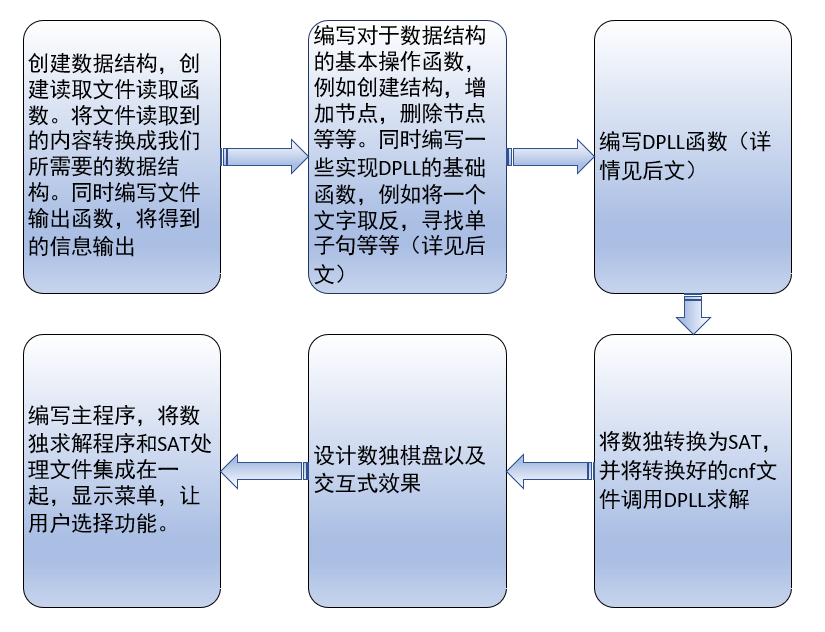


图1-3 主要工作流程图示意

以上六点是我对于此次课程设计的主要内容，下面我对于上图进行一些补充说明。

* 创建数据结构、数据结构的基本处理与文件处理文件在structure.h文件里面。里面定义了数据类型，编写了有关数据结构的所有基本操作，例如addClause、ctreateClause、destroyClause等等。
* 编写DPLL函数以及其准备函数在文件solveDPLL.h文件里面，主要包含DPLL主函数以及其辅助函数，比如找到一个单子句，将一个文字变反等等。
* 数独转换成SAT，即编写cnf的函数在文件sudokutosat.h中，编写的具体过程上文已经提到，再次不再赘述
* 设计棋盘效果以及交互式设计在文件sudokushow.h中，这个文件包括对于棋盘位置以及颜色的定位，处理用户的交互感受。同时，该文件也处理判断用户的单个输入是否正确。
* 最终数独游戏的生成和sat处理的主函数以及主菜单在文件main中。
* 除此之外，还有一个文件extension.h，这个文件中主要有一些辅助函数，比如把从文件中提取到的list打印到屏幕上，对于literal的真值进行排序等等辅助函数。

2 系统需求分析与总体设计

2.1 系统需求分析

本系统主要实现了解析cnf文件以及求解其中的SAT问题，在此基础上设计并求解了数独游戏，实现了一定的交互性和界面美化。以下简要介绍下本系统的功能。**（具体详细操作请见附录中的操作手册，其对于这个系统及其操作有更细致的介绍）**

* 设计主页面，让用户选择是解析求解cnf文件中的SAT问题还是玩数独游戏，截图如下：

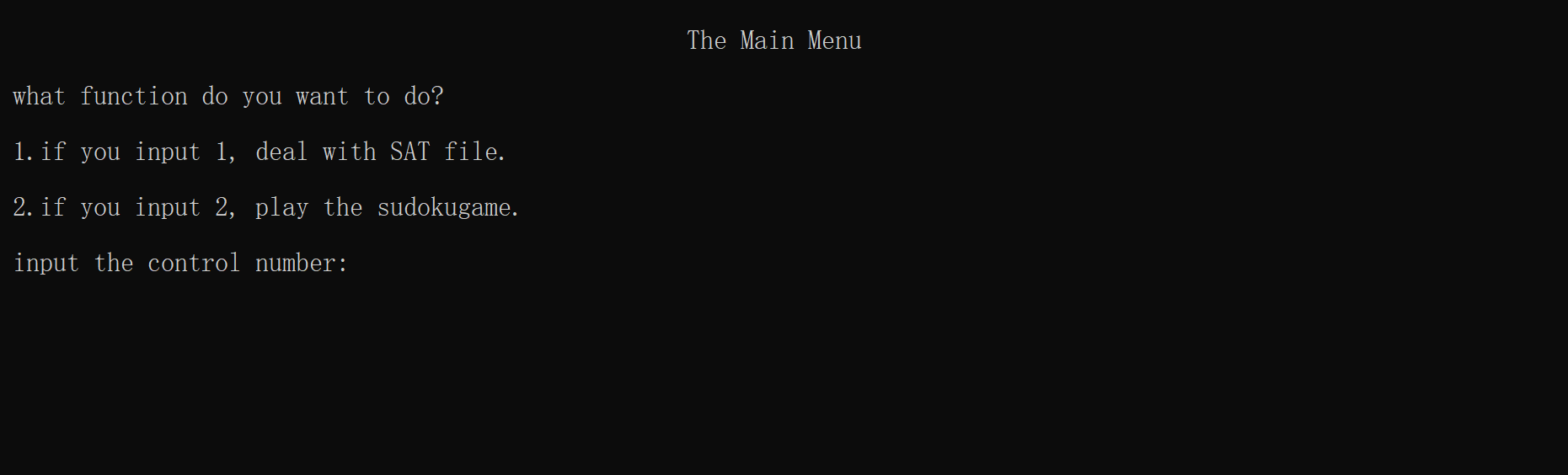


图2-1 主页面显示图

* 设计cnf文件的解析与SAT问题的求解，截图如下：

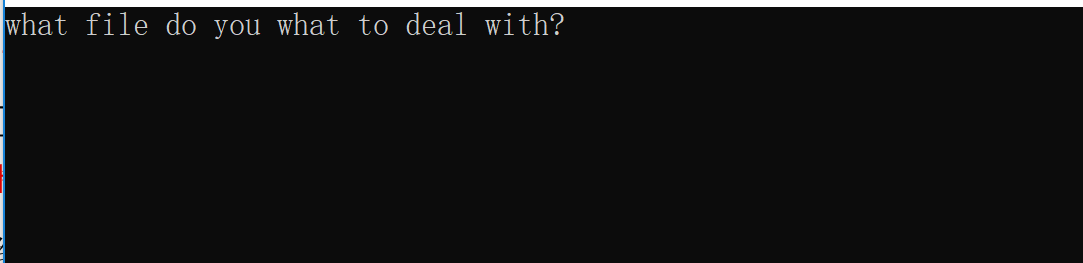


图2-2 SAT问题求解页面图

此时用户只需要输入处理文件的绝对路径即可解析并求解SAT问题，并在同目录下输出res结果文件。

* 设计数独游戏以及其交互，截图如下：

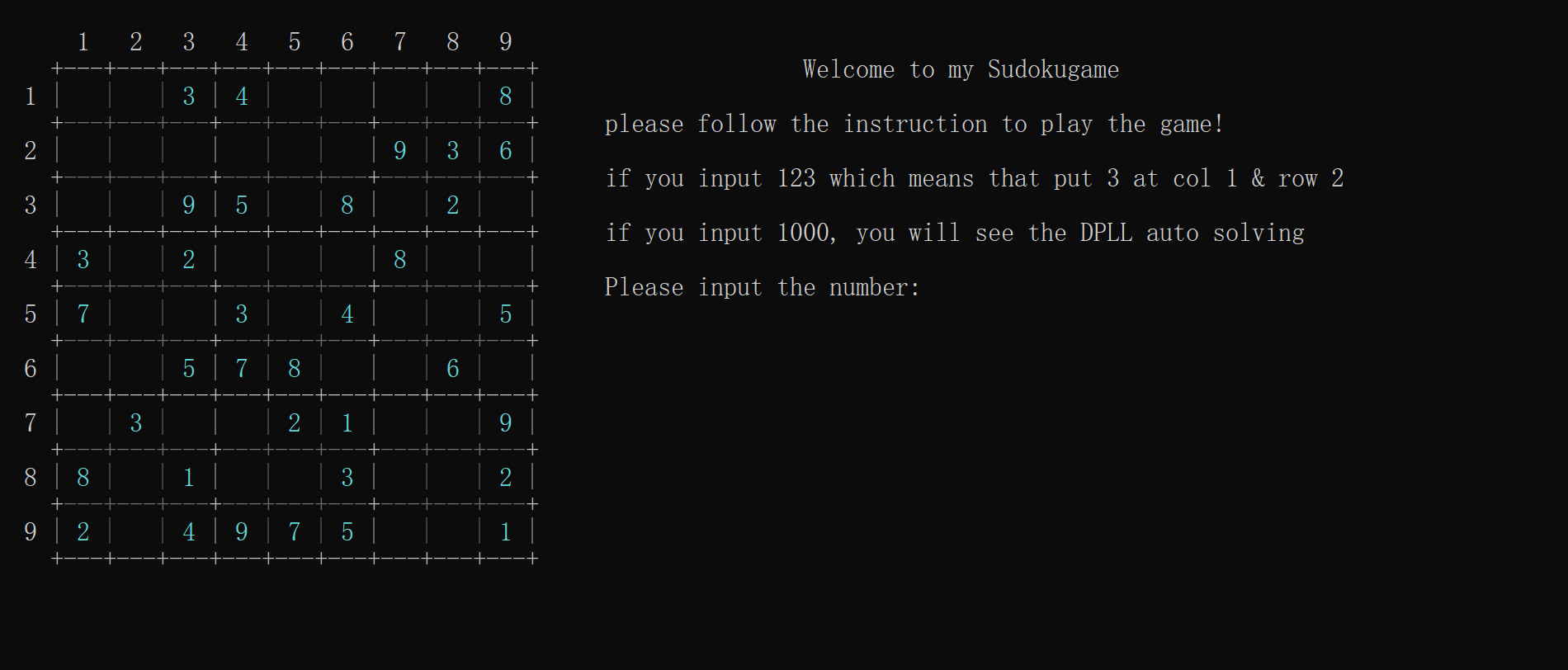


图2-3 数独问题主页面图

用户可以自己输入数字求解**（详见附录操作手册），**也可以输入1000自动求解，效果如下图

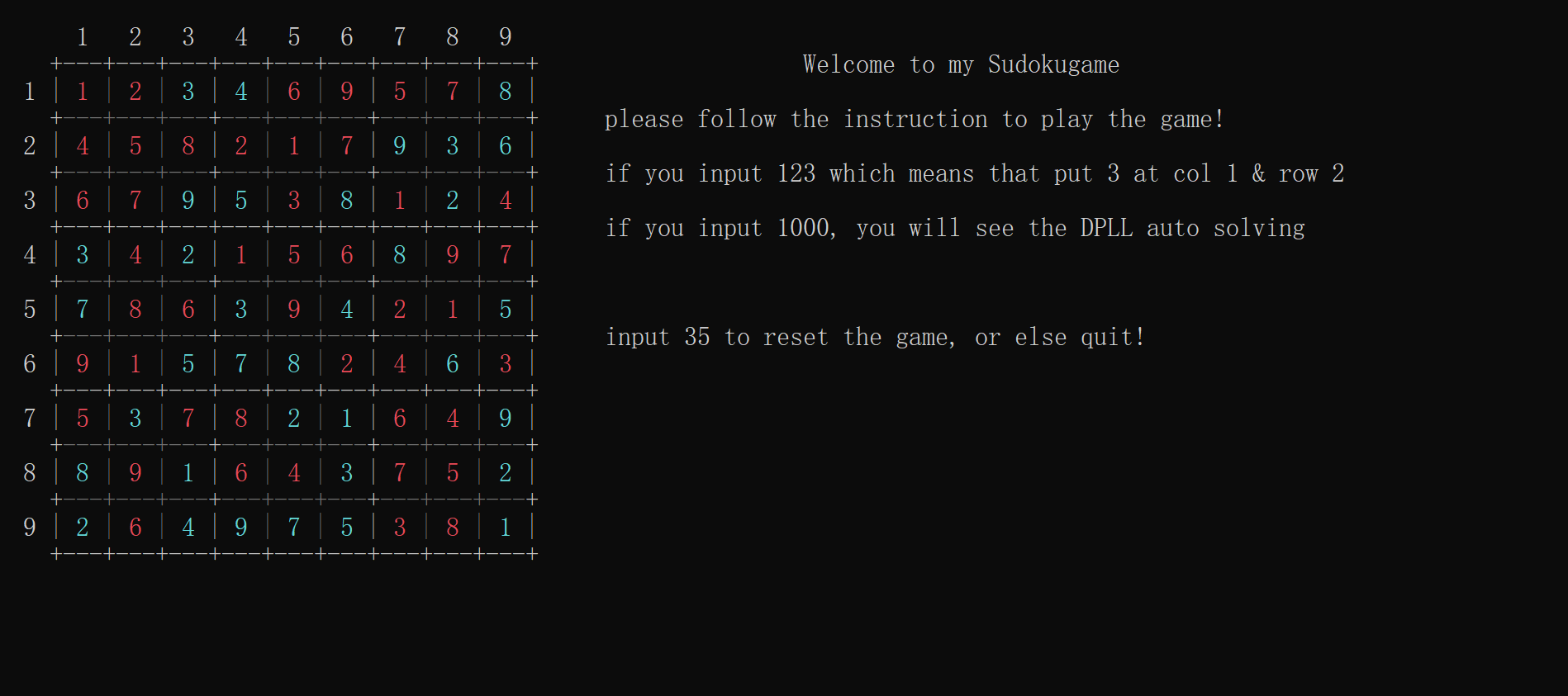


图2-4 DPLL求解示意图

2.2 系统总体设计

* **程序中主要结构可以概述如下**：
* 建立数据结构类型，编写数据结构基本操作函数（例如create，add，destroy等等）。
* 在建立的数据结构的基础上，编写一些上层函数，例如寻找构造链表中的单子句，将文字进行排序，将文字状态进行变反等等。这些函数为处理文件内容以及编写DPLL主程序打下基础。
* 编写一些输入输出函数，例如读取cnf文件，写出res文件，提取cnf文件中内容，将提取的内容用我们的数据结构进行存储等等
* 编写DPLL函数
* 编写数独转SAT函数，调用DPLL解决该问题
* 设计数独游戏格局函数，实现界面设计与交互
* 编写sat主函数，整合算法和文件输入输出功能，实现系统目的
* 编写sudoku主函数，整合数独格局的生成与用户交互效果函数以及转为SAT求解函数
* 编写main函数，调用sat函数与sudoku函数，设计交互
* **这些模块之间的关系可以由下图清晰地表示：**

