

**课程设计报告**

**题目：基于SAT的二进制数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： CS1807**

**学 号： U201814685**

**姓 名： 张嘉洋**

**指导教师： 李剑军**

**报告日期： 2020.03.20**

计算机科学与技术学院

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 2](#_Toc35617781)

[一、 引言 5](#_Toc35617782)

[1.1 课题背景与意义 5](#_Toc35617783)

[1.2 问题简介 5](#_Toc35617784)

[1.3 国内外研究现状 10](#_Toc35617785)

[1.4 课程设计的主要研究工作 10](#_Toc35617786)

[二、系统需求分析与总体设计 12](#_Toc35617787)

[2.1 系统需求分析 12](#_Toc35617788)

[2.2 系统总体设计 14](#_Toc35617789)

[三、系统详细设计 16](#_Toc35617790)

[3.1 数据结构的定义 16](#_Toc35617791)

[3.2 主要算法设计 17](#_Toc35617792)

[四、系统实现与测试 23](#_Toc35617793)

[4.1 系统实现 23](#_Toc35617794)

[4.2 系统测试 25](#_Toc35617795)

[五、总结与展望 35](#_Toc35617796)

[5.1工作总结 35](#_Toc35617797)

[5.2工作展望 35](#_Toc35617798)

[六、心得体会 37](#_Toc35617799)

[参考文献 39](#_Toc35617800)

[附录 程序使用手册 40](#_Toc35617801)

# 一、 引言

## 1.1 课题背景与意义

经过C语言和数据结构的学习，我们已经拥有了基础的的运用高级语言设计和编写程序水平，并且具备建立结构解决复杂问题的能力。该课题“基于SAT的二进制数独游戏求解程序”是一个大型复杂的问题，要求建立合适的数据结构，基于该结构实现DPLL算法，从而求解SAT问题。此外要求在解决SAT问题的基础上，将其建立二进制数独游戏求解程序，具有较高的综合性，考察了对数据结构的认识水平与运用能力，以及程序和算法实现能力，着重考察了将理论问题转化为实际问题并加以解决的能力。该课程设计是对C语言和数据结构课程的实践与检验，也是对计算机编程思维能力的一次综合考验。

## 1.2 问题简介

下面是对SAT问题和二进制数独问题简单的介绍：

* **SAT问题概述**

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。具体定义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，即称该问题是可以满足的，即SAT，此时输出给布尔变元赋值的结果。

例如有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，则称这个公式是可满足（satisfied）的，一组解是a=0，b=0，c=1。

* **DPLL算法相关概念介绍**
* **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。

* **DPLL算法综述**

DPLL算法是解决SAT问题的一个经典算法，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。



**图1.1 DPLL算法搜索树**

* **二进制数独游戏综述**

与普通数独游戏（Sudoku）相似，n阶（偶数n=2m≧4）二进制数独游戏（Binary Puzzle）要求在n×n的网格中每个单元（cell）填入一个数字1或0，必须满足约束：

* 在每一行、每一列中不允许有连续的3个1或3个0出现；
* 在每一行、每一列中1与0的个数相同；
* 不存在重复的行与重复的列。

一个二进制数独游戏初始时已经提供了一些提示数，如图1.2左图，要求在剩下的空格中填满数字，图1.2右图给出了其解，即完整填充。

****

**图1.2 一个Binary Puzzle格局与其解**

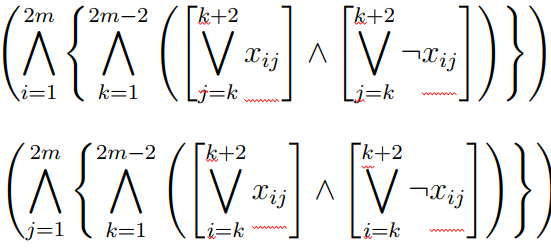
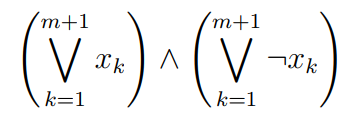
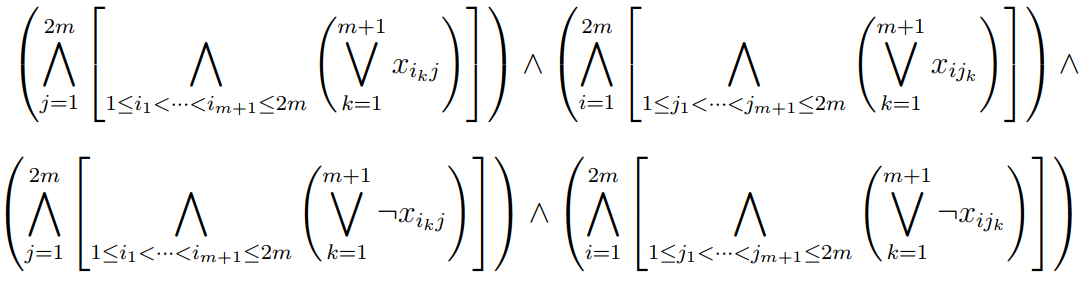
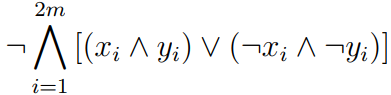
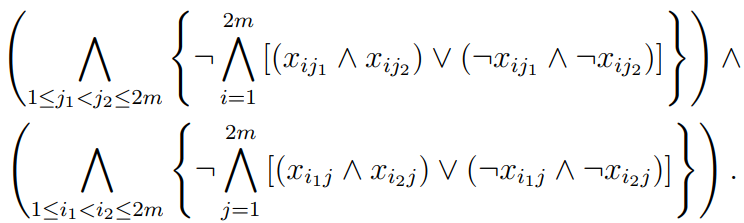
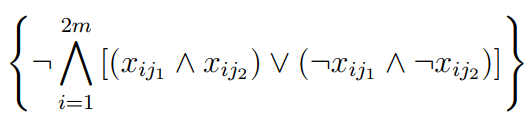
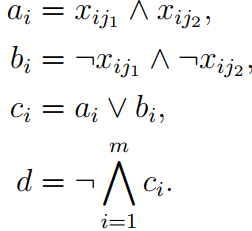
* **二进制数独归约为SAT**

二进制数独归约为SAT问题的基本思路是将格局内的每一个方块都转化为布尔变量，再利用初始格局以及三个约束条件得到关于格局布尔变量的CNF文件，便可转化为一般的SAT问题利用DPLL算法求解。求解得到的布尔变量值若为真，相应的空格便填入1,；反之，则填入0。

假设在阶数为2m（m<5）的二进制数独游戏中，第i行j列的方格转化为布尔变量表示为i \* 10 + j。

因此，图1.2左图中的游戏可生成如下单子句集：

¬12 24 26 ¬28 ¬33 42 55 ¬62 66 ¬72 ¬75 ¬85 ¬87。

* 约束一：  
  要保证任意行任意列相邻的三个变量不相同，即需满足表达式：( x1∨x2∨x3)∧( ¬x1∨¬x2∨¬x3)为真。  
  那么在2m阶的二进制数独游戏中，我们一共有2m\*2m个变量，将产生字句的表达式为：  
    
    
  即将产生8m\*(2m-2)个字句，每个字句含有3个文字。
* 约束二：  
  要保证在每一行、每一列中1与0的个数相同，则需保证在每一行的所有变量即2m个变量中，选取m+1个变量，不全相同。即满足：  
  。  
  那么在任意行任意列中，一共将产生的字句表达式为：  
    
  即共有个字句，每个字句中含有m+1个文字。
* 约束三：  
  任意行任意列都不完全相同，即对于某两行或某两列有如下表达式：  
    
  用二进制数独中的布尔变量来表示，则全部可能性表示为：  
    
  为了使用Tseytin 变换，将上述表达式转化为CNF范式，以该式的第一部分为例，做出如下操作：首先定义如下Tseytin变换变量：  
  。  
  我们便可以将约束三中关于二进制数独格局中的布尔变量相互联系在一起，为了符合CNF范式，可将上述Tseytin变换变量进一步变换为附加布尔变元以便参与DPLL算法运算。变换规则可参照下面针对8阶二进制数独附加变元的举例：  
   15711= 51∧71；15710= ¬51∧¬71；1571= 15711∨15710；

15721= 52∧72；15720= ¬52∧¬72；1572= 15721∨15720；

…

15781= 58∧78；15780= ¬58∧¬78；1578= 15781∨15780；

157= ¬[1571∧1572∧…∧1578].