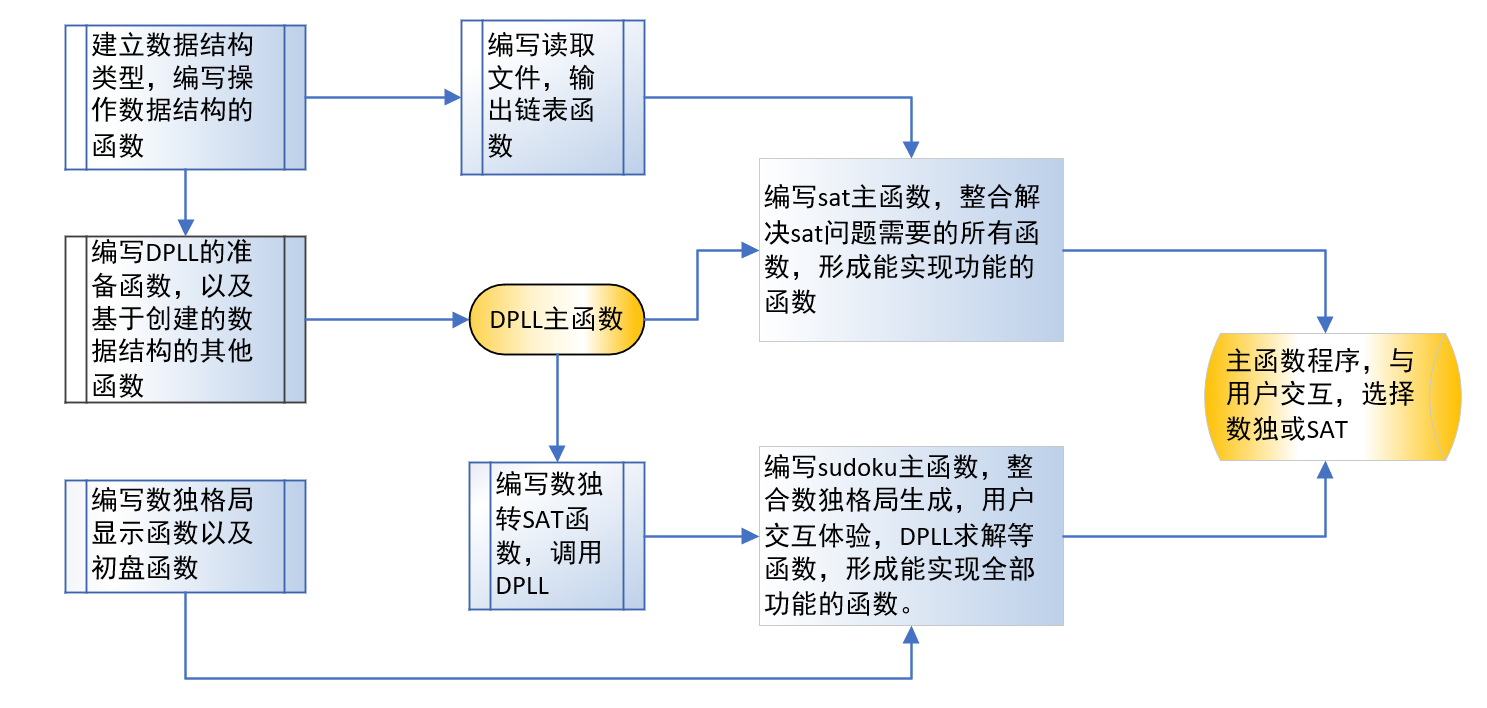


图2-5 系统各模块设计示意图

上图展示了我解决这个问题的逻辑，每个模块在源代码中都有对应的部分，例如建立数据结构和基本操作、读取写入文件等操作在structure.h，编写DPLL准备函数和DPLL主函数在文件solveDPLL中。数独格局生成和判断用户输入以及交互设计在sudokushow.h中，将数独换成sat写入cnf文件的操作在sudokutosat.h中。sat主程序，数独主程序以及main函数在main.cpp中。

3 系统详细设计

3.1 有关数据结构的定义

1. **首先，介绍我如何存储从文件读出的数据。**

建立如下的一个链表，每一句话的首节点连成一个链表，是clause链表，每一句话（每个子句），肯定有很多文字，这些文字接在句子头结点的后边形成一个个子链表，大概结构如下图：

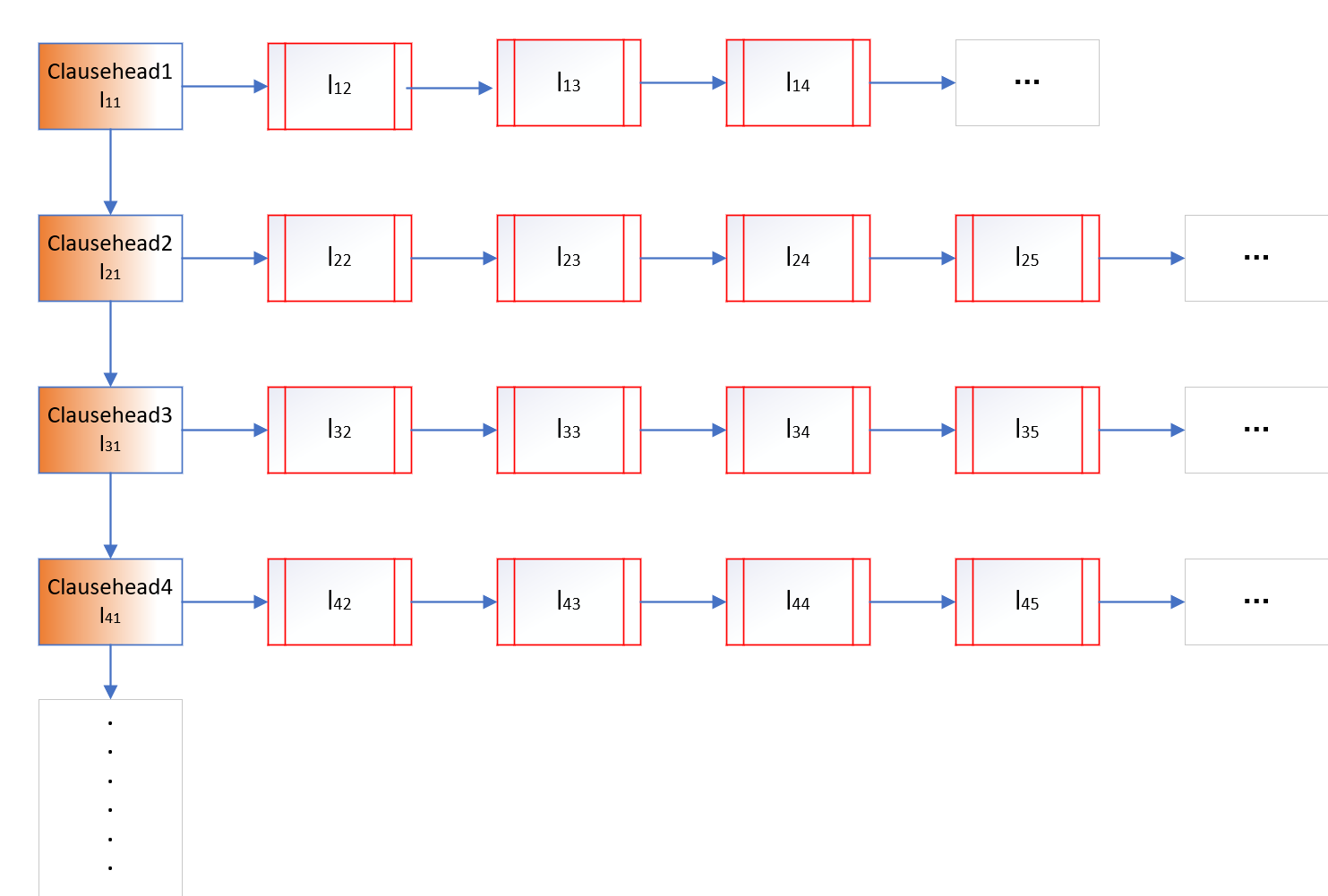


图3-1 我建立的数据结构示意图

那么上图每一行就代表着cnf文件中的每一个子句（遇到0结束），那么有多少行代表cnf文件中有多少行，即有多少子句。

1. **介绍基于该数据结为实现DPLL而建立的数据结构。**

在实现DPLL算法时，往往我们不能通过单子句规则一步到位，很多时候需要回溯，这是若我们已经删除了相应节点再恢复起来会很困难。为解决这个问题，我的策略是，为每一个子句，每一个变量都设置一个状态量state，如果要删除这一句，只需要把状态从true改为false，在最后输出结果的时候不考虑为false的子句或者文字即可。这样我们只需要多建立一个结构Memory实时记录变为false的文字的位置即可，这样恢复改文字与该文字变反会容易很多。

下面简要描述我建立的结构以及结构中的数据类型（详见源程序structure.h）有Clause、literal、Memory、Sign四个结构。

**对于四个结构简介如下：**

|  |  |
| --- | --- |
| **结构名** | **主要功能** |
| **Clause** | 子句链表，连接每个子句的句首 |
| **Literal** | 文字链表，由句首的子句指针指出，连接该子句所有文字 |
| **Memory** | 记忆链表，记忆要删除文字的位置 |
| **Sign** | 符号链表，记录每个文字取的真值 |

表3-1 数据结构功能介绍

**下表介绍了这些结构的主要数据类型：**

|  |  |  |
| --- | --- | --- |
| **结构名** | **数据类型** | **数据意义** |
| **Clause** | clauselen | 子句的长度 |
| location | 子句的位置（编号）（如上一节结构图） |
| state | 布尔值，判断该句是否存在 |
| \*next | 指针类型，指向下一个子句 |
| \*sub | 指针类型，指向下一个 |
| **Literal** | location | 这个文字在该句中编号 |
| state | 布尔值，判断该文字是否存在 |
| num | 这个文字代表的数字 |
| \*next | 指针，指向下一个文字 |
| **Memory** | C\_location | 记录state变为false文字所在句的位置 |
| L\_location | 记录state变为false文字在句中的位子 |
| \*next | 指针，指向下一个变为false的节点 |
| **Sign** | num | 记录文字编号 |
| sgn | 记录文字取真还是假（真为1假为-1） |
| \*next | 指针，指向下一个文字 |

表3-2 结构中数据类型的介绍

3.2 主要算法设计

3.2.1 DPLL算法策略

* **算法概述**
* 考虑子句集S（即链表中是否有单子句），若有，用单子句规则化简S，若正确（S为空），DPLL函数输出true，若错误（S不为空且S中有空子句），DPLL函数输出false。
* 若是其他情况，循环执行上一步过程，直到没有单子句为止
* 若是执行完上面步骤还没有输出结果，则基于某种策略从中选取一个文字v，使其成为一个单子句，并入S，递归执行DPLL(S)。
* 若是此操作返回true，则DPLL函数返回true，不然将v变反，递归执行DPLL
* **我的DPLL实现策略（代码详情见源文件solveDPLL.h）**

在实验初期遇到了一个问题，那就是如果我选择了一个变元v取执行DPLL，结果没有返回true，这时候我们要恢复整个链表，并把变元v变反，可是由于我们已经删除了含有v的文字或者子句，此时将要恢复过来不仅要找到删除位置，还要记录信息，十分复杂，无法很好地实现。下面我将分步给出这个问题的解决方案和DPLL的解决策略：

* 在这里我对每一个子句和每一个文字都添加了一个state布尔变量（如上一节所示），如果这个文字或者子句没有在该链表中删除，则它的state是true，若删除了就为false。
* 在这里我用结构Memory记录了删除的文字（状态为false的文字）（若是子句则记录其中每一个文字）的位置和它的数字，到时候如果要恢复就只用遍历Memory链表去寻找位置，便能很快地定位到原链表（子句集S），恢复该数字。
* 在执行DPLL时候，同时创建Sign链表，用sgn记录每一个文字的布尔真值，若为真取1，不为真取0

3.2.2 有关数独的算法设计（详见源代码sudokushow.h）

在实现数独的时候，有一些策略在此作简要说明

1. **数独生成策略**

* 随机排列生成一个数独（详见sudokushow.h中的Sudoku函数）
* 随机挖去数独中的数字，直到剩余的数字符合用户input的规定（详见sudokushow.h中的digging函数）
* 在屏幕上定位，显示棋盘边界并输入挖去之后的初始数独（详见sudokushow.h中的Boundary函数）

**2. 用户输入判断策略**

* 判断填入位置是否已经有数字
* 判断填入的数字是否符合符合规定（同一行、同一列、同一块是否已经有该数字出现）
* 以上代码详见sudokushow.h中的Flag函数以及main.cpp中的sudoku主函数。

**3. 转SAT自动求解策略**

* 数独转SAT问题已在第一章1.2节问题概述中的数独转换成SAT基本思路作了详尽的介绍。
* 转换成SAT问题后，调用DPLL函数求得结果，取真值的项为填入项，将结果存入到数组sudoku中。
* 将结果输出。以上步骤详见sudokutosat.h和main.cpp中的sudoku函数。

4 系统实现与测试

4.1 系统实现

* **编译IDE**：用devc++即可，支持C11
* **系统功能实现总流程（前面几章节都已有叙述，这里做一个总结）**

1. 建立数据结构，创建基本操作函数（create，add，destroy等）
2. 从文件中提取信息存入建好的数据结构
3. 编写DPLL程序
4. 将数独转为sat函数
5. 数独格局生成函数
6. 设计交互，总合sat功能和数独功能

* **DPLL算法实现的结构**

上一章已经详细说明了算法的思路和求解过程，这一章这里的内容只展示函数的调用。

所有提到的函数可在solveDPLL.h或者extesion.h中找到。

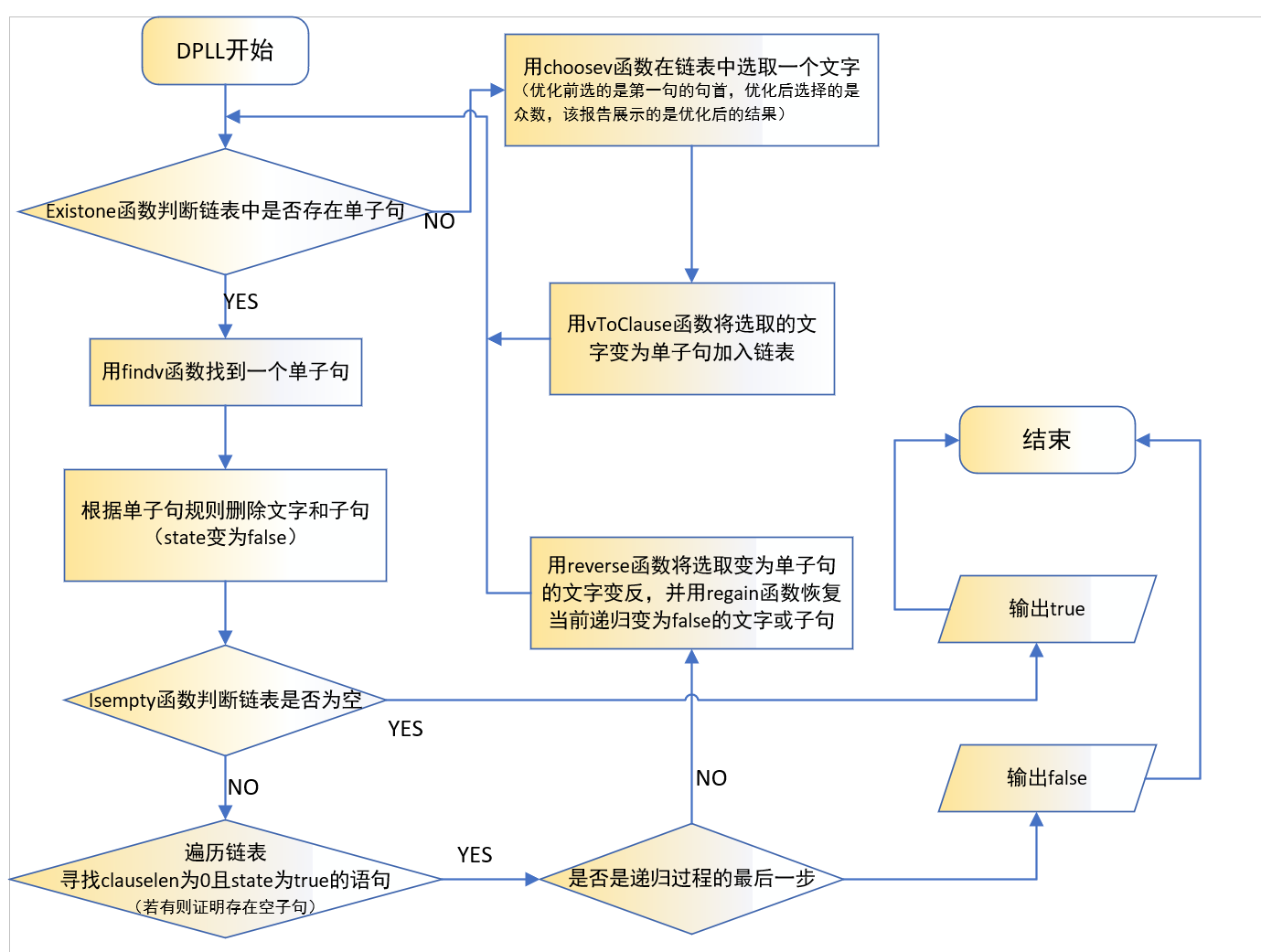


图4-1 DPLL结构流程示意图

上图为我的DPLL实现的整个实现结构。算法思路介绍见3.2.1节DPLL算法策略

* **数独实现结构**

上一章已经详细说明了数独实现的思路和求解过程，这一章这里的内容只展示函数的调用。

所提到的函数在sudokutosat.h和sodokushow.h中可找到

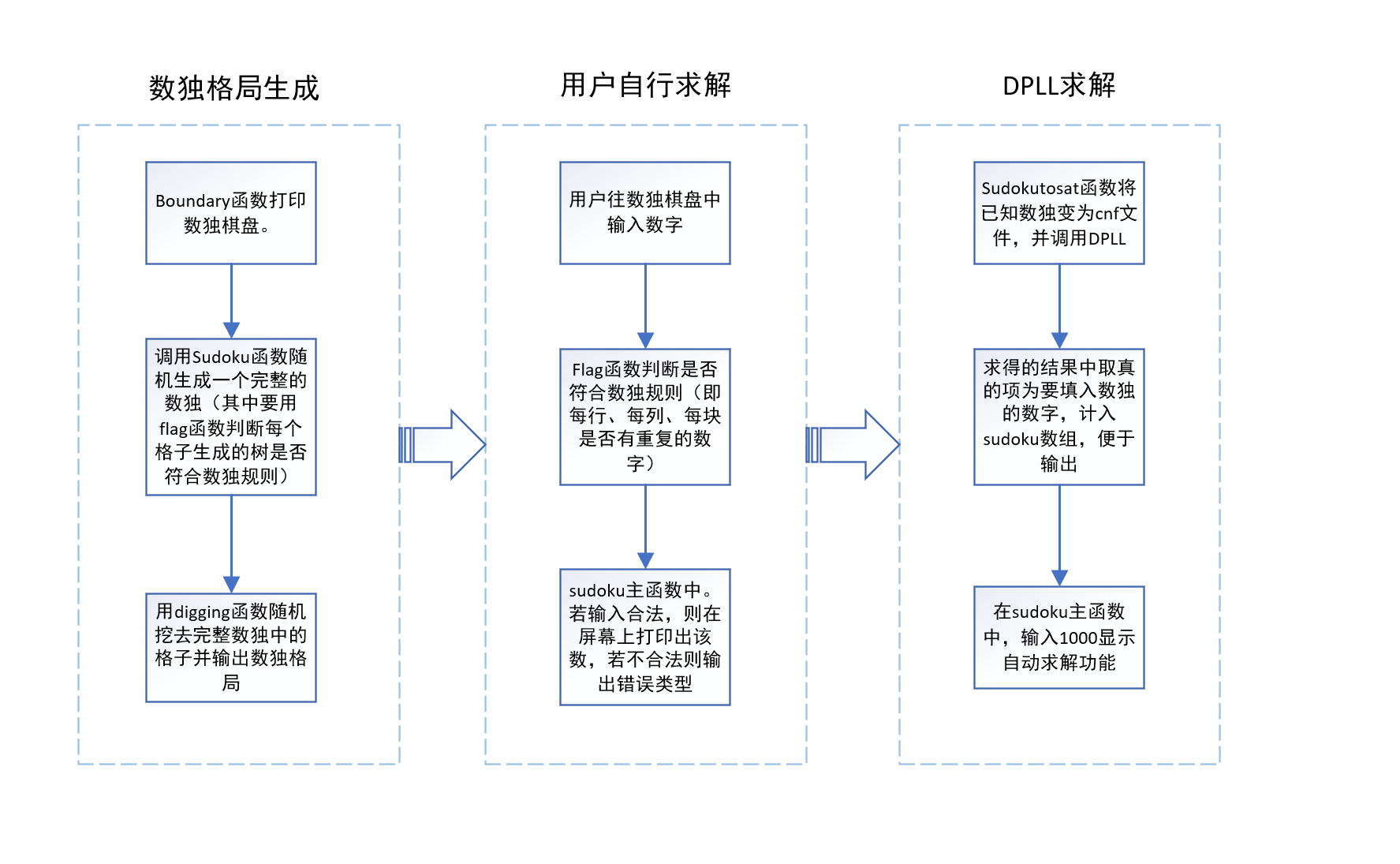


图4-2 数独实现结构示意图

上图为我的数独游戏的实现结构。算法思路介绍见3.2.2节有关数独的算法策略

4.2 系统测试

**本章节主要展示程序测试效果，若想了解程序使用方法，详见目录用户使用手册。**

4.2.1 主程序测试

* 该模块内容是程序主菜单，和用户交互，提示用户是求解SAT问题还是玩数独游戏。源代码见main.cpp中的main函数。
* 编译main.cpp即可弹出该界面，测试结果如下图。

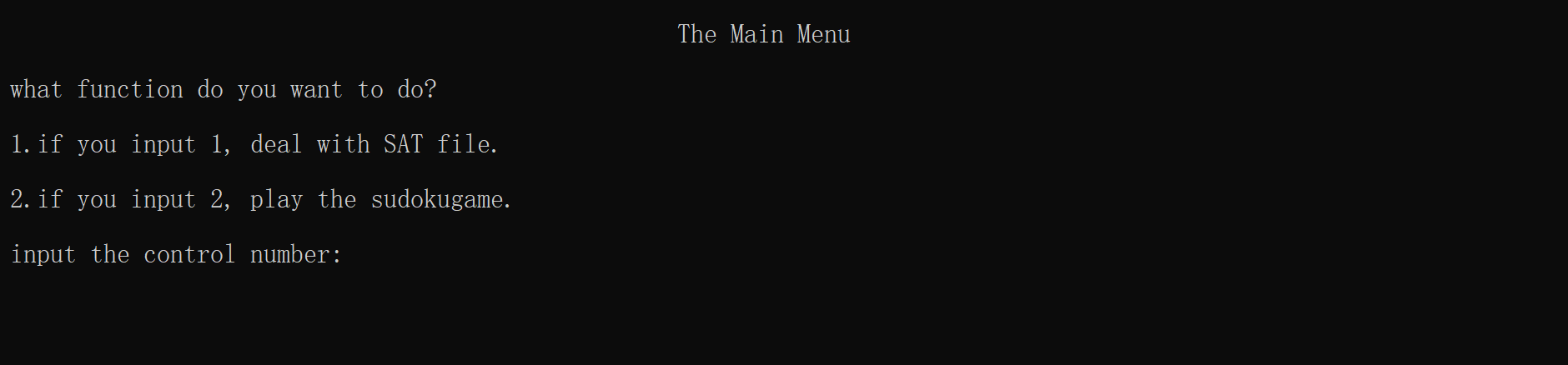


图4-3 主页面菜单示意图

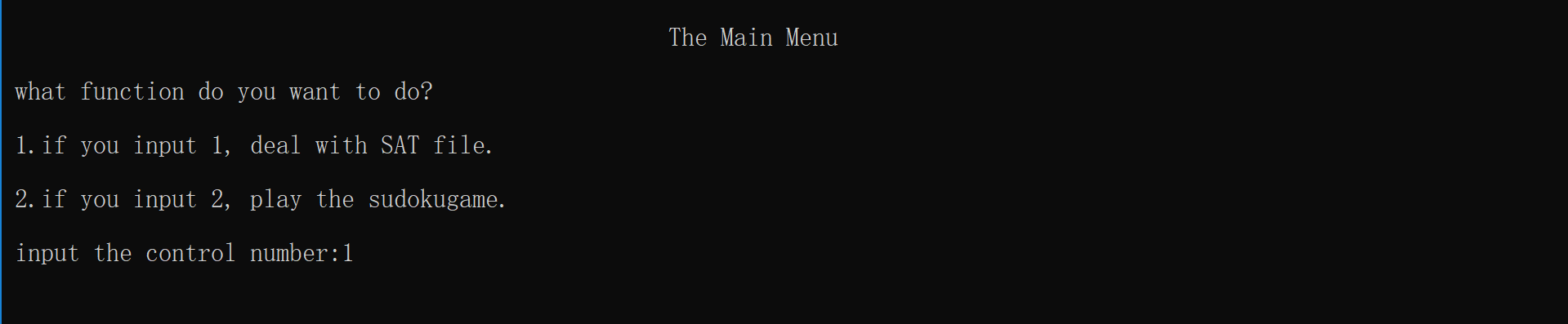
* 输入控制1跳转到求解SAT界面，测试结果如下图。  
  

图4-4 输入1进入SAT界面

跳转后界面测试结果如下，此后输入你要处理的文件。



图4-5 跳转SAT后界面示意图

* 输入控制2跳转到数独游戏界面，测试结果如下图。

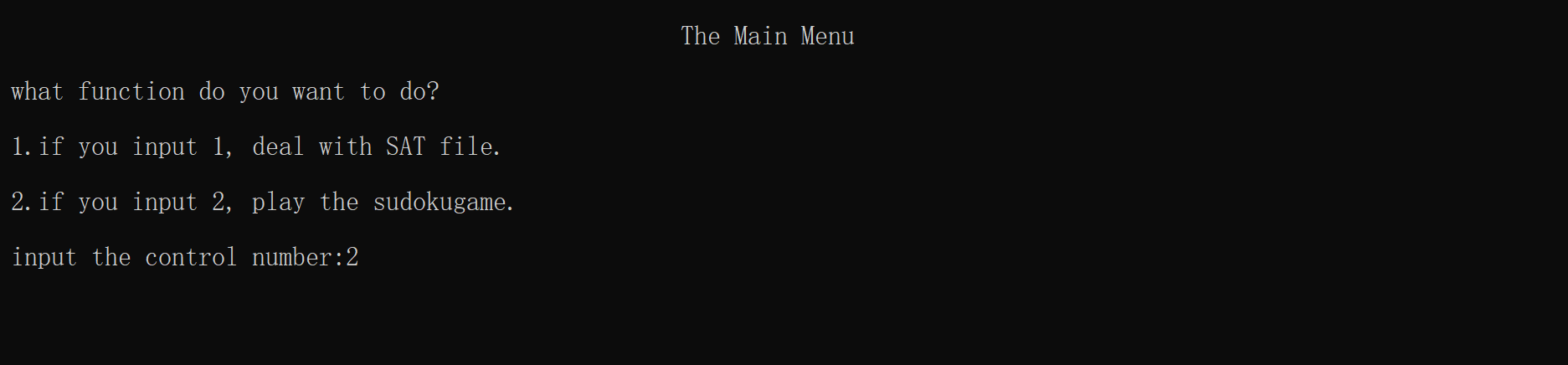
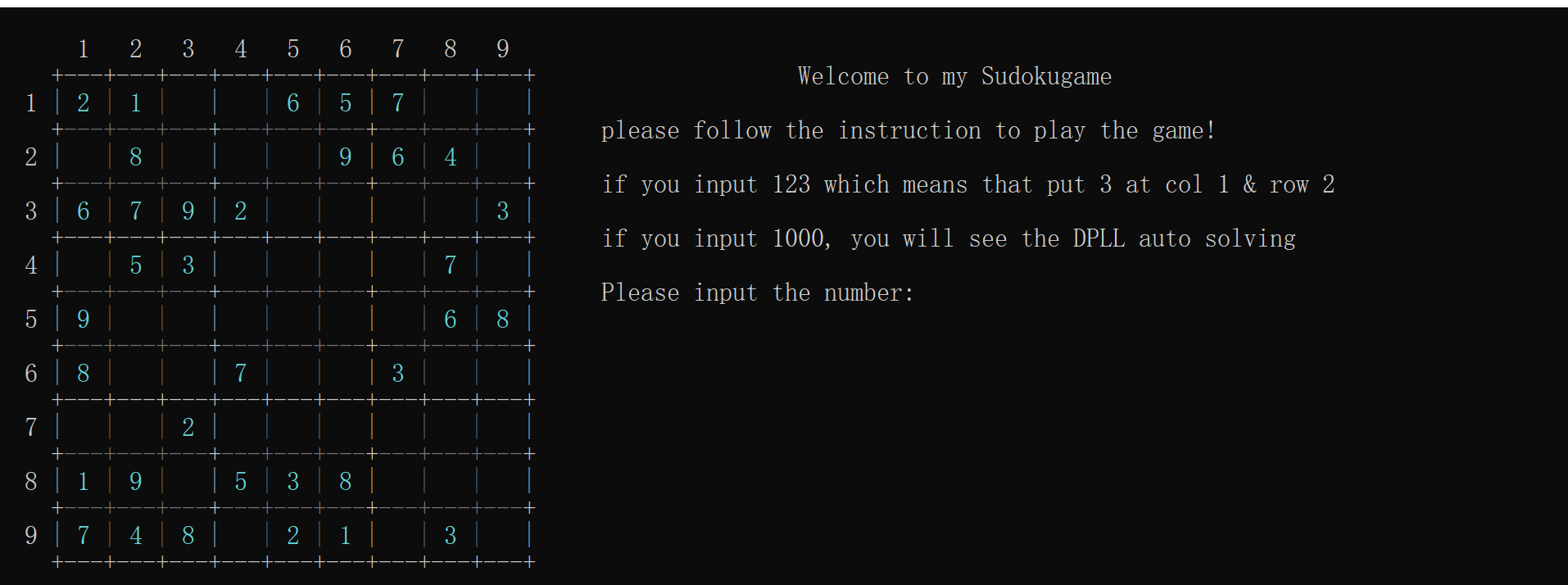