其中，最高位数字1为行标志（2则表示列）；次高位5及之后的一位7表示对应的第5行与第7行；第4位数字1，2，…，8分别表示行中的第1个单元，第2个单元，…，第8个单元；第5位取1或0，含义自明。  
最终我们可以得到个字句。

将三个约束的所有具体要求分别转换成CNF子句集，连同预填提示数对应的单子句，便得到二进制数独游戏所生成的完整CNF公式。

当利用DPLL算法求得对应CNF公式的解后，需通过式i \* 10 + j对应的逆变换对解的含义进行解析，获得游戏的填充方案。

## 1.3 国内外研究现状

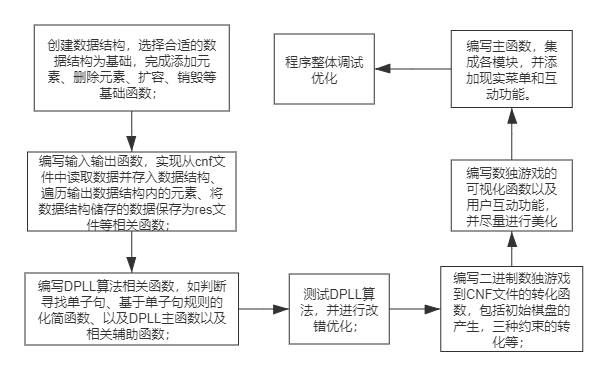
命题逻辑公式的可满足性问题(SAT)是数理逻辑、计算机科学、集成电路设计与验证和人工智能等领域中的核心问题,并且是第一个被证明出来的NP问题。SAT问题在计算复杂性理论中具有非常重要的地位,设计并实现解决该类问题的高效算法意义重大。但目前不存在一种求解算法在最坏情况下的时间复杂度是多项式级别,其求解速度仍是制约SAT算法发展的一大难题。因此,世界各国的学者都在努力研究新的求解算法,以寻求出一种高效的求解算法。

## 1.4 课程设计的主要研究工作

本设计要求设计建立有效的物理存储结构，基于DPLL过程实现一个高效SAT求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。功能具体要求如下：

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。
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。
3. **DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。
4. **时间性能的测量：**基于相应的时间处理函数，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. **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。
6. **SAT应用：**将二进制数独游戏问题转化为SAT问题，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。

主要工作的详细流程图如图1.3所示：



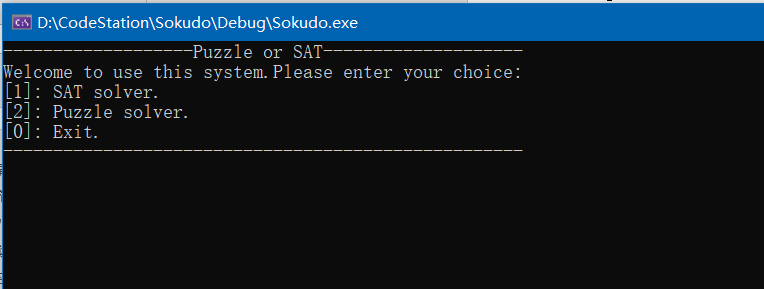
**图1.3 主要工作流程图**

二、系统需求分析与总体设计

## 2.1 系统需求分析

本系统实现了对CNF文件的解析以及在此基础上对SAT问题进行了DPLL算法求解。用户进入系统后可以直接选择进行SAT问题的求解，也可进去本系统的拓展程序——二进制数独游戏的解答。用户在输出游戏阶数后，系统会自动生成初始格局，用户可选择自动求解或手动求解。本系统与用户进行交互的各个界面都进行了一定的美化并可确保较好的互动效果。系统的具体运行流程如下：

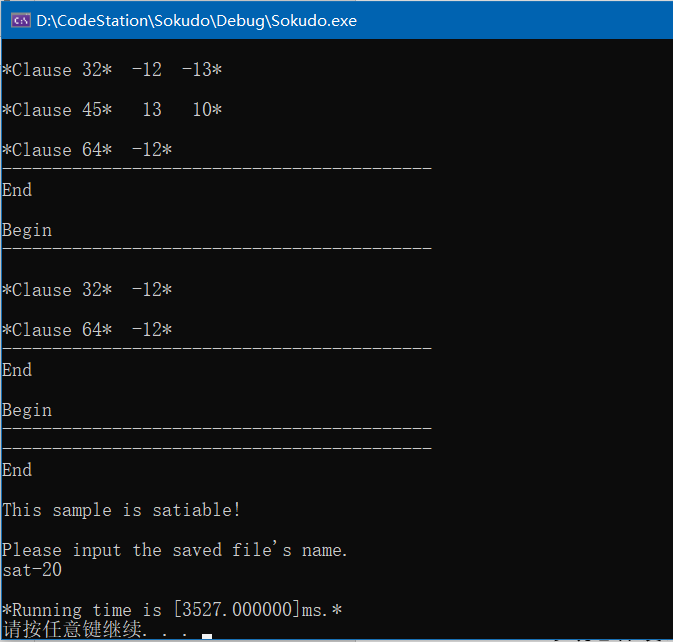
运行程序进入主菜单：



**图2.1 主菜单显示页**

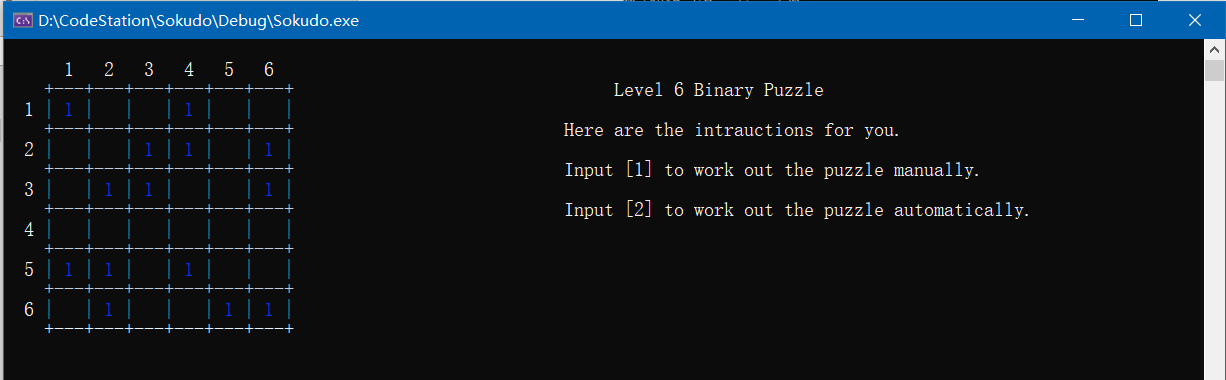
根据用户输入进入不同程序。

* SAT求解器  
  运行并保存为文件：

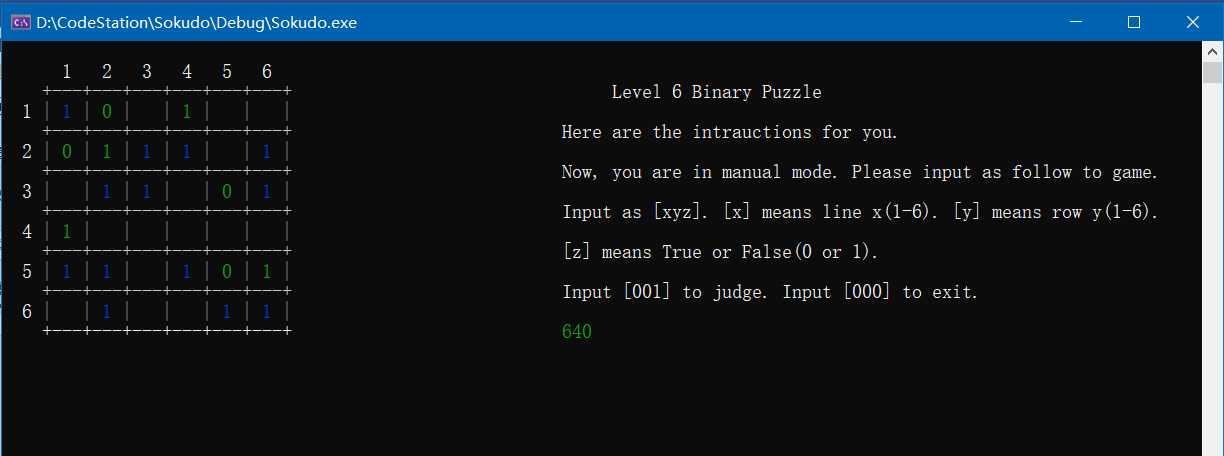


**图2.2 SAT求解结果**

* 二进制数独游戏求解  
  进入二进制数独游戏界面：

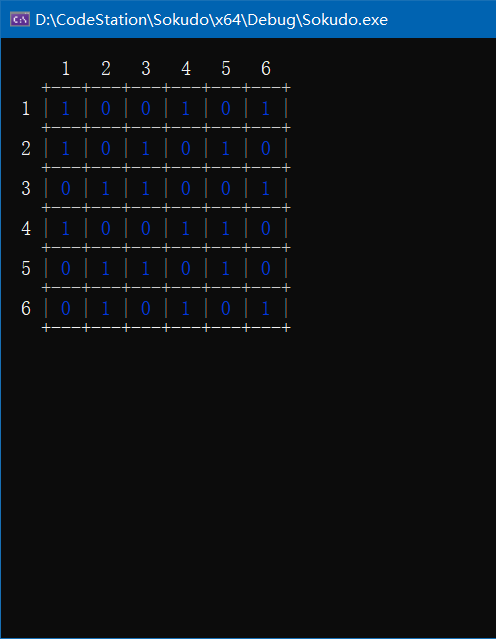


**图2.3 二进制数独游戏界面**



**图2.4 手动游戏中界面**

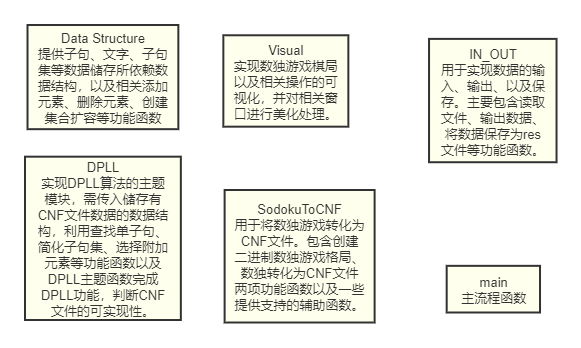
求解结果：



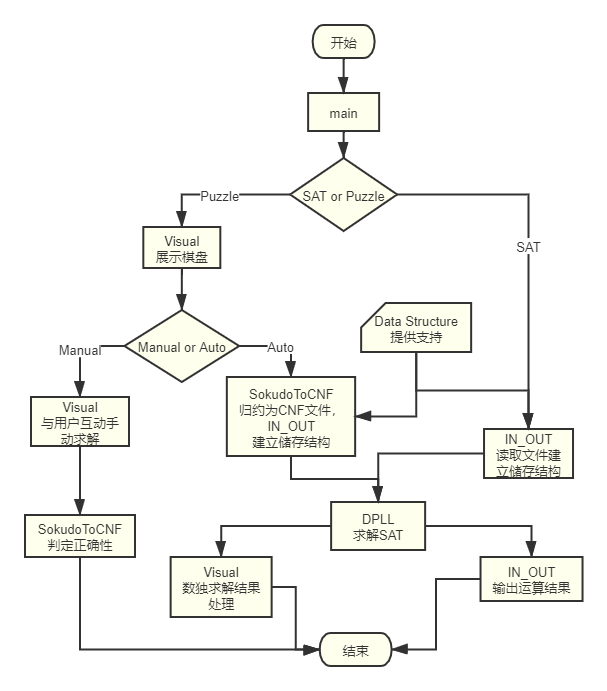
**图2.5 求解结果**

2.2 系统总体设计

本系统分为六大模块，分别为Data Structure，DPLL，IN\_OUT， SokudoToCNF， Visual， 以及main。六大模块的功能及其联系如图2.6，图2.7所示：