图4-6 输入2跳转数独界面

跳转后界面测试结果如下，此后可以开始玩数独游戏



**图4-7 跳转数独游戏界面后示意图**

4.2.2 SAT问题求解测试

注意：该板块所有算例运行均为优化之后的程序。（优化前的可参考包内的测试样例文件夹中的优化前存储文件夹，后面也将列表展示部分样例的优化比例。）

* **基准算例测试展示**
  + **功能测试**
    - **sat-20.cnf测试如下**（链表输出过长，在报告中不方便展示）：

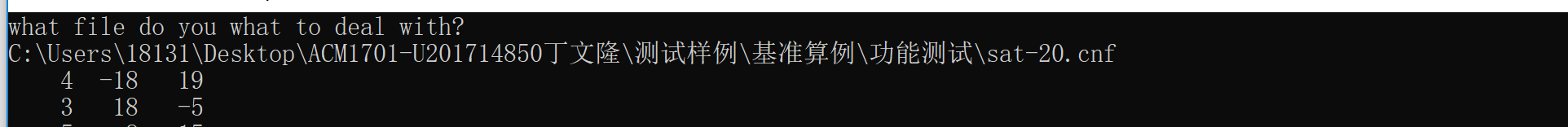




图4-8 sat-20.cnf测试输出结果示意图

证明这是一个满足算例，输出的res文件截图如下：

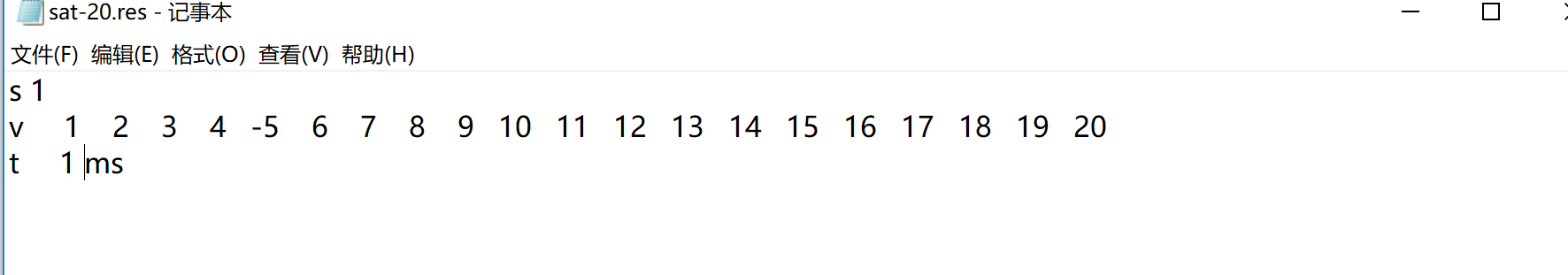


图4-9 sat-20.cnf相应输出的res文件输出示意图

S为1代表满足算例，v后面跟着文字的真值，DPLL运行时间t为1ms。

* + - **unsat-5cnf-30.cnf测试结果如下：**

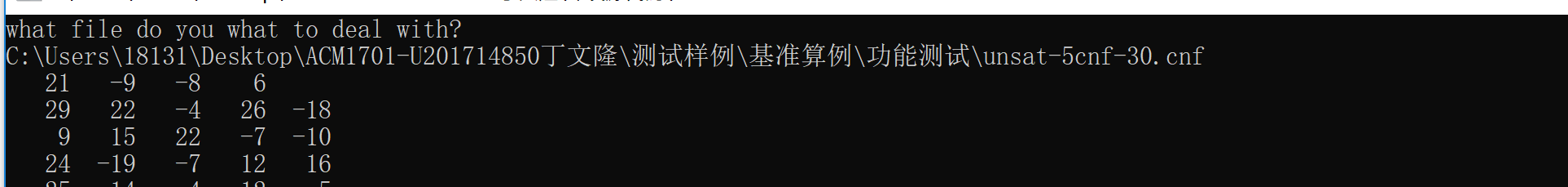




图4-10 unsat-5cnf-30.cnf测试输出结果示意图

证明这是一个不满足算例，输出的res文件如下：

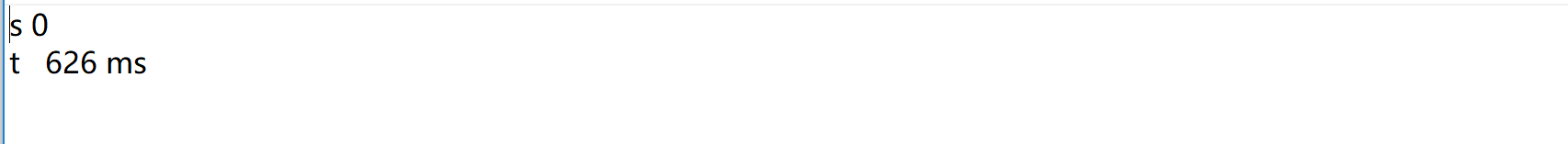


图4-11 unsat-5cnf-30.cnf相应输出的res文件输出示意图

S为0表示不满足，DPLL运行时间为626ms。

* + **性能测试**
    - **ais10.cnf测试结果如下**

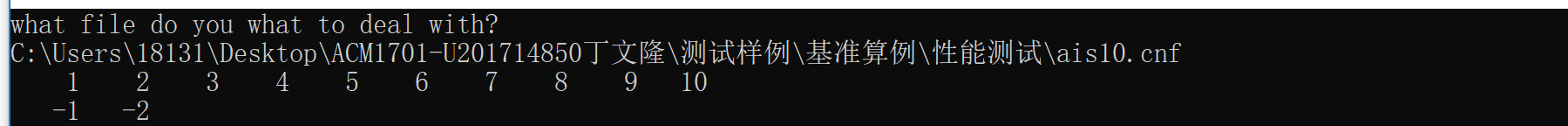




图4-12 ais10.cnf测试输出结果示意图

证明这是一个可满足算例，输出的res文件如下：

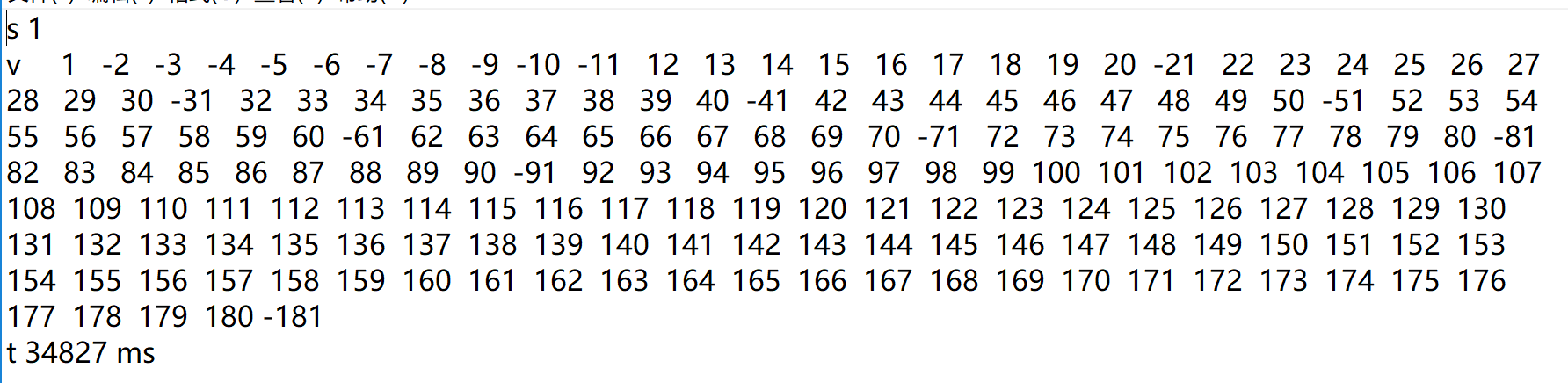
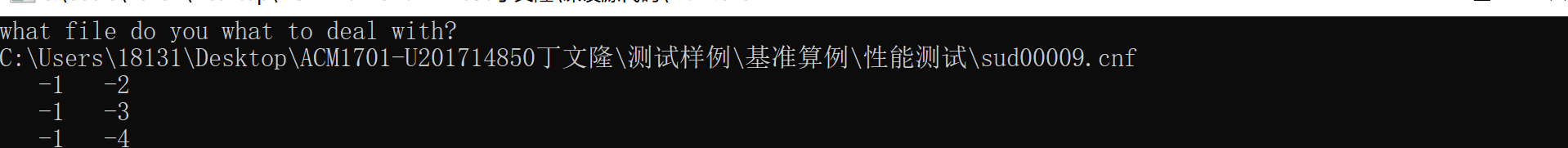


图4-13 ais10.cnf相应输出的res文件输出示意图

S为1代表可满足算例，v后面输出真值，DPLL运行时间为34.827s

* + - **sud00009.cnf测试结果如下**



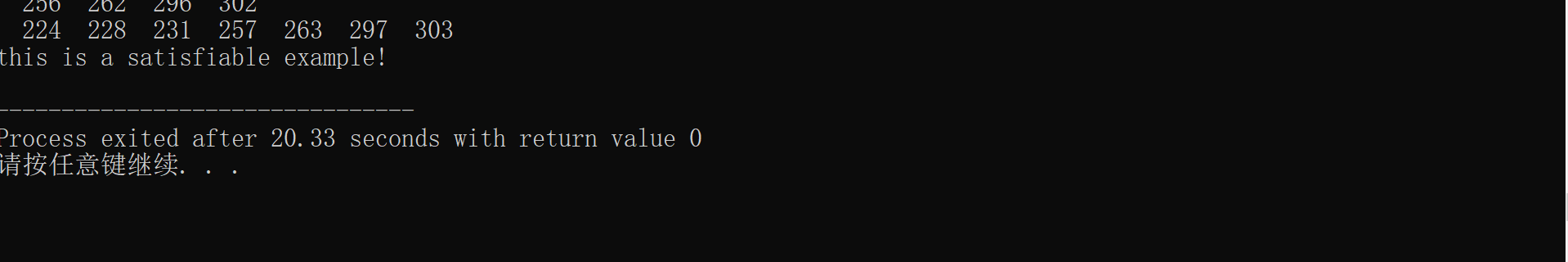


图4-14 sud00009.cnf测试输出结果示意图

这是一个可满足算例，输出res文件如下：

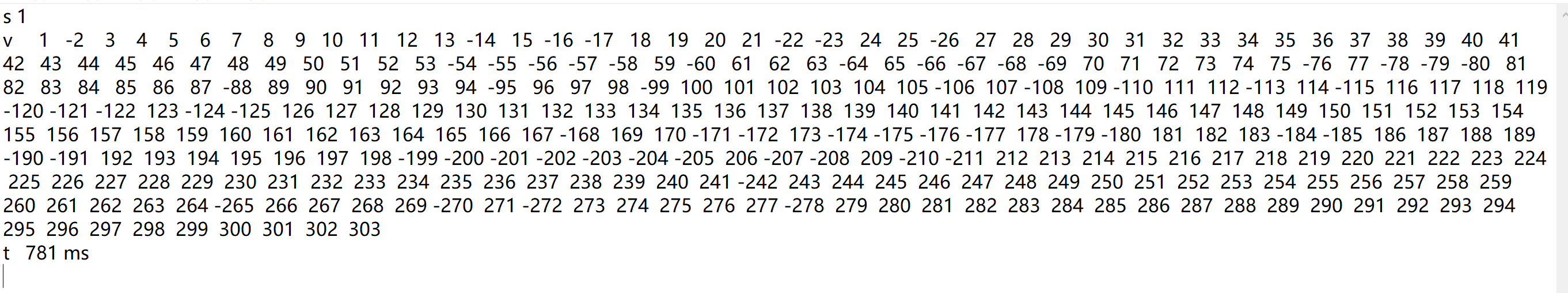


图4-15 sud00009.cnf相应输出的res文件输出示意图

S为1代表可满足算例，v后面输出真值，DPLL运行时间为781ms。

* + - **优化比例**

（注意：优化策略为选取众数，对于一些小样例，优化效果可能出现的偏差较大，性能测试的两个样例能更好地反应优化水平，样例优化率大概在15%-30%。这些样例优化前的文件可以在测试样例的优化前存储文件夹中找到）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件名 | 所属类型 | 是否满足 | 优化前运行时间 | 优化后运行时间 | 优化比例 |
| sat-20.cnf | 基准算例-功能测试 | 是 | 7 ms | 1 ms | 86% |
| unsat-5cnf-30.cnf | 基准算例-性能测试 | 否 | 627 ms | 626 ms | 0.2% |
| ais10.cnf | 基准算例-性能测试 | 是 | 41363 ms | 34827 ms | 16% |
| sud00009.cnf | 基准算例-性能测试 | 是 | 972 ms | 781 ms | 20% |

表4-1 SAT部分样例优化率

* + - **其他测试**

（测试的其他样例不一一列举，这里将测试过的部分样例用下表表示（这里展示的都是优化之后的时间）

|  |  |  |  |
| --- | --- | --- | --- |
| 文件名 | 所属类型 | DPLL运行时间 | 是否满足 |
| sat-20.cnf | 基准算例-功能测试 | 1 ms | 是 |
| unsat-5cnf-30.cnf | 基准算例-性能测试 | 626 ms | 否 |
| ais10.cnf | 基准算例-性能测试 | 34827 ms | 是 |
| sud00009.cnf | 基准算例-性能测试 | 781 ms | 是 |
| bart17.shuffled-231.cnf | 满足算例-M | 16 ms | 是 |
| problem5-200.cnf | 满足算例-M | 47 ms | 是 |
| problem12-200.cnf | 满足算例-M | 64 ms | 是 |
| sud00001.cnf | 满足算例-M | 379 ms | 是 |
| sud00009.cnf | 满足算例-M | 1348 ms | 是 |
| sud00012.cnf | 满足算例-M | 126 ms | 是 |
| sud00021.cnf | 满足算例-M | 3852 ms | 是 |
| sud00079.cnf | 满足算例-M | 549 ms | 是 |
| sud00082.cnf | 满足算例-M | 64 ms | 是 |
| sud00861.cnf | 满足算例-M | 140 ms | 是 |
| tst\_v200\_c210.cnf | 满足算例-M | 1 ms | 是 |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 满足算例-S | 938 ms | 是 |
| problem1-20.cnf | 满足算例-S | 15 ms | 是 |
| problem2-50.cnf | 满足算例-S | 16 ms | 是 |
| problem3-100.cnf | 满足算例-S | 31 ms | 是 |
| problem6-50.cnf | 满足算例-S | 1 ms | 是 |
| problem8-50.cnf | 满足算例-S | 15 ms | 是 |
| problem9-100.cnf | 满足算例-S | 32 ms | 是 |
| problem11-100.cnf | 满足算例-S | 32 ms | 是 |
| tst\_v25\_c100.cnf | 满足算例-S | 1 ms | 是 |
| tst\_v10\_c100.cnf | 不满足算例 | 1 ms | 否 |
| u-problem7-50.cnf | 不满足算例 | 1 ms | 否 |
| u-problem10-100.cnf | 不满足算例 | 30 ms | 否 |
| ais6.cnf | 其它可供选择的算例-ais | 16 ms | 是 |
| ais8.cnf | 其它可供选择的算例-ais | 755 ms | 是 |
| ais10.cnf | 其它可供选择的算例-ais | 34671 ms | 是 |

表4-2 SAT测试样例及其输入结果一览图

4.2.3 数独游戏测试

* + - **首先在主菜单界面输入2，进入数独游戏界面，界面如下：**

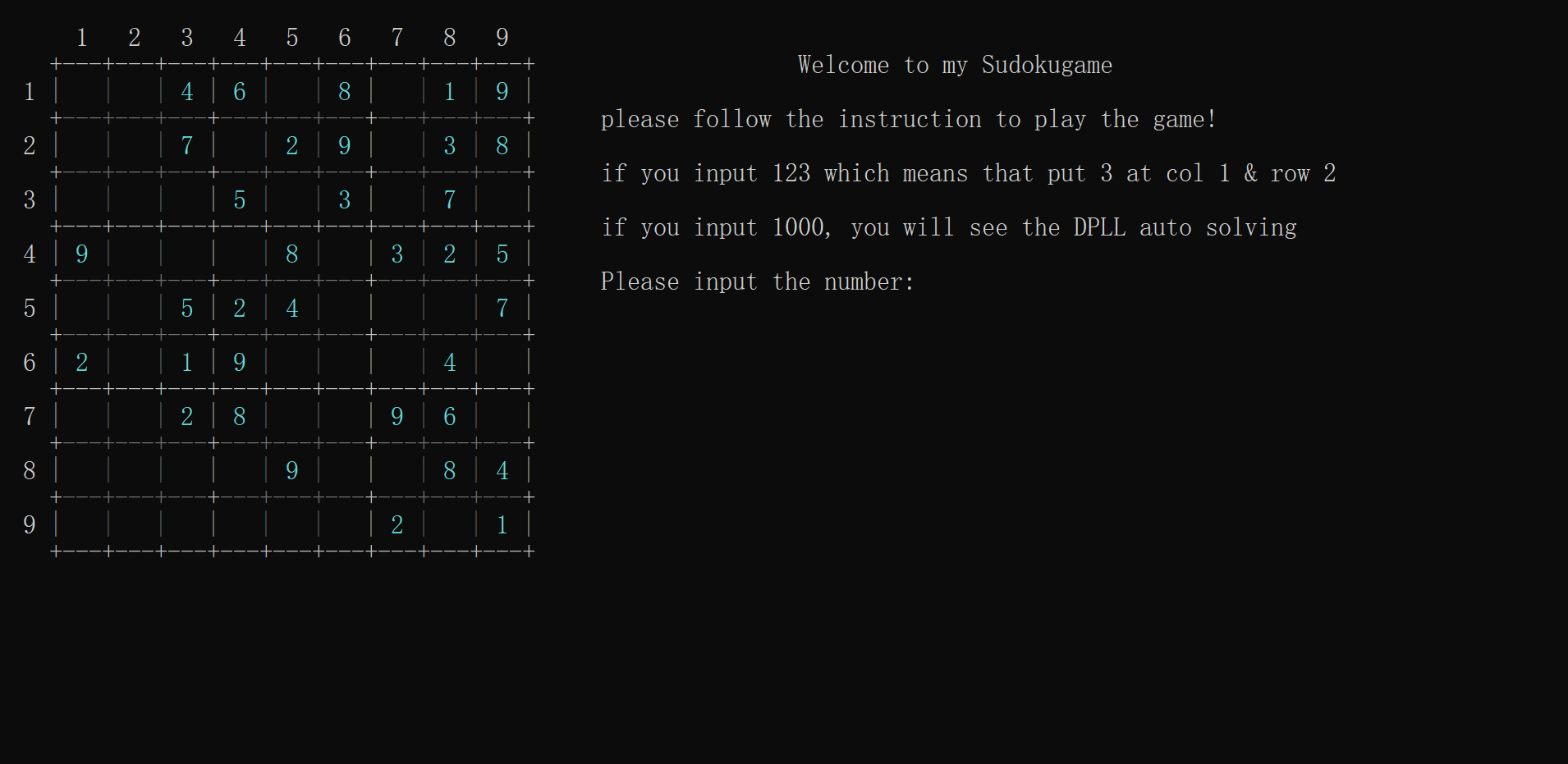


图4-16 数独游戏界面示意图

* + - **此时会生成cnf文件，在用户指定的路径中生成。（注意一定要设置路径，不然无法生成cnf文件）**
      * 路径设置方式

打开源文件main.cpp，找到对应位置，如下截图：

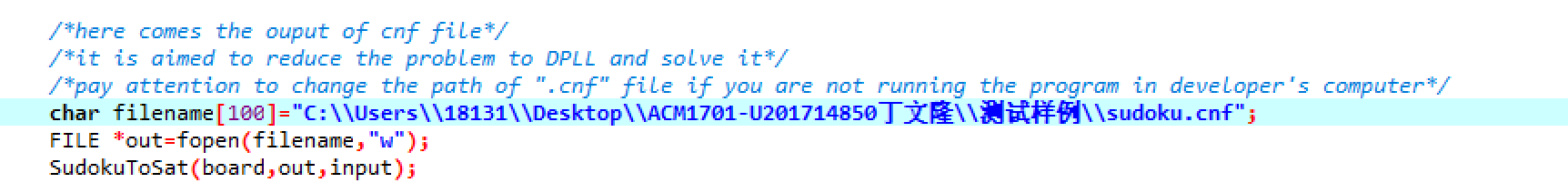


图 4-17 cnf文件路径修改位置示意图

在filename数组中输入你想要的位置，注意，根据c语言语法，必须使用双斜杠而不是单斜杠。

* + - * 在指定路径中会生成cnf文件，截图如下：

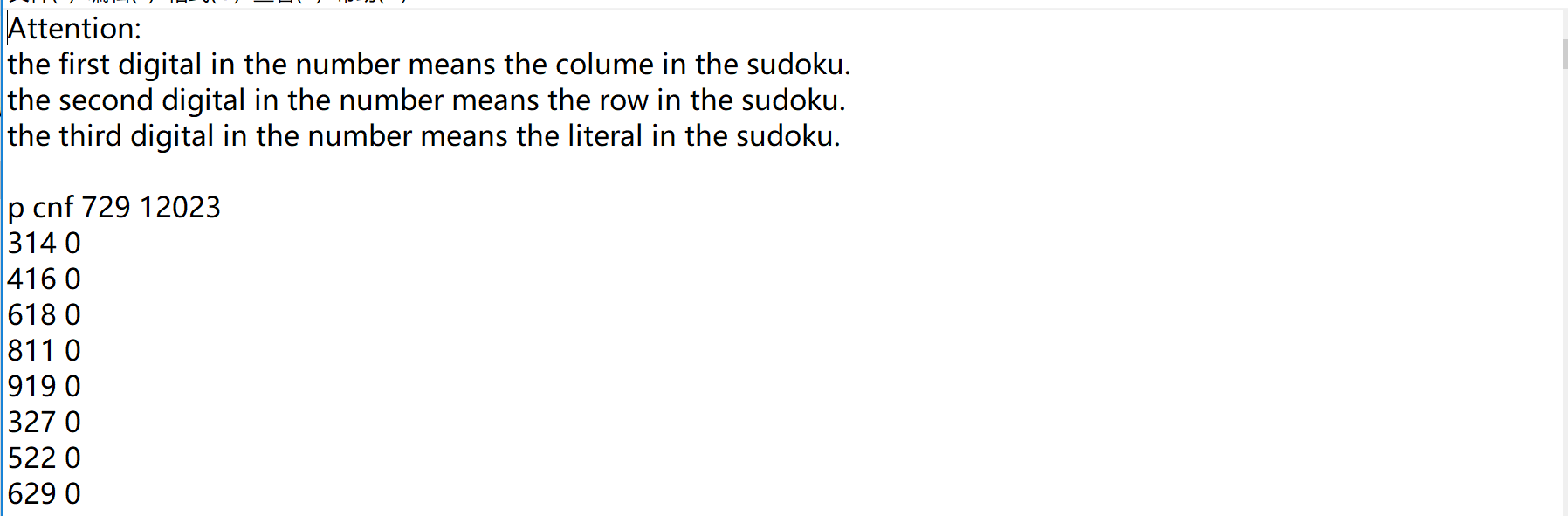


图4-18 生成的cnf文件示意图

* + - **接下来是用户输入的环节（这一章节详情见用户手册）**

**注意，输入的三位数，第一位是列，第二位是行，第三位是要输入的数字。**

* + - * 输入正确，屏幕上会用黄色在对应位置打印出该数，比如输入113，截图如下：

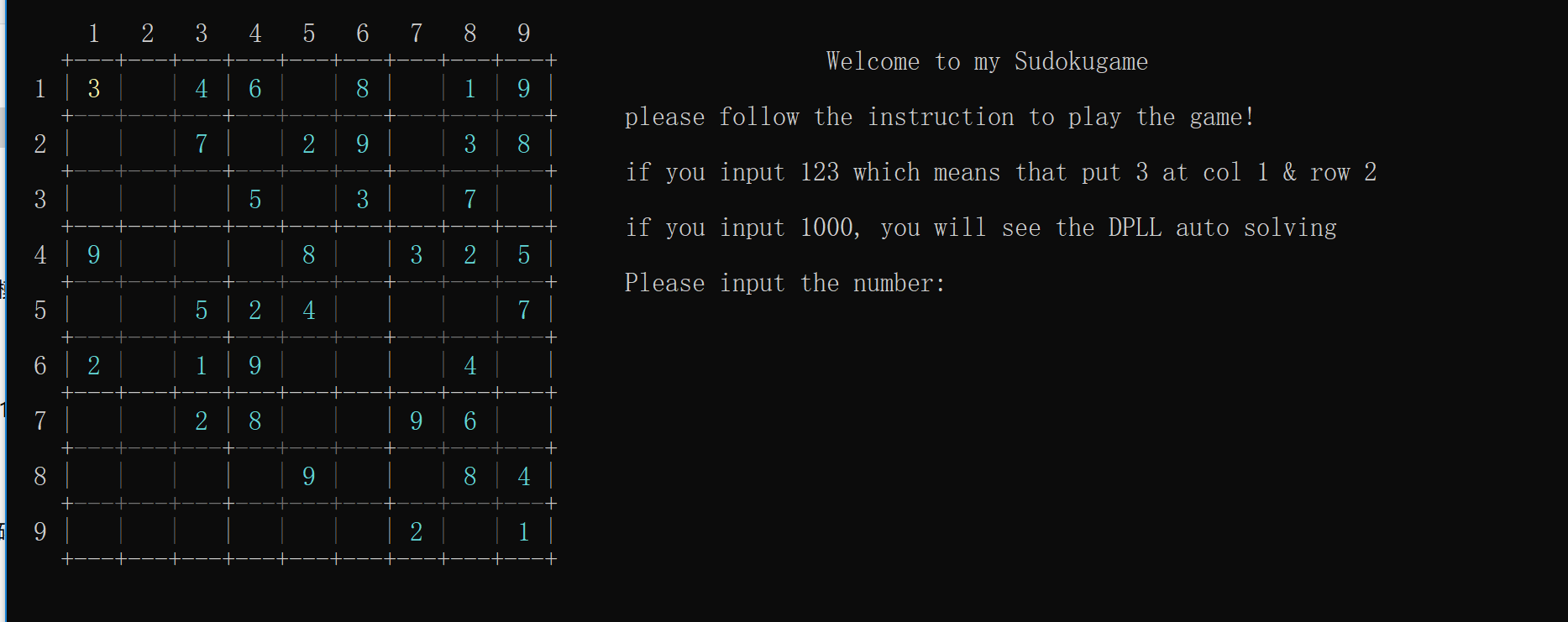


图4-19 输入正确示意图

* + - * 若输入的数字不符合数独规则，即同行同列或者同块中已经有相同数字，系统会抛出错误提示，如下图所示：

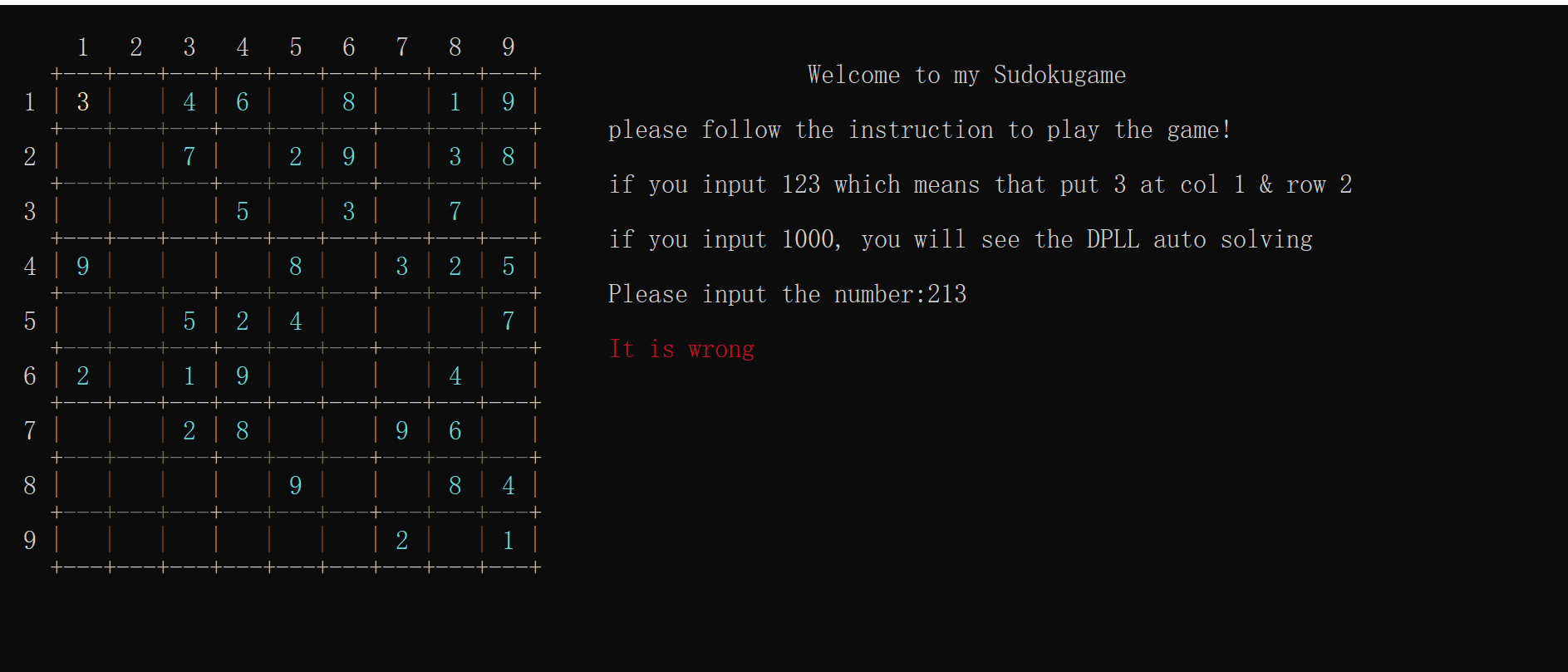


图4-20 输入错误示意图

这代表在第二列第一行输入数字3，而3已经存在于同一行中，会抛出错误提示，错误提示会在短时间内消失，方便下一次再填入数据。

* + - **用DPLL自动求解测试**

在屏幕上输入1000，会出现DPLL自动求解结果，截图如下：

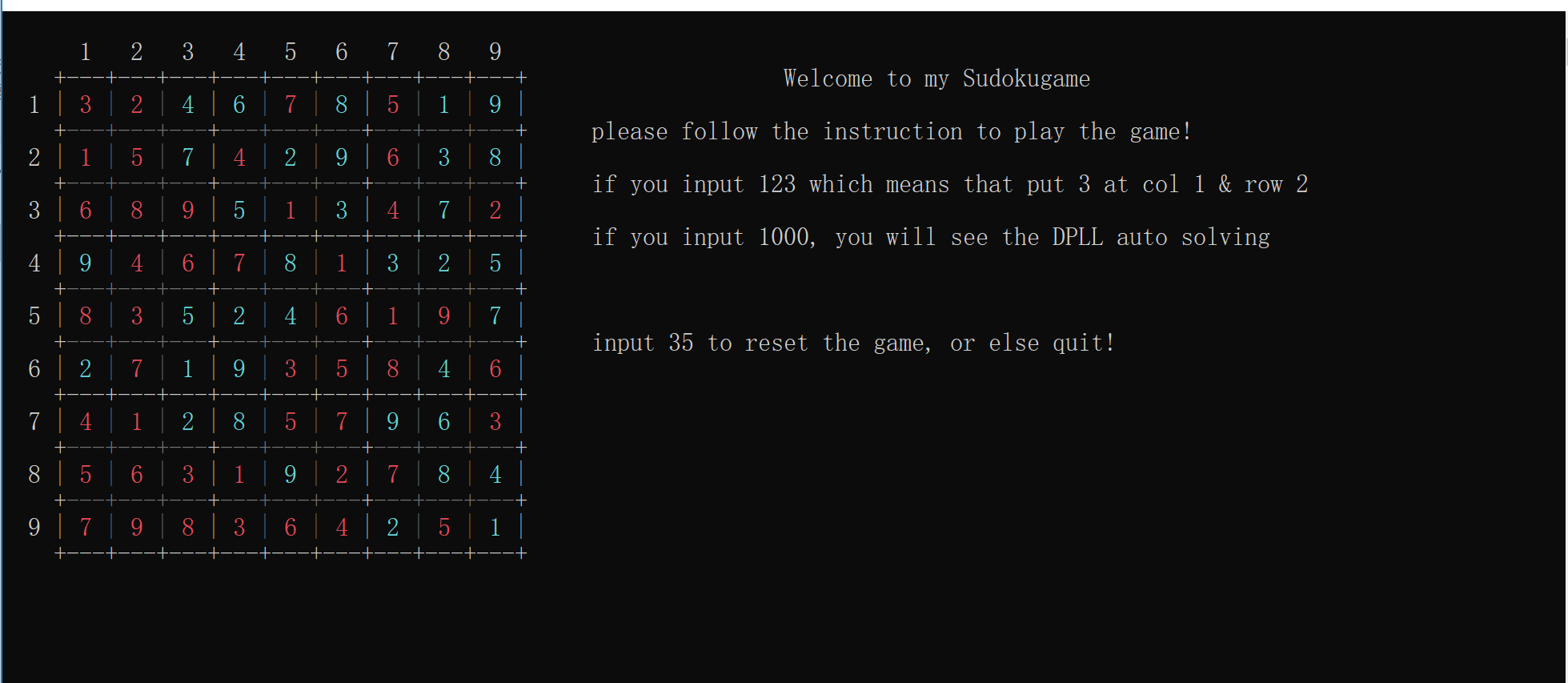


图4-21 自动求解结果示意图

* + - **完成了之后你会看到重启游戏的提示，输入35-55便可重启游戏（最好输入35，因为35个初始变量游戏体验更好），输入其他数字便会推出程序。**

5 总结与展望

5.1 全文总结

**在这次课设中，我的主要工作如下：**

* + - 设计了合适的存储结构来解决SAT问题
    - 设计了一种比较优化较为巧妙的方法来解决DPLL
    - 设计了一个比较美观，交互性较强的数独游戏界面，游戏可玩性较强、
    - 设计了一个用户友好的主界面和数独游戏界面，给了充足的输入提示以及良好的输入感受。

5.2 工作展望

**在这次课程设计中依然有许多需要优化的地方，具体如下：**

* + - 我使用的DPLL是用递归实现的，而显然迭代的效率会更高，我也尝试过迭代的算法，很可惜花了很长时间没有调试成功，希望今后有时间能够有所突破。
    - 设计数独界面的时候，我才用的是输入的方法，也有的人做出来了点击输入的效果，我也尝试过去做这个效果，可惜并不是很理想，有时候会有奇怪的现象发生。

**通过此次课设的经验，希望在今后的编程学习中，我能更好的做到以下几点：**

* + - 更加熟练的掌握有关数据结构，图、树、链表、字符串等基本写法
    - 对于一些基本的字符串、图、树的算法能有更深的理解以及运用能力
    - 增强大工程的设计能力，因为以前都写的是不超过200行的小程序，所以这次在课设中还是有点吃力的，在大工程中最头疼的是寻找bug的能力，希望在今后能逐步提高

6 体会

这次课设是对我们C语言和数据结构能力的一次很好的实践，我们通过课程设计对于链表等结构有了更深的了解与实现能力，使得我们的数据结构的知识不是再浮于理论层面。同时这也是我们第一次做一个大工程，难免会有一些困难。从最开始的设计整个系统开始，便是一件很头疼的事情。但是比较麻烦的是当你编译通过之后仍然有许多你找不出来的bug，这个时候需要你单步调试，找出错误在哪里。通过这一次训练，我们对于大工程的理解，寻找bug的能力都有所提高，对于C语言和数据结构的理解也有所提高。