**图2.6 六大模块及其功能**



**图2.7 六大模块的联系及系统流程图**

# 三、系统详细设计

## 3.1 数据结构的定义

从CNF文件中读取数据，建立顺序表，储存结构变量Clause，用数组储存各Clause包含的结构变量Literal。Clause->sub为指向该子句（Clause）所有文字（Literal）数组的指针。结构示意图如图3.1：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ClauseSet |  | LiteralSet | | |  | Clause | | |
| Clause0 |  | Variety1 | Variety2 | … |  | Length | State | Sub |
| Clause1 |  | Literal1 | Literal2 | … |  |  | | |
| Clause2 |  | Literal1 | Literal2 | … |  | Literal(Variety) | | |
| Clause3 |  | Literal1 | Literal2 | … |  | Location | State | Value |
| …  …  … |  | … | … | … |  |  |  |  |
| Clausen |  | Literal1 | Literal2 | … |  |  |  |  |

**图3.1 数据结构示意图**

在该示意图中，ClauseSet以及LiteralSet仅作为标注，数据结构中的顺序表均从之后的一行开始。在ClauseSet中，第一个元素以及最后一个元素不用于储存从CNF文件中读取的字句信息。Clause0用于储存数据结构的总体信息，其Length为子句总数目n的转化式，为2000 \* n + m，m为当前可用的字句数；Sub指向储存有所有布尔变元的数组。Clausen 作为预留位，用以在DPLL算法中将选择的元素作为单子句添加至该位置。

**表3.1 数据结构类型**

|  |  |  |
| --- | --- | --- |
| 结构名 | 变量名 | 数据意义 |
| Clause  （1~n-1） | Sub | 指向该子句所包含的文字集的指针 |
| Length | 储存该文字集的文字数目的转化式200 \* n + m，n为文字总数，m为当前可用数。 |
| State | 标注该字句的状态，0表示已被删除，1表示可用 |
| Location | 该子句在子句集中的位置 |
| Literal | Location | 标注该文字在文字集中的位置 |
| State | 标注该文字的状态，0为负文字，1为正文字 |
| Value | 该文字代表的布尔元素值 |

在Data Structure模块，即Data Structure.h 和Data Structure.cpp文件中，所建立的函数介绍如下：

**表3.2 数据结构相关函数**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 输入 | 输出 | 功能 |
| ClauseResize | Clause\* 型变量base  子句集的基地址 | Clause\*型变量  base | 对子句集进行扩容 |
| int型变量num  扩容后的元素总数目 |
| LiteralResize | Literal\* 型变量base  文字集的基地址 | Literal\*型变量  base | 对文字集进行扩容 |
| int 型变量num  扩容后的元素总数目 |
| ClauseSet\_init | Void | Clause\*型变量  base | 建立一个空子句集 |
| LiteralSet\_init | Void | Literal\*型变量  base | 建立一个空文字集 |
| VarietySet\_init | Void | Literal\*型变量  base | 建立一个空元素集 |

3.2 主要算法设计

* **DPLL算法**
* 算法概述

1. 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。
2. 若是其他情况，循环执行上一步过程，直到没有单子句为止
3. 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)。
4. 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。
   * 函数实现

实现DPLL算法的相关函数介绍如下表3.3：

**表3.3 DPLL算法相关函数**

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 输入 | 输出 | 功能 |
| AddToTail | Clause\* 型变量ClauseBase  子句集的基地址 | int型 | 将被选出的元素转化为单字句添加到子句集最后，并将该元素在元素集中的状态改变。 |
| int型变量v  被选出元素的值 |
| ChooseV | Literal\* 型变量Variety | int型 | 从元素几种选出一个未被删除的元素 |
| SimpClause | Clause\*型变量ClauseBase  子句集的基地址 | int型 | 根据单子句规则化简子句集 |
| int型变量v |
| SinClauseExist | Clause\*型变量  CLauseBase  子句集的基地址 | int型  num\_Clause(Exist.The location of single clause.) or FALSE(Didn't exist.). | 判断是否存在单子句。若存在返回单子句位置；否则，返回FALSE。 |
| Literal\*型变量Variety元素集基地址 |
| DPLL | Clause\*型变量  CLauseBase  子句集的基地址 | int 型变量 | DPLL算法主体部分，判断Clause储存的CNF文件是否可行 |

* + 其他问题

AddToTail函数利用了Data Structure部分创建数据结构时，在子句集中末位留出的空位，以方便调用。

DPLL算法采用递归方式进行，由于需要对正文字和负文字分别进行递归操作，在递归之前需要创建当前子句集的备份。解决方案是调用Data Structure中的CopyClauseSet函数创建一个副本并作为输入进行下一个调用。

* **二进制数独相关算法**
  + **数独格局生成**

数独格局的初始化采用文件读取方式进行。根据SokudoToCNF模块中的InitSokudo函数提示，用户需要进行输入，之后系统从相关库中读取初始格局，并调用Visual模块中的InitBoard函数完成棋盘格局的展示。

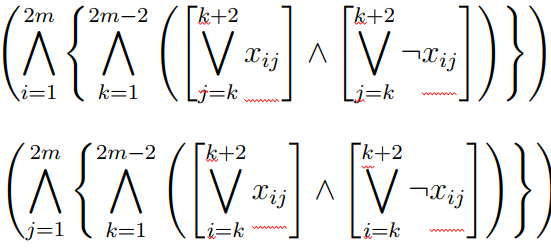
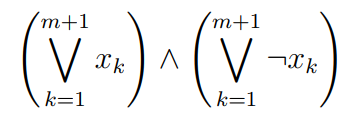
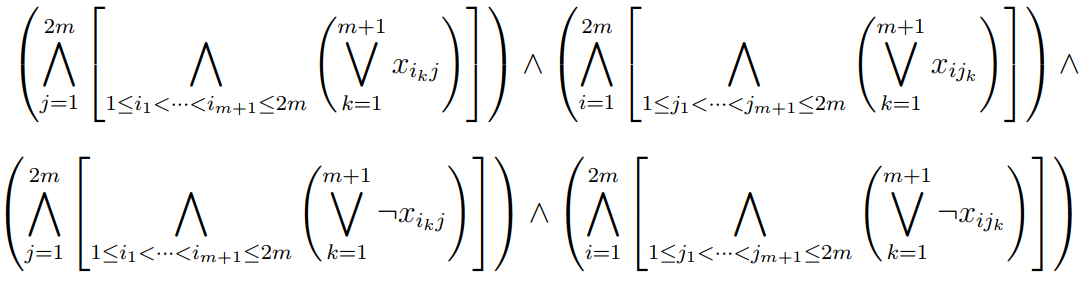
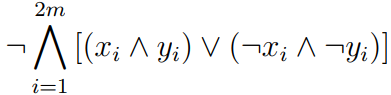
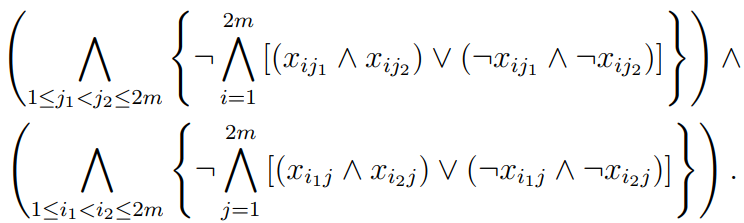
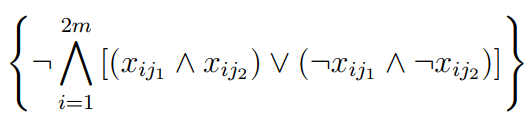
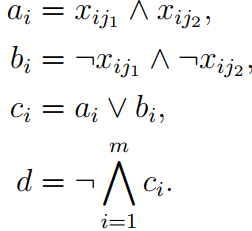
* + **用户手动求解**

若用户根据提示选择手动求解，则调用Visual模块中的ShowBoard\_manu函数，展示手动求解菜单，并根据用户输入，执行填充算法，逐步填充棋盘格局。填充完毕之后，可调用Visual中的Judge函数进行正确性判断。

* + **系统自动求解**

若用户选择了自动求解模式，则会调用SokudoToCNF模块中的Constraint函数，通过三个约束条件及初始棋盘创建CNF文件并进行DPLL算法的求解。求解成功后利用Visual模块中的ShowBoard\_auto函数进行填充。

三个约束条件详细介绍如下：

* 约束一：  
  要保证任意行任意列相邻的三个变量不相同，即需满足表达式：( x1∨x2∨x3)∧( ¬x1∨¬x2∨¬x3)为真。  
  那么在2m阶的二进制数独游戏中，我们一共有2m\*2m个变量，将产生字句的表达式为：  
    
    
  即将产生8m\*(2m-2)个字句，每个字句含有3个文字。
* 约束二：  
  要保证在每一行、每一列中1与0的个数相同，则需保证在每一行的所有变量即2m个变量中，选取m+1个变量，不全相同。即满足：  
  。  
  那么在任意行任意列中，一共将产生的字句表达式为：  
    
  即共有个字句，每个字句中含有m+1个文字。
* 约束三：  
  任意行任意列都不完全相同，即对于某两行或某两列有如下表达式：  
    
  用二进制数独中的布尔变量来表示，则全部可能性表示为：  
    
  为了使用Tseytin 变换，将上述表达式转化为CNF范式，以该式的第一部分为例，做出如下操作：首先定义如下Tseytin变换变量：  
  。  
  我们便可以将约束三中关于二进制数独格局中的布尔变量相互联系在一起，为了符合CNF范式，可将上述Tseytin变换变量进一步变换为附加布尔变元以便参与DPLL算法运算。变换规则可参照下面针对8阶二进制数独附加变元的举例：  
   15711= 51∧71；15710= ¬51∧¬71；1571= 15711∨15710；

15721= 52∧72；15720= ¬52∧¬72；1572= 15721∨15720；

…

15781= 58∧78；15780= ¬58∧¬78；1578= 15781∨15780；

157= ¬[1571∧1572∧…∧1578].