在先开始，最大的问题就是设计整个系统，在理清的问题的基本思路之后，开始对于完成问题的基本数据结构进行设计。在任务书的提示下，我建立了以子句链表链接文字链表的数据结构，从效率上看，这样做的效率很高。但是到后来进行DPLL的时候我发现，链表中删除过的节点便无法再恢复，而重新存一个链表空间开销很大，而且无法准确定位，需要一个个遍历来恢复。并且，因为不知道递归循环的次数，所以并不知道要复制多少个这样的链表，这样程序很可能会崩溃。所以经过思考我提出的方案是，设置一个开关state，我并不将节点真的从链表中删除，而是如果删除，将节点的state由true变为false，遍历时候仅仅只需要遍历true节点即可。然后我们再构建一个数组去存储删除节点的位置，这样恢复起来可以直接定位，我们直接找到位置，回复的时候将flase变为true。这样相当于我们只用多构建一个链表，而不是许多链表，并且也极大减少了链表遍历而带来的时间开销。按照新思路，我重新构建了结构。

而后遇到的主要问题就是一些隐性的bug。程序编译通过之后能跑过一些样例而有一些样例跑不过。面对这样一个工程，单步调试还是十分困难的，经过大约一周不断的尝试，终于发现了有几个会使得DPLL崩溃的小错误，这个调试的经理是很痛苦也是很宝贵的。

最后便是数独界面的设置，我想尽量做出一个美观的界面，于是便上网去搜了C语言如何在窗口定位，显示颜色等等内容，最后做出了一个交互效果和动态效果都还不错的界面。也有一些想实现的效果没有实现，比如用鼠标定位制定填入的位置等等这些效果，希望在今后能找到这些问题的更好方法。

综上，通过这次课设，我们对于C语言及其工程有了更好的认识，锻炼了我们的结构设计能力以及调试代码的能力。这次课设也让我们学习了一些新的东西，比如如何设计界面等等，也让我们接触到了更多C语言的库，对于我们是十分有好处的。

参考文献

[1] 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

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

[3] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360

[4] Wikipedia: Sudoku <https://en.wikipedia.org/wiki/Sudoku>

[5] 百度文库：基于“挖洞”思想的数独游戏生成算法

<https://wenku.baidu.com/view/200c448af705cc1754270957.html>

附录 程序使用手册

**基于SAT的数独游戏求解程序”用户使用手册**

**内容提要**

**本手册包含以下三个方面的内容：**

* **程序主页面（菜单页面）介绍**
* **求解cnf文件中的SAT问题介绍**
* **数独游戏介绍**

**其中，程序主页面是用户编译程序之后看到的页面。用户可以根据交互提示选择近视求解cnf程序或者进入数独游戏程序，详情请见后面对于三个模块的具体操作使用方法的介绍**

* 1. **程序主页面（菜单页面）介绍**
* **编译源文件main.cpp。（因为设计效果的问题，有些地方需要用c++语法，除了个别地方，其他均为c语言语法）**
* **你会看到如下界面**

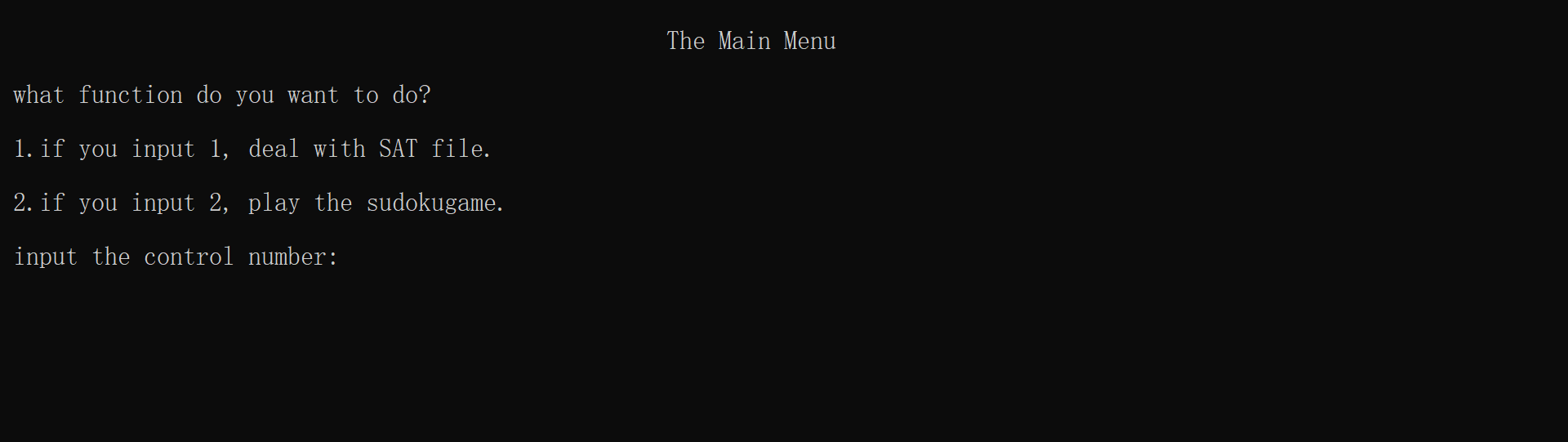


图1-1 主菜单示意图

**你需要做的是输入1或者2。**

* 其中输入1代表你将要用DPLL算法求解SAT问题（求解cnf文件中的SAT问题）
* 输入2代表你要玩数独游戏
* **比如输入1，你会看到下左图，回车后清屏，进入SAT程序，如下右图**

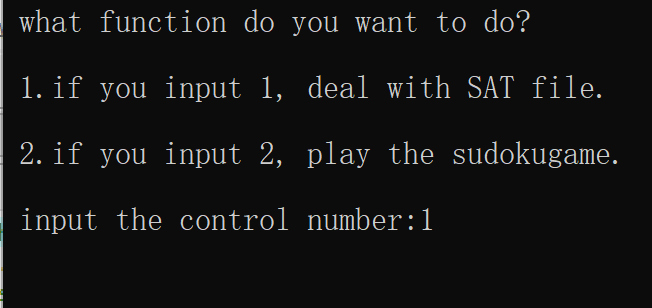
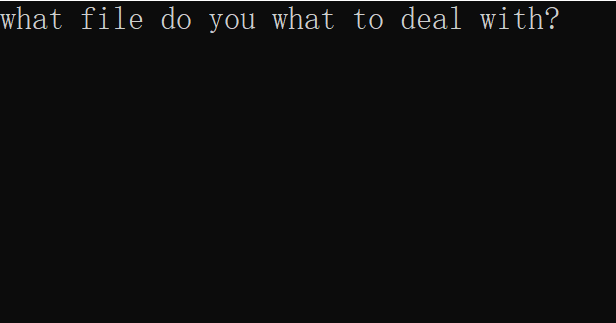
 

图1-2 输入1后界面示意图

* **输入2，回车后清屏，进入数独游戏**

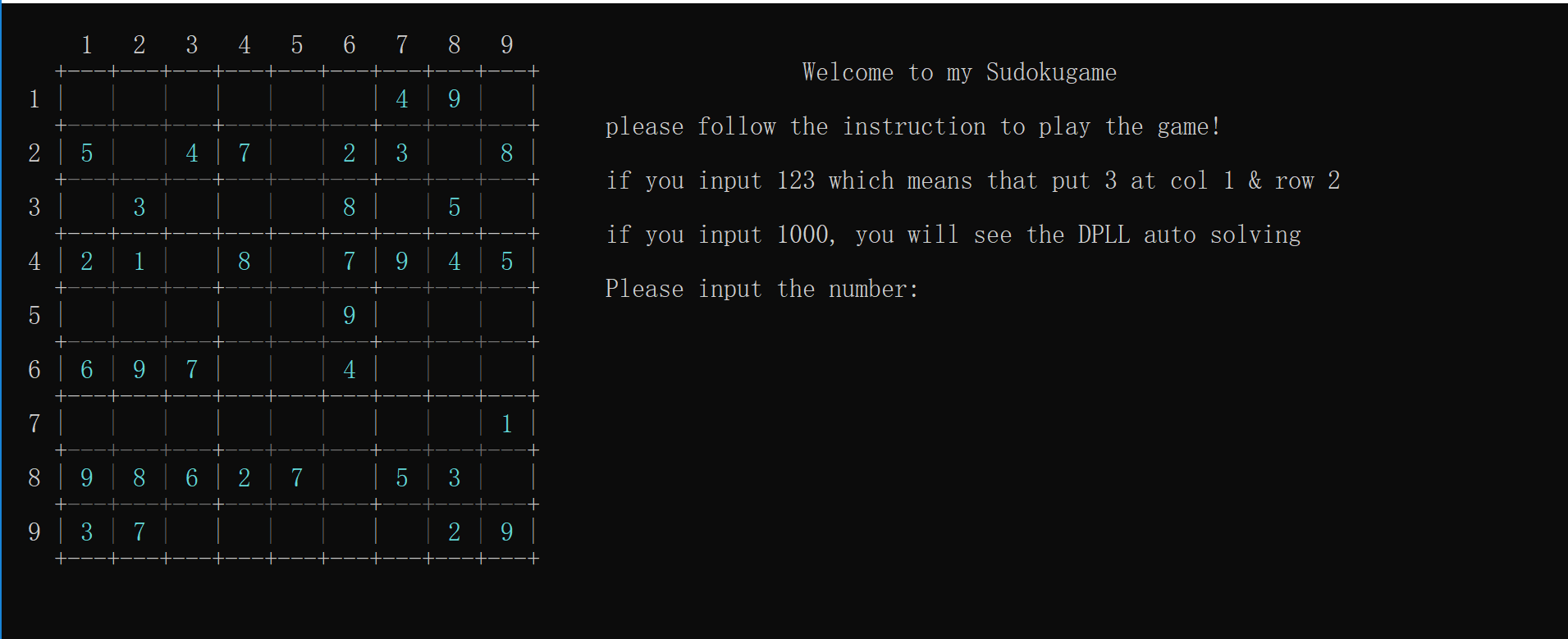


图1-3 输入2后界面示意图

* **以上为程序菜单的介绍**

**二、求解cnf文件中的SAT问题介绍**

* **首先，在主菜单中输入1，上一节已经介绍**
* **然后在新界面输入你要处理的文件的绝对位置，如下图所示**

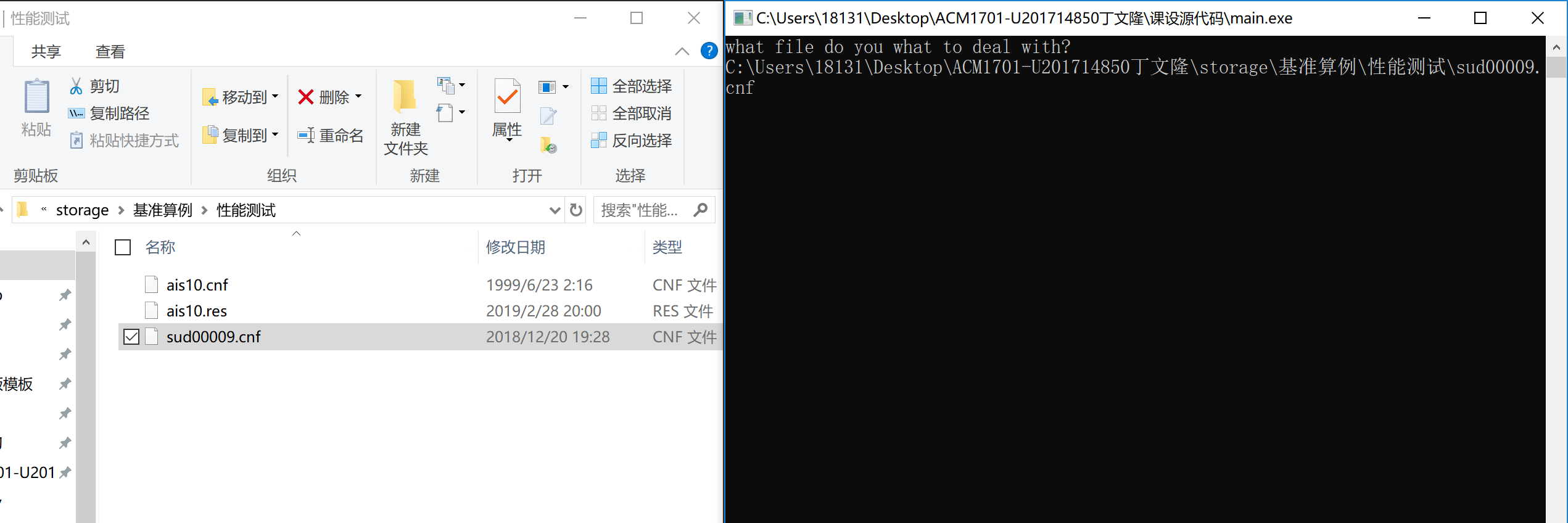


图2-1 输入处理文件的绝对位置示意图

* **点击回车，会发现屏幕上出现输出的链表，链表输出完成后，会显示这是否是一个满足的算例，如下图所示**

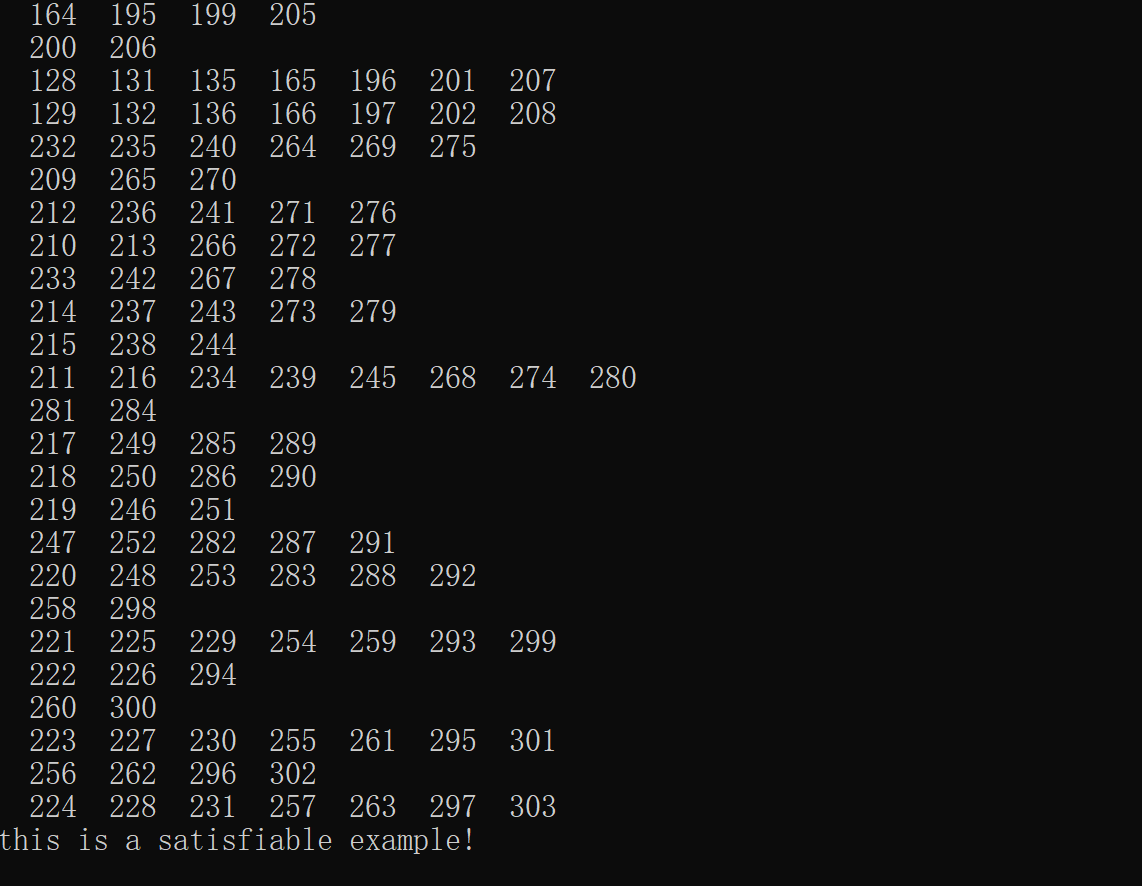


图2-2 输出链表以及结果示意图

由输出结果可知，这是一个满足算例

* **结束程序之后，你会发现res文件出现在了同一目录下，点开可查看输出信息，如下图所示：**

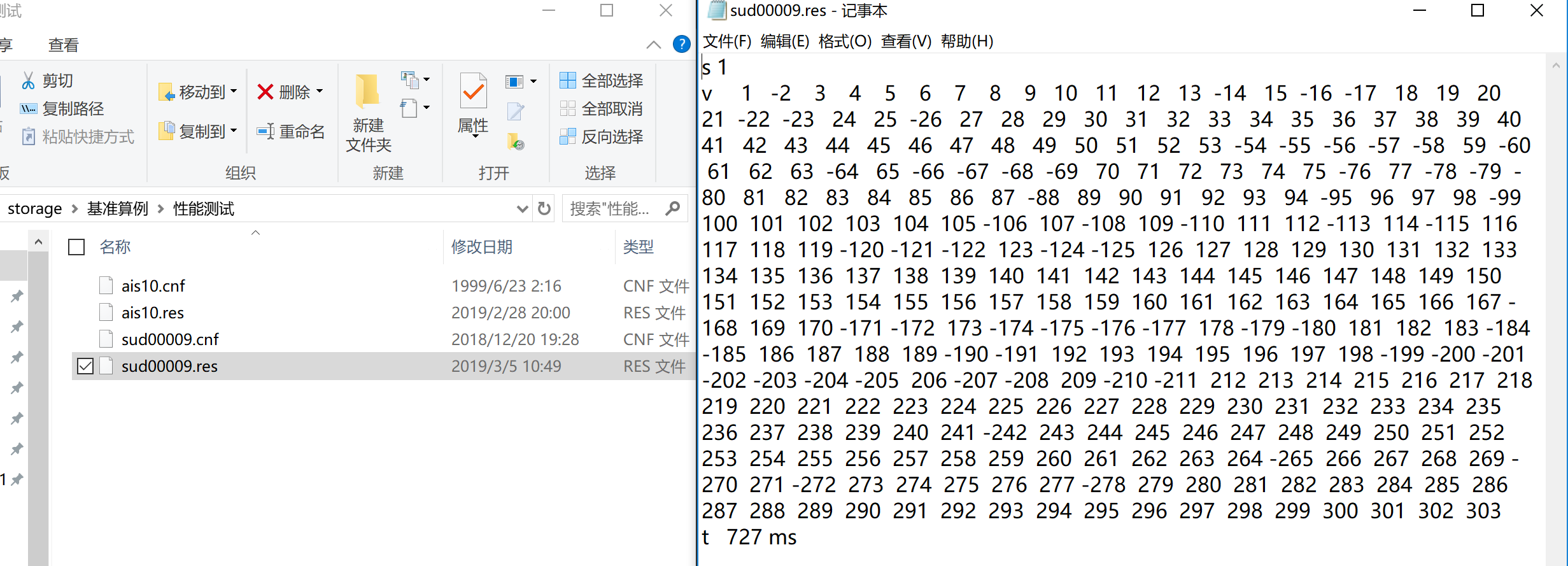


图2-3 对应res文件内容示意图

* 第一行的s代表是否是一个满足算例，若是则为1，不是则为0。
* v后面跟的一串数字代表每个文字取的布尔真值，是正取真，是负取假。
* t代表其中DPLL算法运行的时间。（注意并不是整个程序运行的时间，因为输出链表花费时间很长。）
* **以上是关于处理cnf文件中SAT问题的程序的操作介绍**

**三、数独游戏介绍**

* **在主菜单中输入2并回车，如第一节所示。**
* **你将会看到如下界面：**

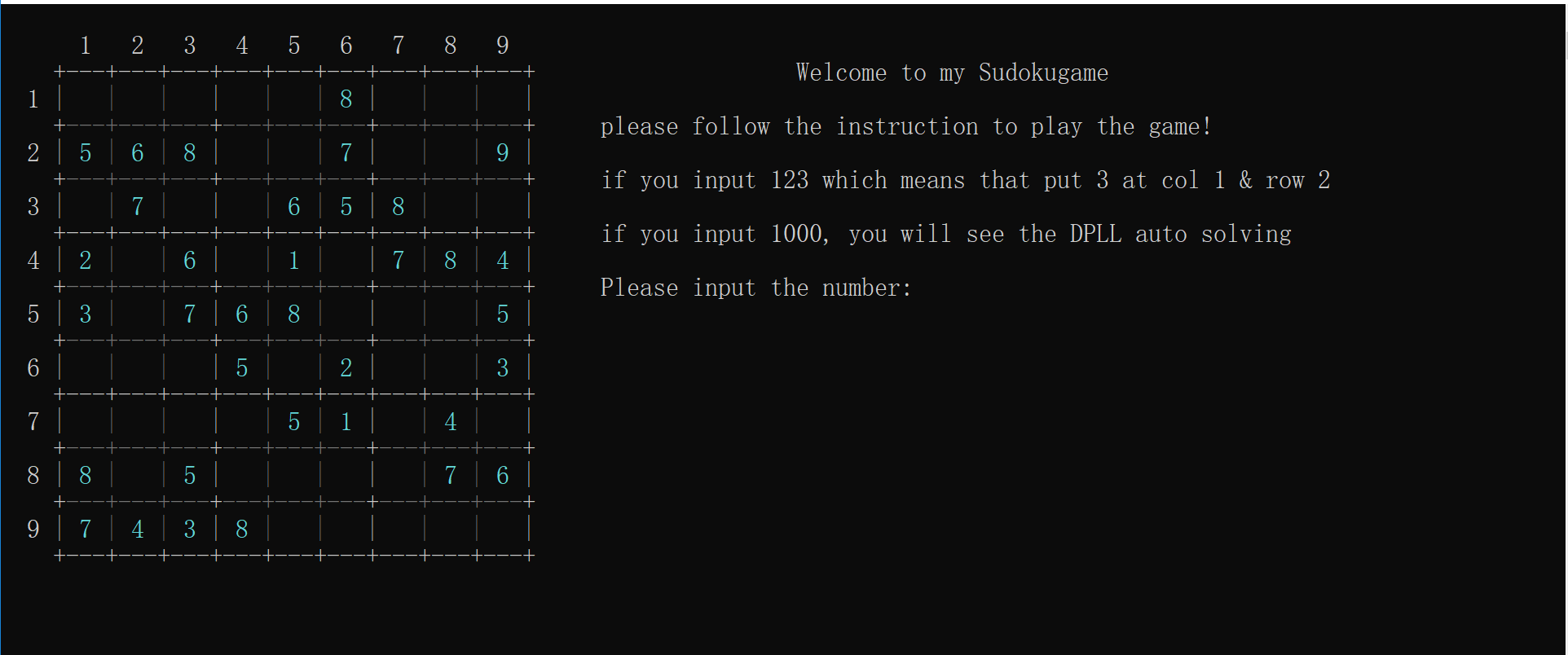


图3-1 数独游戏界面示意图

* **输出的cnf文件路径需要玩家自己设置，在main.cpp中改源代码，输入你想输出cnf文件的位置。**（PS：若玩家自己不更改则无法看到cnf文件）（PS：C语言语法要求，输入路径要双斜杠）下图仅为参考所用：