其中，最高位数字1为行标志（2则表示列）；次高位5及之后的一位7表示对应的第5行与第7行；第4位数字1，2，…，8分别表示行中的第1个单元，第2个单元，…，第8个单元；第5位取1或0，含义自明。  
最终我们可以得到个字句。

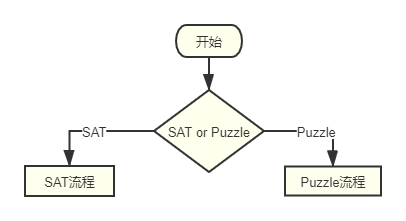
# 四、系统实现与测试

4.1 系统实现

本系统包含.h头文件、.cpp程序源码、数独CNF文件库，SATCNF文件库部分，在VS2019上运行。

具体运行流程如下所述：

程序开始运行进入选择菜单，用户根据提示进行输入。

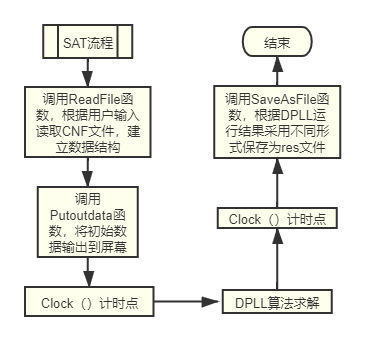


**图4.1 起始菜单选择流程**

之后进入SAT流程或者Puzzle流程的求解。

* + - SAT流程

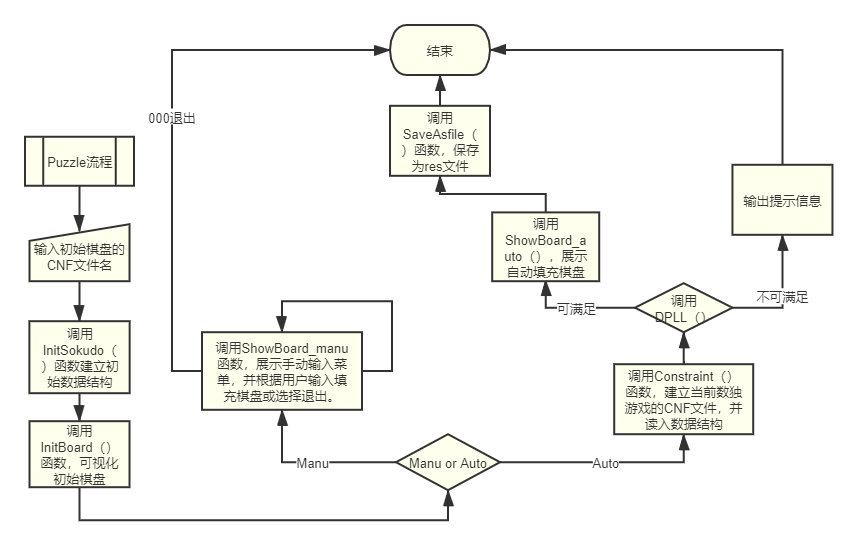
具体流程如图4.2：



**图4.2 SAT流程图**

* + - Puzzle流程

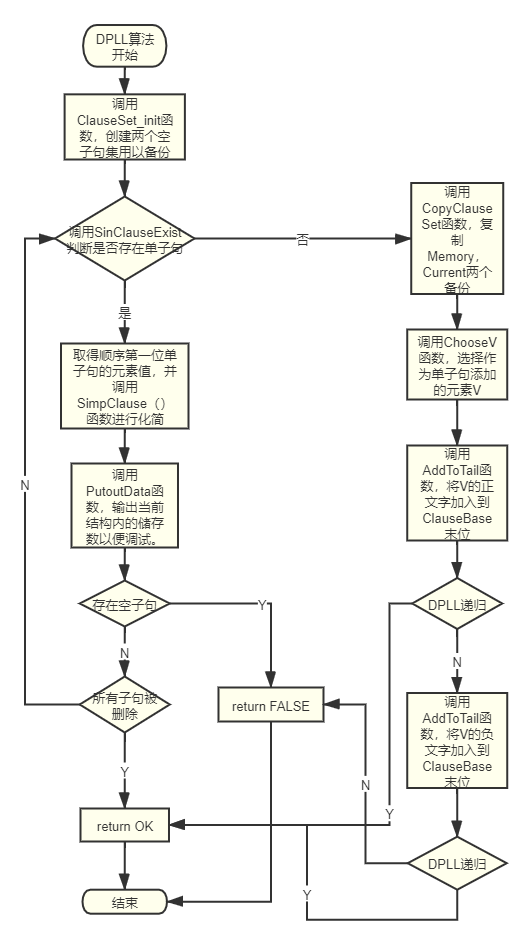
Puzzle的求解流程分为手动求解和自动求解两部分，具体流程如图4.3：



**图4.3 Puzzle流程图**

* + - DPLL算法流程

DPLL算法是上述流程的的核心步骤，也是本系统的核心算法，在此详细介绍其运行流程，具体实现流程如图4.4：



**图4.4 DPLL流程图**

* 1. 系统测试

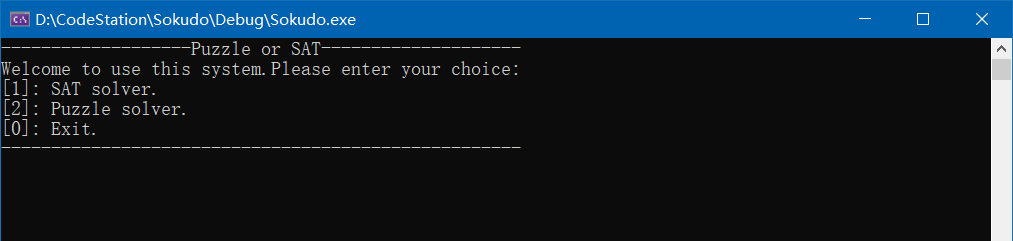
该部分对本系统各项功能进行详细测试，并展示测试效果。

* **主程序部分**

运行程序，首先会进入起始菜单，提供【1】、【2】、【0】三个选项。

1. 进入SAT求解器，直接读取CNF文件求解SAT问题；
2. 进入Binary Puzzle游戏；
3. 退出程序。

起始菜单界面如图4.5：

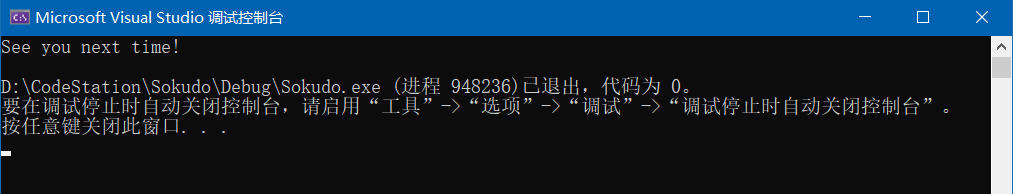


**图4.5 起始菜单界面**

若用户输入【1】，进入SAT求解器，测试见SAT求解器部分。

若用户输入【2】，进入Binary Puzzle游戏，测试见Binary Puzzle部分。

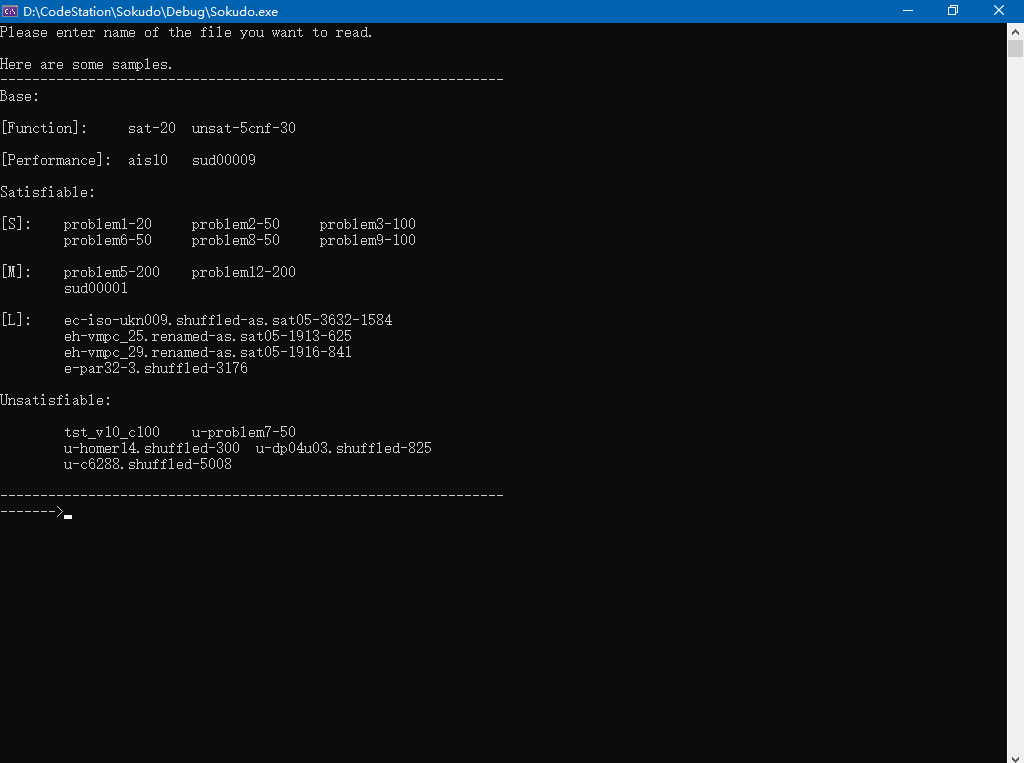
若用户输入【0】，退出程序，如图4.6：



**图4.6 退出程序界面**

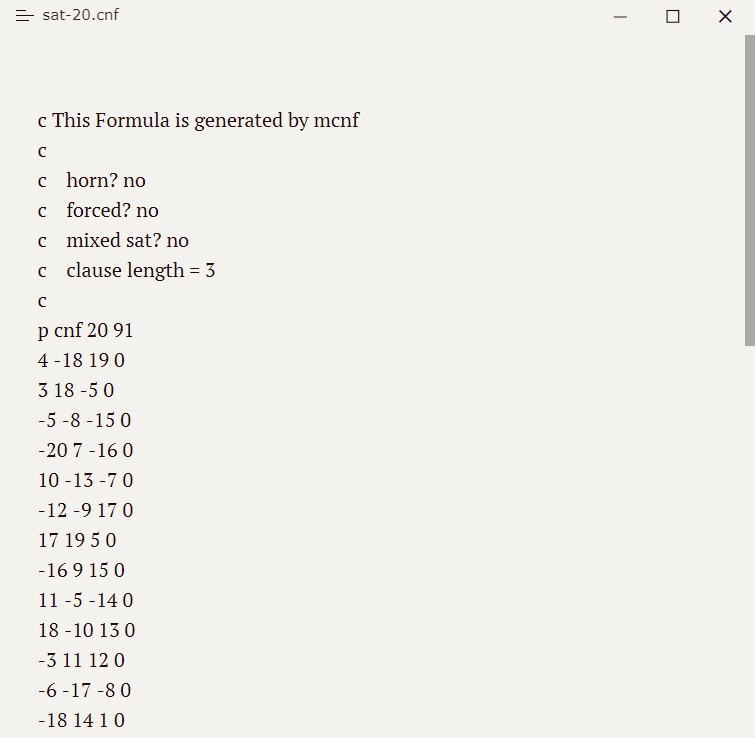
* SAT求解器部分

进入SAT求解器后，首先会提示用户输入将要读取的CNF文件，并列出了一部分不同大小不同类型算例的文件名作为提示，用户只需输入文件名，无需输入.cnf后缀。如图4.7：



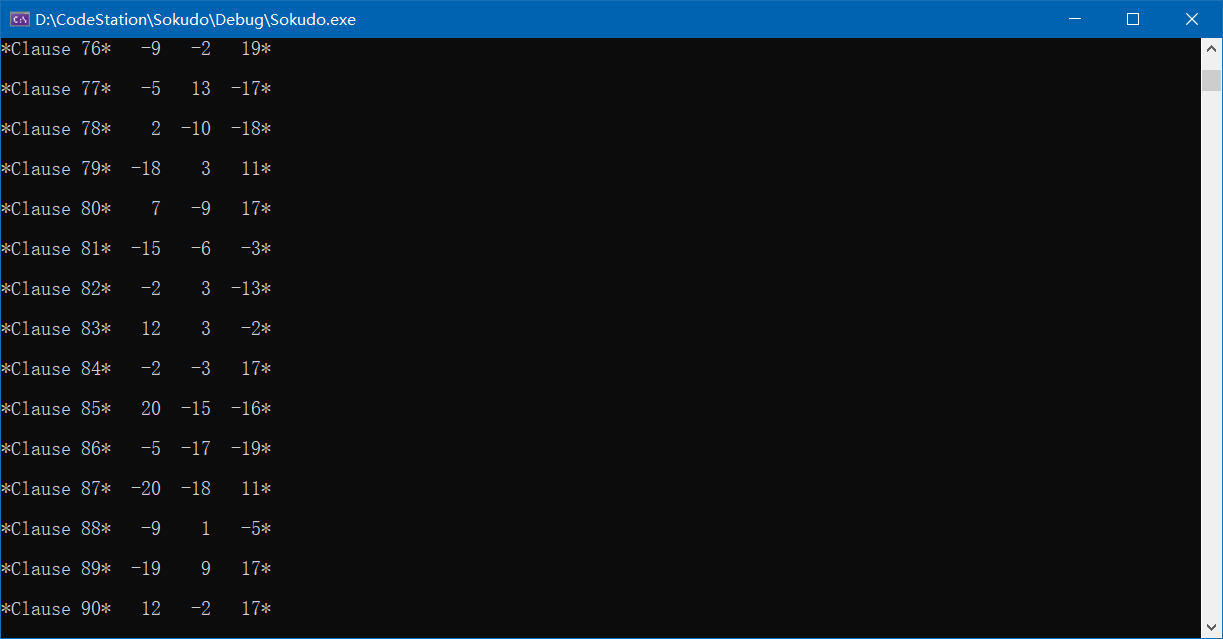
**图4.7 SAT起始菜单**

这里以可满足小算例sat-20.cnf作为示例，该文件部分内容如图4.8：



**图4.8 sat-20.cnf部分内容**

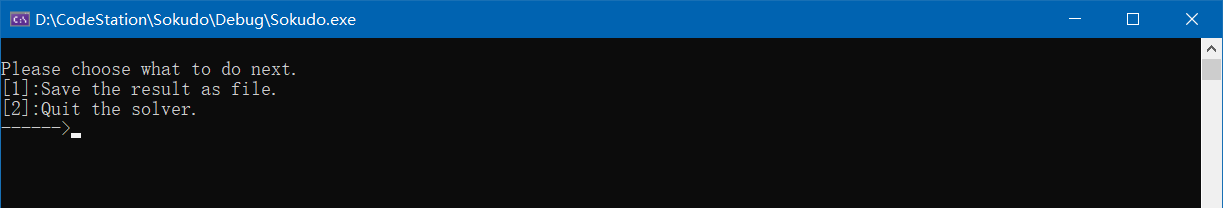
输入文件名sat-20后，系统便会进行DPLL算法求解，在求解之前，可选择会输出数据结构内的所有数据，以便人工校验，如图4.9：



**图4.9 算例数据**

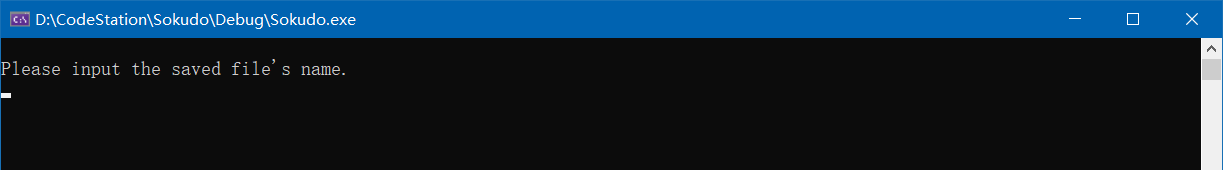
求解完成后，系统会显示求解结果，按下任意键后展现次级菜单提示用户结束求解器，或选择将求解结果保存为.res文件。如图4.10所示：





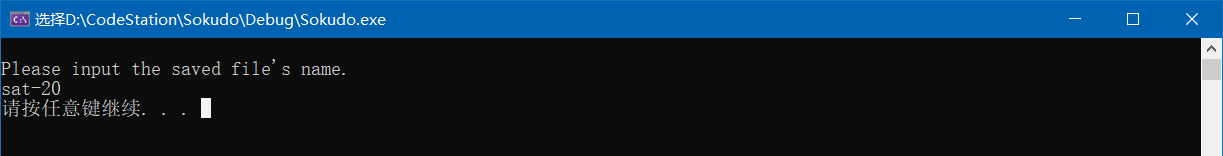
**图4.10 SAT次级菜单**

* + 若选择【1】，将会进入到文件保存操作，需用户输入文件名，系统会将执行结果生成相应的.res文件到库文件夹内。文件保存界面如图4.11：



**图4.11 文件保存界面**

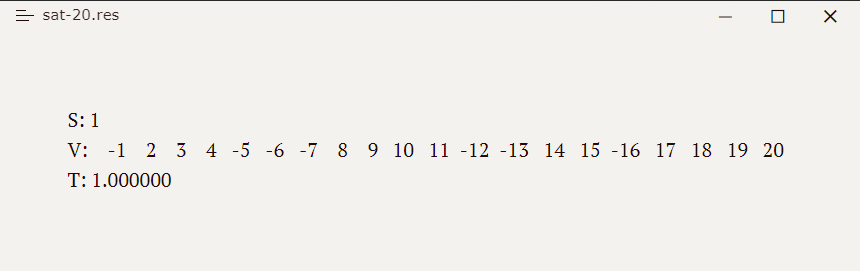
输入sat-20作为保存文件名，执行结果如下图：



**图4.12 文件保存执行结果**

按任意键结束SAT求解器，返回系统初始界面。

保存的sat-20.res文件内容如图4.13：



**图4.13 sat-20.res内容**

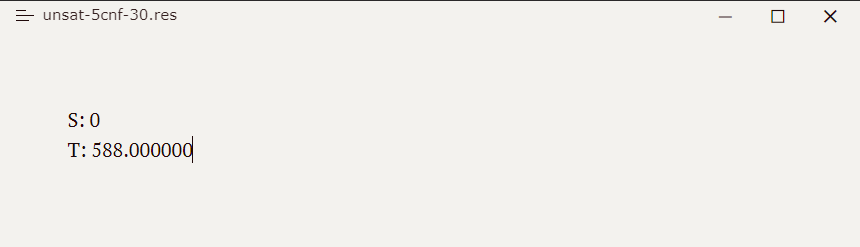
S代表运行结果，1为可满足，0为不可满足；V代表各变量的取值，正数为正文字，负数为负文字，未显示的变量说明被化简，对结果无影响；T为运行时间，单位ms。