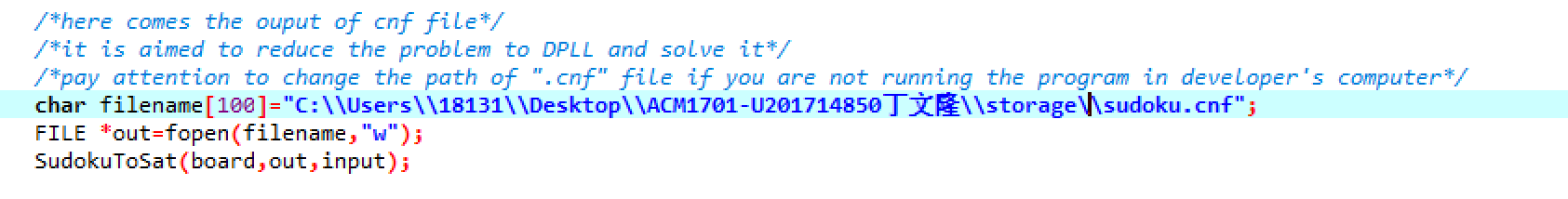


图3-2 修改cnf文件位置示意图

* **你可以看到输出的cnf文件如下：**

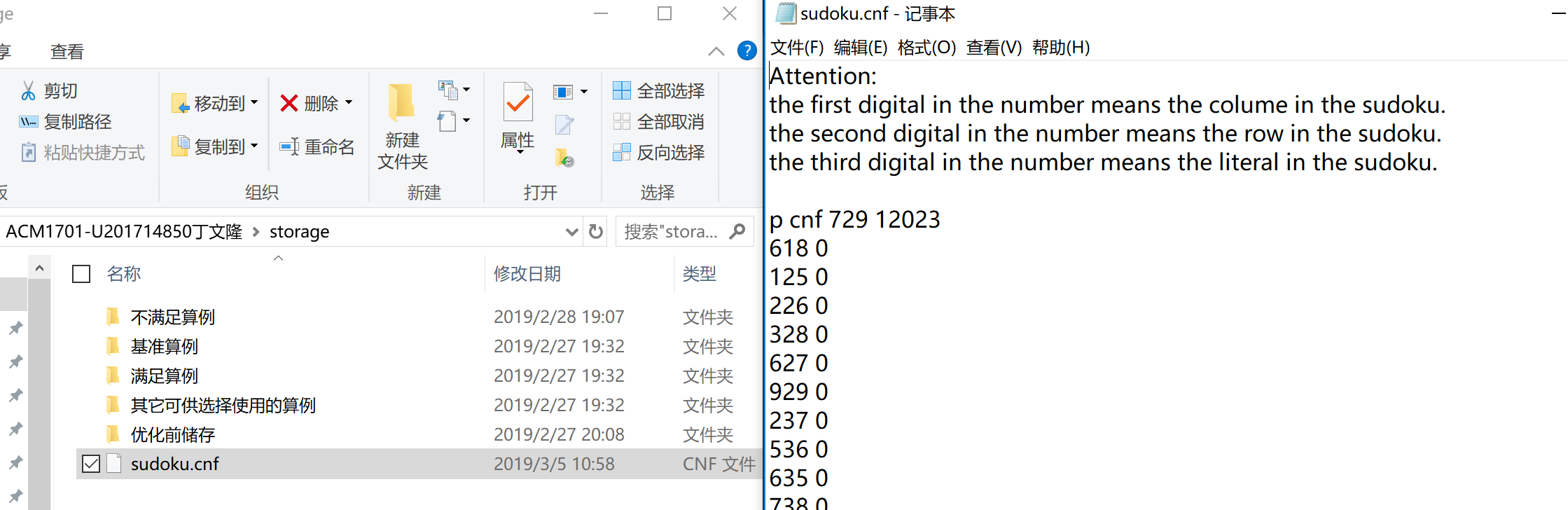


图3-3 输出的cnf文件示意图

* 回到数独游戏的玩法，正如提示所示，**输入一个三位数，如果想在第一行第2列输入9，则输入219（\*注意第一个数字是列，第二个数字是行）效果如下图所示：**

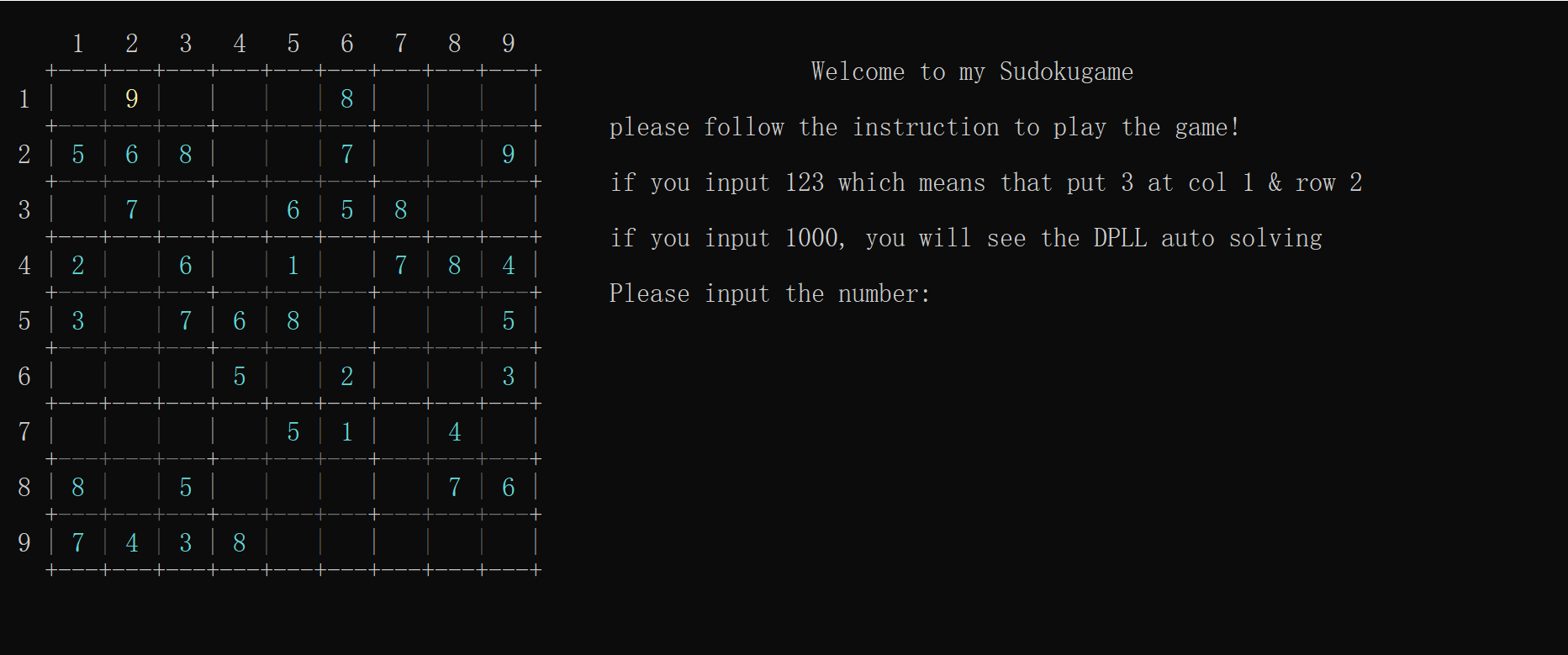


图3-4 用户输入正确结果效果图

你会发现黄色的9填入了第一行第二列，填完之后填数字的位子自动清空方便下一次填写。

* **如果你填入的数字不符合数独规则（即填入位置同行同列同块有相同数字），程序会自动提示你是错误的，如下图：**

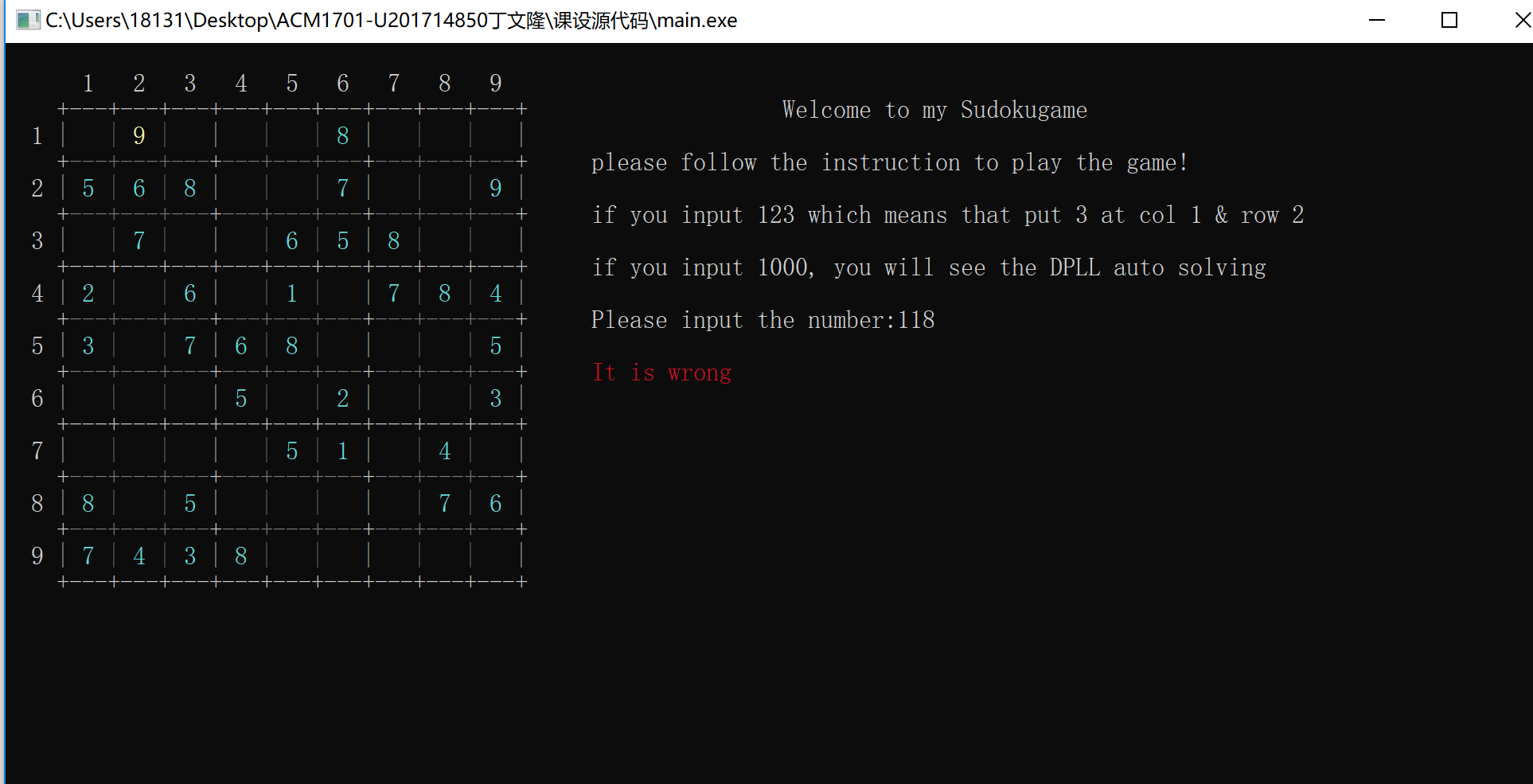


图3-5 用户输入不符合数独规则报错

作短暂停留之后又会清空方便下次填写。

* **如果你填入的位置已经有数字了，程序也会提醒你，如下图：**

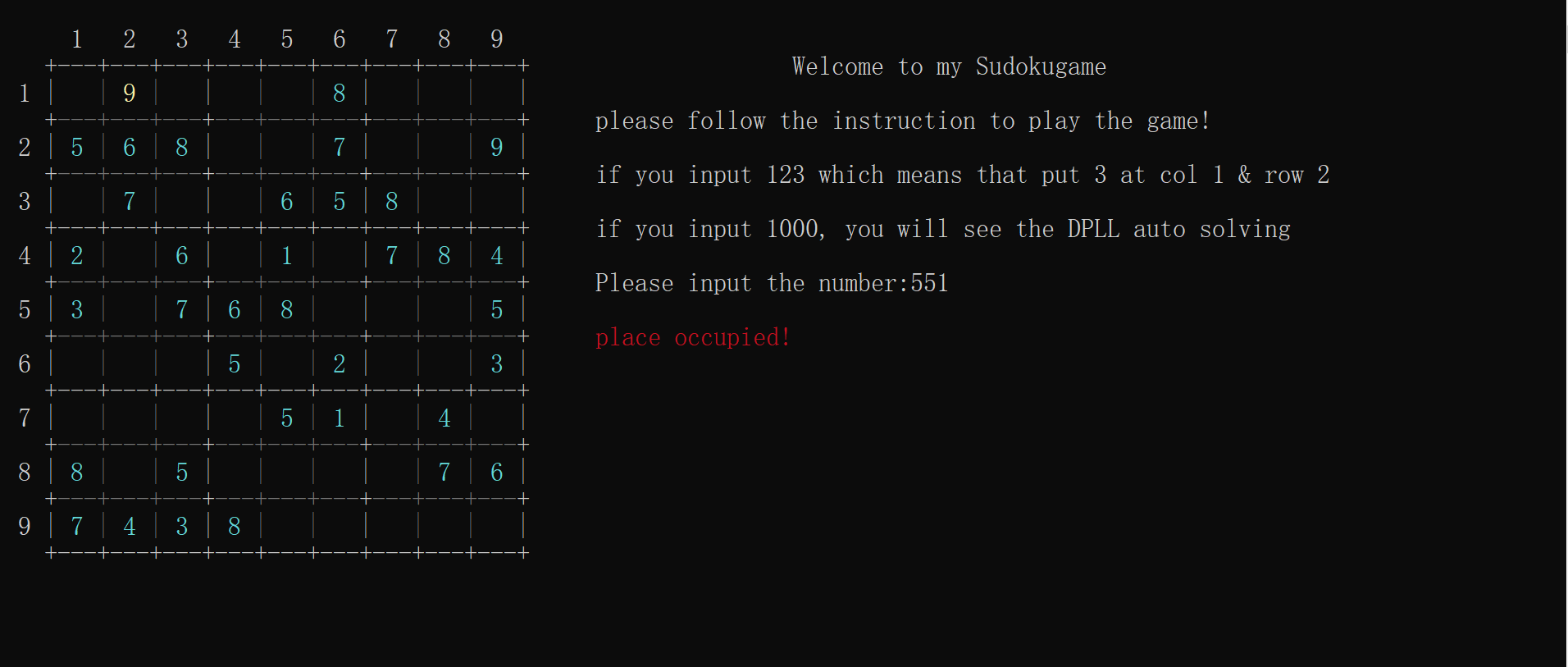


图3-6 用户输入位置被占用报错

作短暂停留之后又会清空方便下次填写。

* **如果想查看DPLL算法自动求解，只需要输入1000即可，程序会动态得显示答案（一个接一个地出现）求解完成截图如下：**

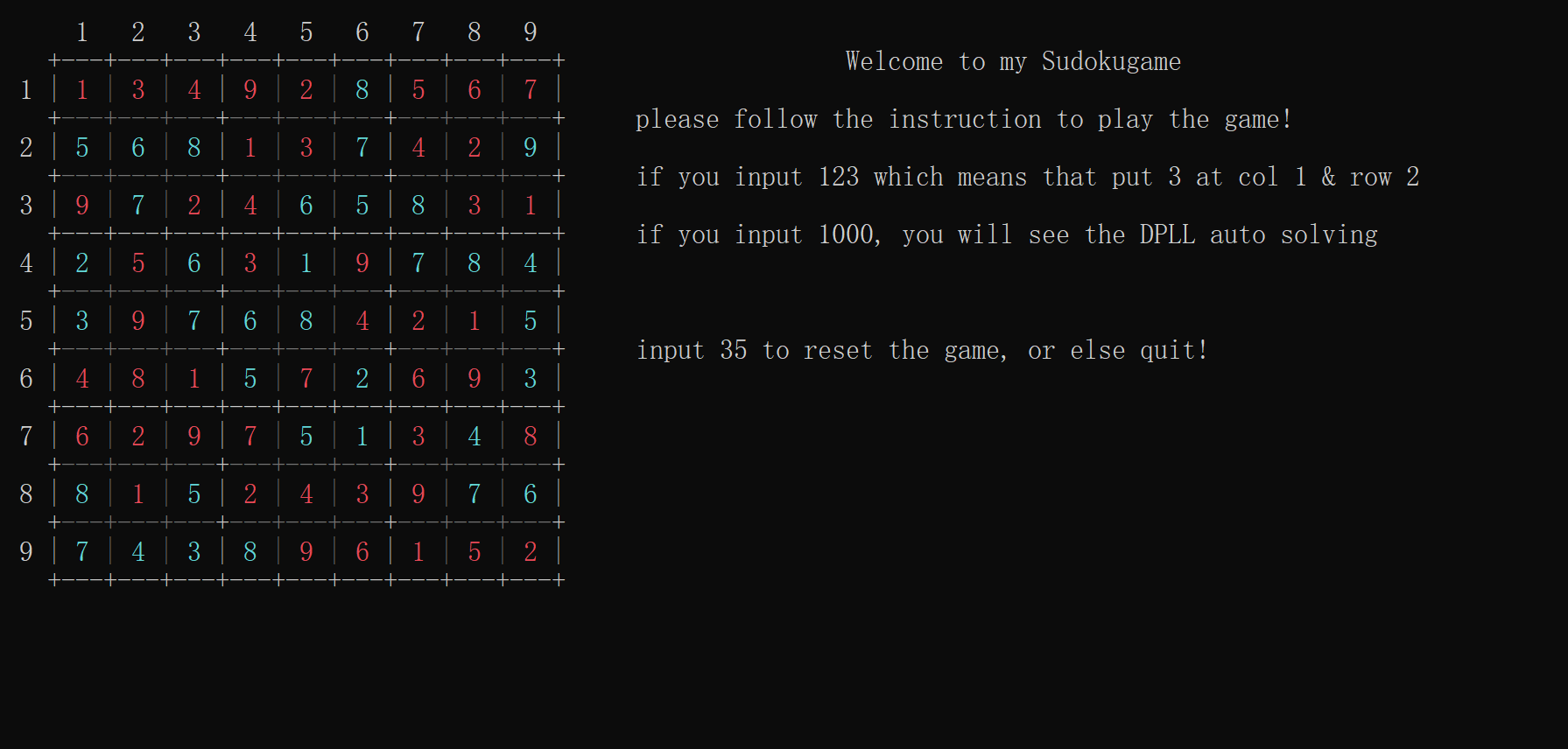


图3-7 DPLL自动求解效果示意图

* **如果想重新玩下一盘游戏，输入35（实际上输入35-55都是可行的，该输入设置的是初始填充的个数，只不过35个游戏体验最佳），若输入其他数字，则退出程序。如下图所示：**

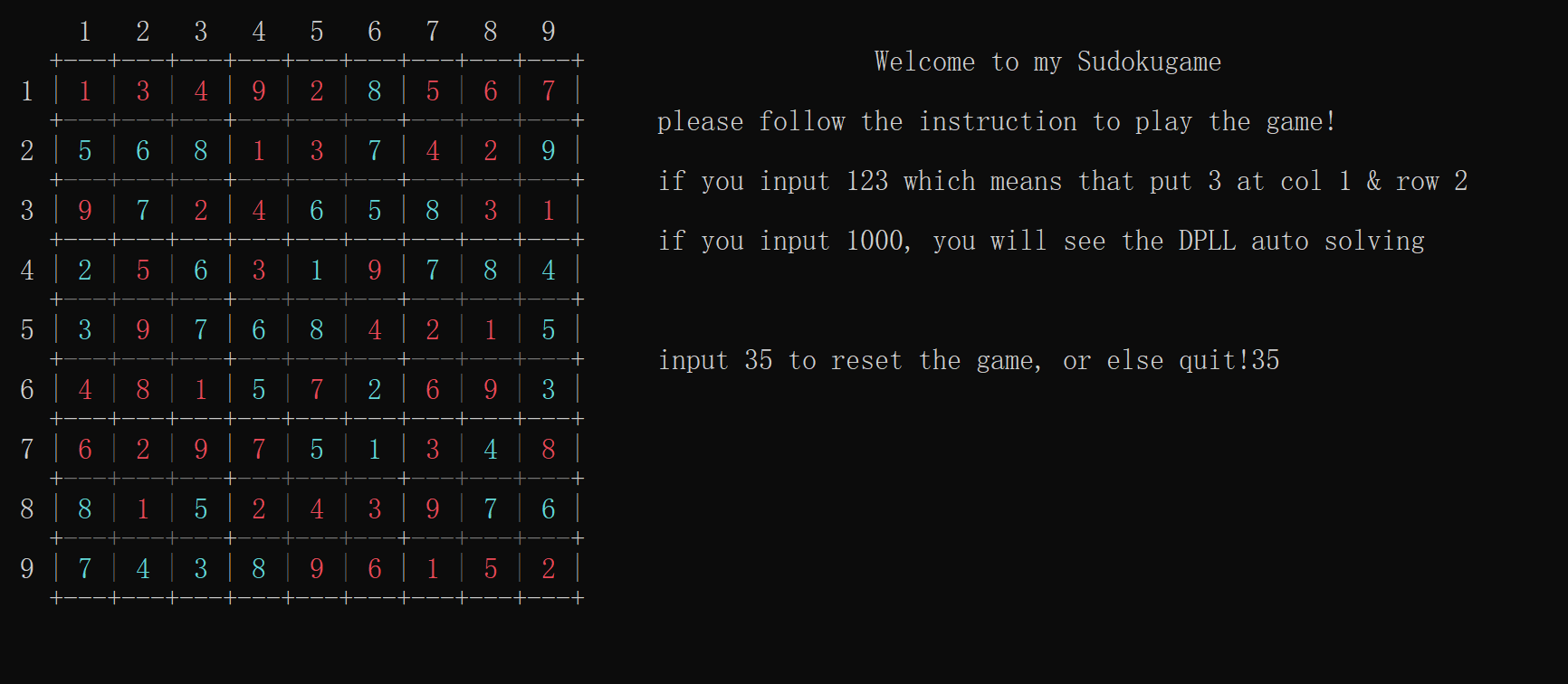


图3-8 输入重启游戏的数字

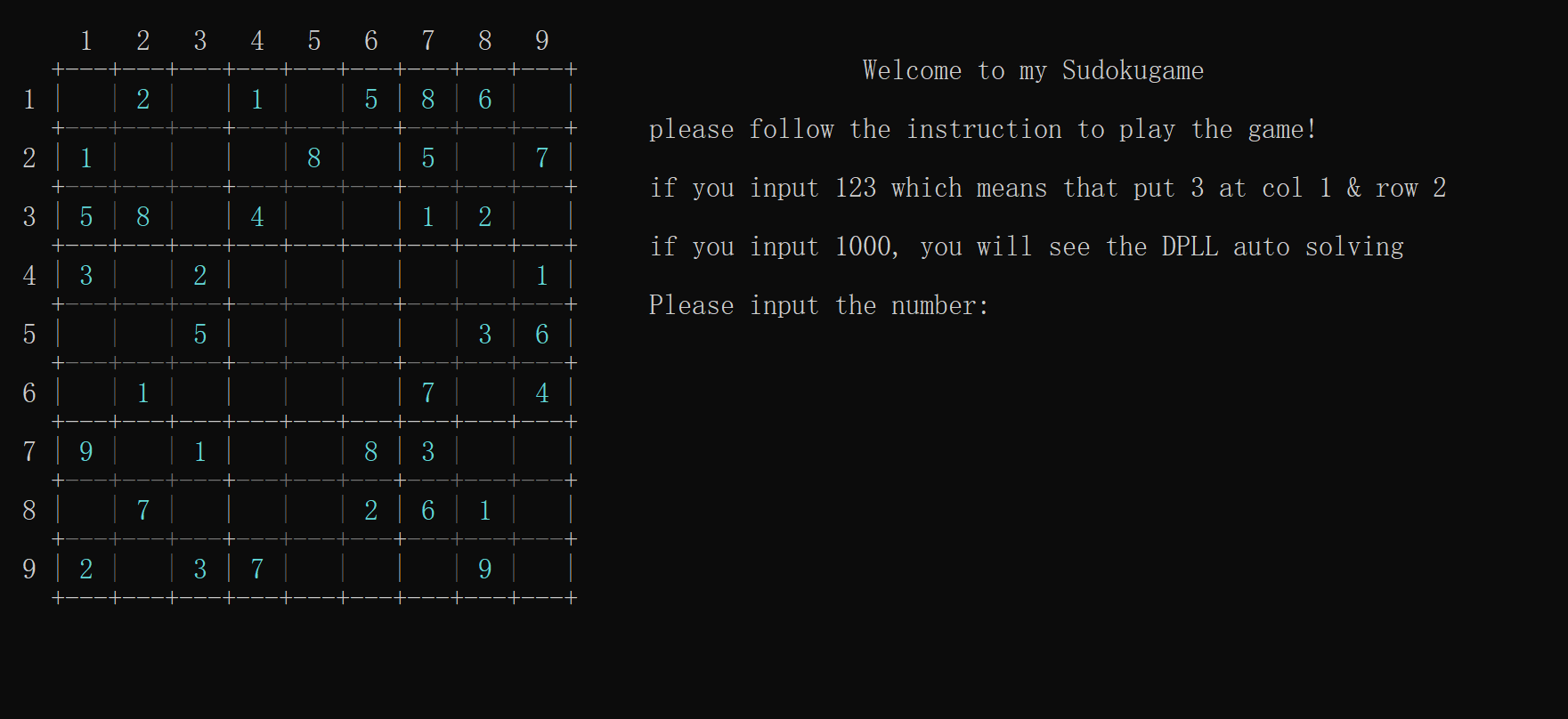


图3-9 重启后游戏界面

* **以上就是关于数独游戏的介绍。**