其余基准情况如下：

* + - unsat-5cnf-30：  
      

**图4.14 unsat-5cnf-30.cnf运行结果**

该算例为不可满足算例，保存为文件时，不保存变量取值。

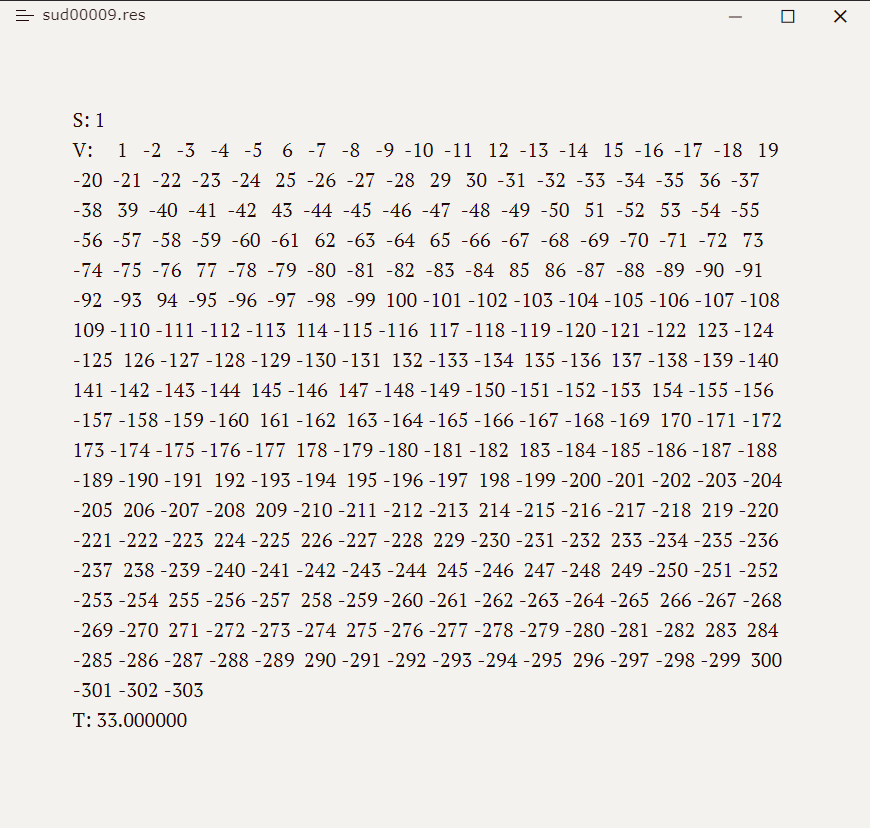


**图4.15 unsat-5cnf-30.res内容**

* + - ais10.cnf：  
      很遗憾，由于程序不够合理，导致元素数较大、字句数极大的算例求解所需空间超出正常范围，调整编译器后，可使用内存达到10G，仍未得到结果。
    - sud00009.cnf：



**图4.18 sud00009.cnf运行结果**



**图4.18 sud00009.res内容**

优化策略方面，将元素的选取由随机选取改为按照地址依次选取，得到了不错的优化效果。

综上，基准算例的测试情况，以及优化情况见下表：

**表4.1 基准算例测试**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | 是 | 2 ms | 1 ms | 50% |
| unsat-5cnf-30.cnf | 基准算例 | 否 | 1053 ms | 588ms | 44.1% |
| ais10.cnf | 基准算例 | 未知 | 未知 | 未知 | 未知 |
| sud00009.cnf | 基准算例 | 是 | 972 ms | 33ms | 96.6% |

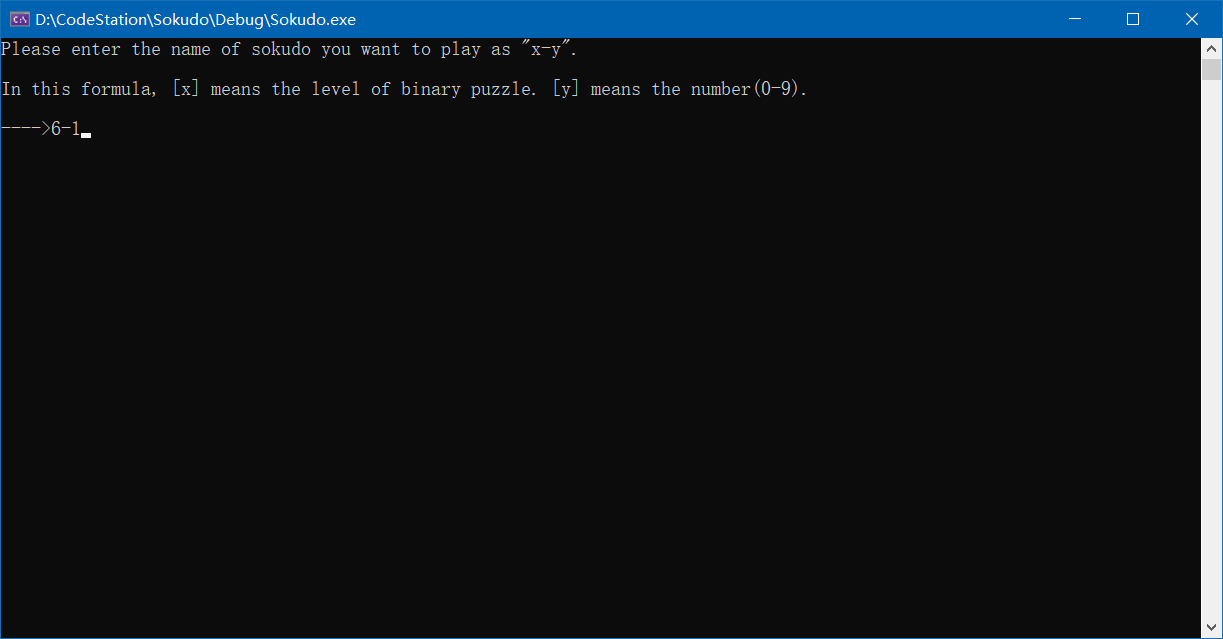
对于其他算例的测试结果，见表4.2

**表4.2 SAT算例测试结果**

|  |  |  |  |
| --- | --- | --- | --- |
| 算例名 | 运行结果 | 运行时间 | 所属类型 |
| problem1-20 | 可满足 | 1ms | 可满足算例 |
| problem2-50 | 可满足 | 4ms | 可满足算例 |
| problem3-100 | 可满足 | 362ms | 可满足算例 |
| problem5-200 | 可满足 | 4320ms | 可满足算例 |
| problem8-50 | 可满足 | 10ms | 可满足算例 |
| problem6-50 | 可满足 | 7ms | 可满足算例 |
| problem9-100 | 可满足 | 704ms | 可满足算例 |
| problem12-200 | 可满足 | 1702ms | 可满足算例 |
| problrm11-100 | 可满足 | 16ms | 可满足算例 |
| sud00001 | 可满足 | 83ms | 可满足算例 |
| sud00012 | 可满足 | 392ms | 可满足算例 |
| sud00021 | 可满足 | 536ms | 可满足算例 |
| sud00079 | 可满足 | 84ms | 可满足算例 |
| sud00082 | 可满足 | 422ms | 可满足算例 |
| sud00861 | 可满足 | 231ms | 可满足算例 |
| tst\_v10\_c100 | 不可满足 | 2ms | 不可满足算例 |
| u-problem7-50 | 不可满足 | 119ms | 不可满足算例 |
| u-dp04u03.shuffled-825 | 不可满足 | 14ms | 不可满足算例 |
| u-5cnf\_3500\_3500\_  30f1. shuffled-30 | 不可满足 | 638ms | 不可满足算例 |
| flat30-1 | 可满足 | 4ms | 其他可选择算例 |
| flat30-99 | 可满足 | 4ms | 其他可选择算例 |
| ais6 | 可满足 | 18ms | 其他可选择算例 |
| ais8 | 可满足 | 472ms | 其他可选择算例 |

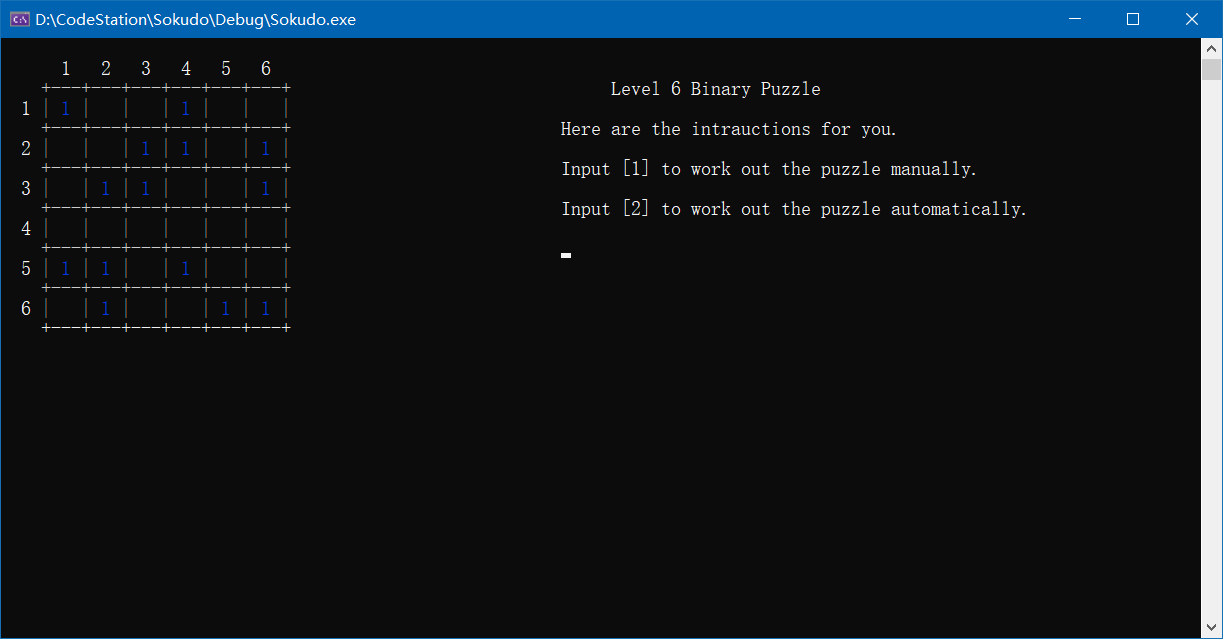
* + 若选择【2】，则将会直接退出求解器，返回系统起始菜单。
* Binary Puzzle游戏部分

进入Binary Puzzle游戏后，显示初始化界面，提示用户输入将要被读取的初始棋盘的文件名。格式为x-y，x代表阶数，y表示棋盘编号。这里输入6-1，即6阶1号棋盘作为测试，如图4.14：



**图4.14 Binary Puzzle游戏初始化界面**

之后便会产生Binary Puzzle游戏的初始棋盘以及菜单，界面如图4.15：

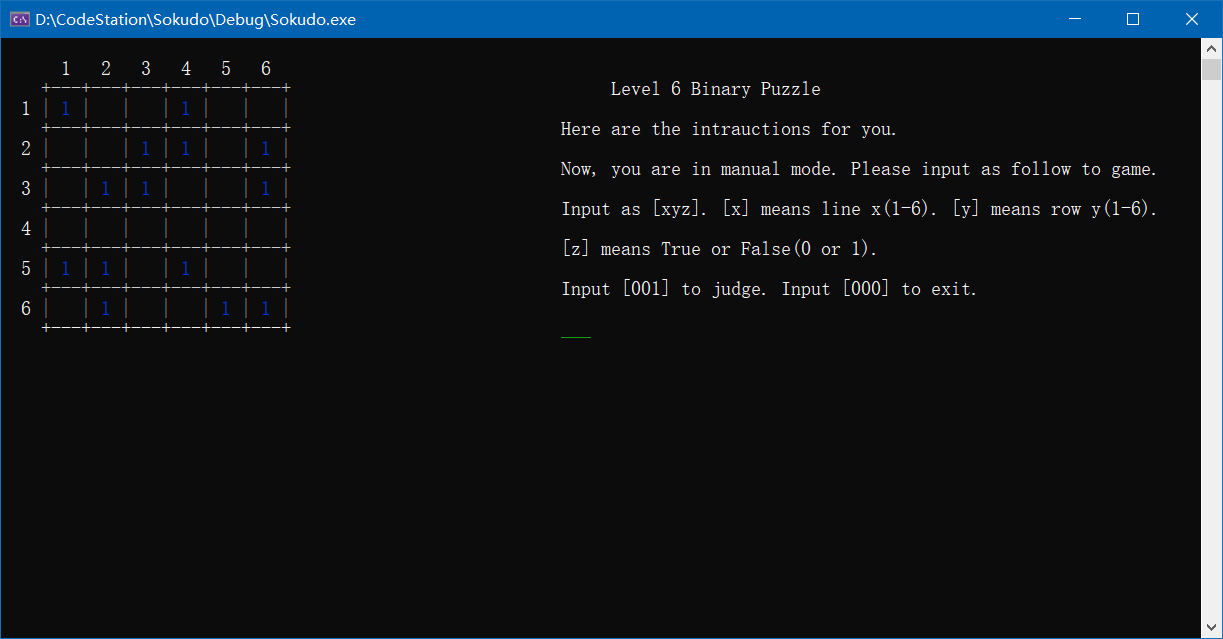


**图4.15 Binary Puzzle 初始棋盘及菜单**

图中左侧即为初始棋盘，在该棋盘中蓝色填充项为初始项，无法改变，其余空白的区域即为需要填充的部分。图中右侧为提示菜单，提示用户选择游戏方式，输入【1】，进入手动游戏模式；输入【2】，进入自动游戏模式。

* + 手动游戏模式

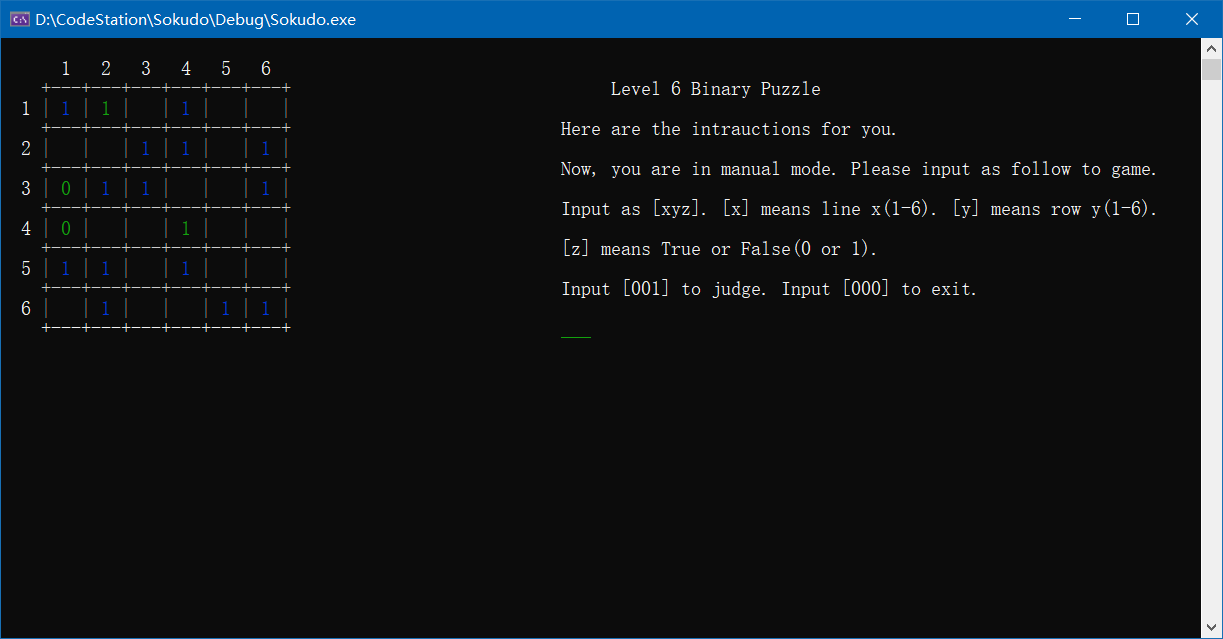
若用户选择进入手动游戏模式，右侧菜单将会更新为手动游戏模式菜单，如图4.16：



**图4.16 手动游戏模式**

手动输入格式为xyz，x代表行数，y代表列数，z代表置入的数字（0 或1）。输入的xyz如果为000，将会退出程序；如果为001，将会进入判定程序，判定当前游戏是否成功。

填充的示意图如图4.17，图中绿色数字为手动填入的数字，可被更改。



**图4.17 手动填充示意**

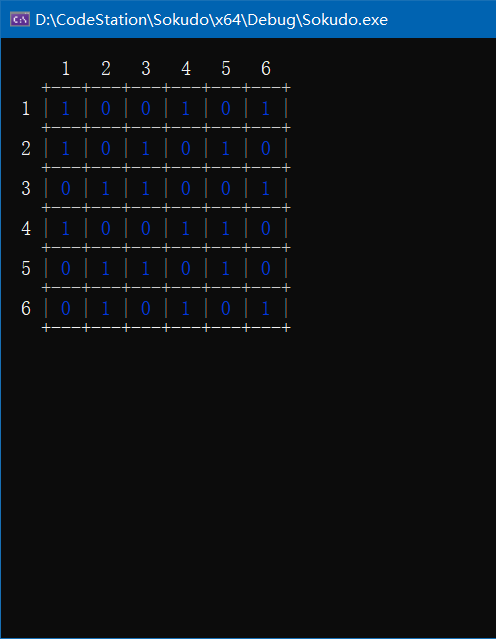
当填充完毕后可以输入001，进行判定，若判定成功，游戏结束，返回系统初始菜单

若判定失败，则会返回游戏界面。

若输出000则会退出Binary Puzzle游戏，返回系统初始菜单界面。

* + 自动游戏

用户进入自动求解模式后，系统将会自动完成Binary Puzzle游戏的求解，并自动将棋盘填充，界面右侧的菜单将会更新为自动模式的菜单。如图4.20所示：



**图4.20 自动游戏界面**

游戏完成后用户可按下任意键退出自动求解模式，返回系统初始菜单。

# 五、总结与展望

5.1工作总结

在本课程设计中，我完成的工作主要工作如下:

* 设计了合适的存储结构读取储存CNF文件；
  + - 完成了DPLL算法的C语言实现，并利用该算法解决了SAT问题；
    - 实现了一个比较美观，交互性较强的数独游戏界面，游戏可玩性较强；
    - 实现了数独游戏向SAT问题的转化，并可自动求解。

但是由于能力所限，本系统还有很多不足之处，部分想要完成的效果也未能完全实现。

* DPLL算法利用递归实现，效率较低，内存占用大，导致很多较大算例因内存溢出无法成功解决，或运行时间过长，结果未知；
* 数独游戏的界面比较简陋，虽然做了一定美化，但整体而言，没能做到更好的美观效果；
* 数独游戏的手动填充实现采用输入方式，比较繁琐，更希望实现鼠标的点击输入，但是未能成功实现；
* 数独游戏的自动求解填充不够完善，某些情况下可能触发异常未能完全解决。

5.2工作展望

在本次课程设计的综合检验下，对自己的知识能力和技能水平有了新的认识，发现了许多不足之处，希望在未来的学习生活自己能够：

* 加强基础知识的巩固，在本次课程设计中，发现自己对与数据结构基础知识的掌握还有很多不足，对C语言的一些特性也有所混淆，基础能力仍然需要加强；
* 提高大工程程序的编写能力，在以往的课程中编写的都是单个文件或者较小的工程文件，较易完成调试和优化工作。本次程序设计中，在debug过程中遇到了很大的困难，花费了很多时间。
* 提高知识的深度，在本次课程设计中，一些想要实现的效果没有实现思路，参阅资料后了解一些没有接触过的实用知识，希望今后继续提高知识的深度。

# 六、心得体会

本次程序设计是自己第一次尝试实现完整的较大型工程，也是对之前相关专业知识的综合考核。经过C语言相关课程的学习，掌握了高级编程语言的基础知识，并能够实现一些简单的函数功能以及小程序；通过数据结构的学习，掌握了数据处理的基础知识。但这些基础知识还没能实现系统的结合，以解决较复杂的实际应用问题，本次课程设计就达到了这样的一个目标。

首先是关于本课程设计的数据处理方面，这是该工程的基础，考虑到实现的复杂性，使用了“顺序表+链表”的方式，但是实际结果并不理想，没有发挥到顺序表的优势，导致大算例求解遇到了很多问题。但是测试时各项功能函数都是基于该数据结构实现的，修改基本数据结构会带来巨大的工作量，最后选择了放弃修改。今后的实践中一定要事先做好规划，否则可能会为今后的工作埋下很多隐患。

接着是DPLL算法求解SAT问题方面，参阅了很多资料后，了解了该问题的发展历史以及重要地位。考虑到工作效率，选择了DPLL算法的递归实现，对于中小算例，个人电脑还可满足，但是递归产生的内存占用使得大算例难以得到解决。以空间换时间并非是很好的解决方法，在今后尽量尝试功能的非递归实现，要充分考虑到工程的实际性能需求。

数独游戏是该程序设计的实际应用，在实现该实际问题时，因为需要考虑到和用户的交互以及美观问题做了很多修改，但因为C++、Python并不熟悉，主体依然是采用C语言，界面的美观效果和预想情况还是有很大差异。

在本次设计中，最耗费时间也最难以解决的问题就是调试。对于断点，单步调试的使用多基于对小程序的了解，在一个大工程里，分析未知的问题位置，并找出解决方法，对自己来说还是相当大的考验。

在一次又一次的调试过程里，在原先的算法里发现了很多问题，最典型的是在DPLL算法中，因为要采取迭代操作，根据情况可能会对同一个对象进行两次次级迭代，原先直接采用当前对象作为参数，导致当前对象被改变，递归出现错误，经过断点定位，逐步调试后发现了问题，并进行的改正。

综上，通过这次课设，自己对于C语言及其工程有了更好的认识，锻炼了自己的结构设计能力以及调试代码的能力。这次课设也让我们掌握了很多新知识，对于工程的系统化设计有了更多的认识。

参考文献

[1] Putranto H. Utomo and Rusydi H. Makarim. Solving a Binary Puzzle. Math. Comput. Sci. (2017) 11:515–526.

[2] Putranto Utomo and Ruud Pellikaan. Binary puzzle as a SAT problem. The 2017 Symposium on Information Theory and Signal Processing in the Benelux, May 11-12, 2017, Delft, the Netherlands. R. Heusdens and J. H. Weber (Eds.), pp. 223-229 , 2017.

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

[4] 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.

附录 程序使用手册

该部分对本系统的使用提供详细引导，用户根据该手册可实现系统的全部功能。该系统分为主程序、SAT求解器、Binary Puzzle游戏三部分。系统的使用说明如下：

* **主程序部分**

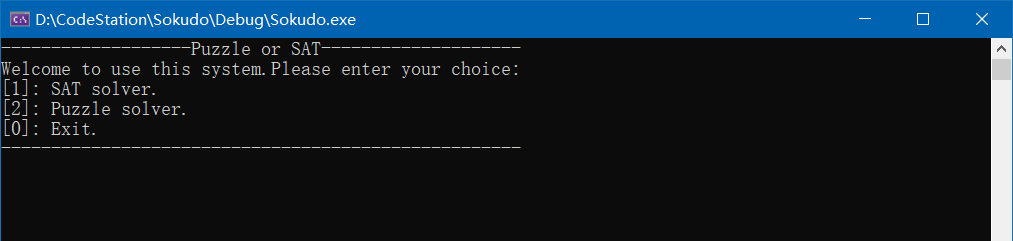
运行程序，首先会进入起始菜单，提供【1】、【2】、【0】三个选项。

【1】进入SAT求解器，直接读取CNF文件求解SAT问题；

【2】进入Binary Puzzle游戏；

【0】退出程序。

起始菜单界面如图1：

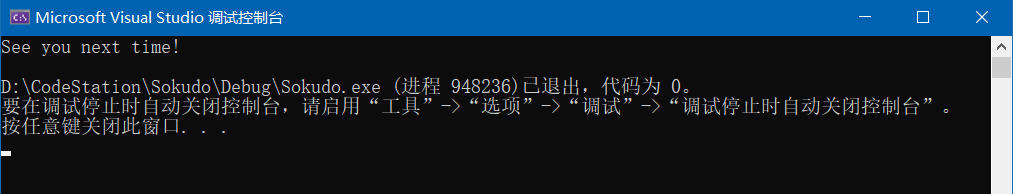


**图1 起始菜单界面**

若用户输入【1】，进入SAT求解器，测试见SAT求解器部分。

若用户输入【2】，进入Binary Puzzle游戏，测试见Binary Puzzle部分。

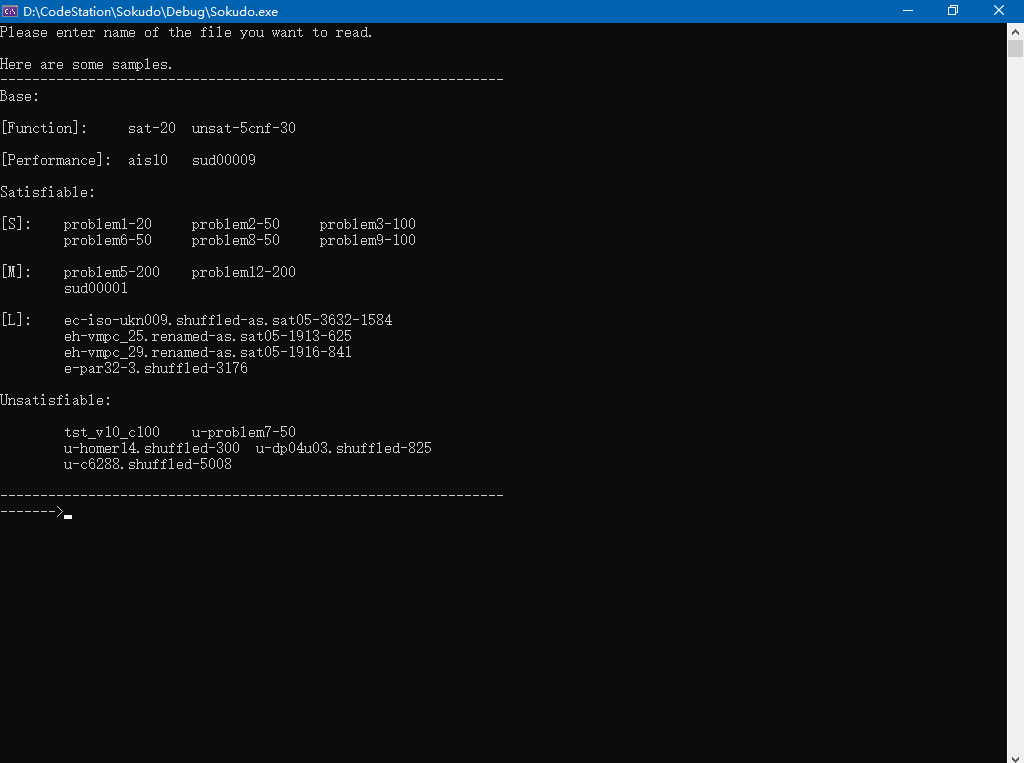
若用户输入【0】，退出程序，如图2：



**图2 退出程序界面**

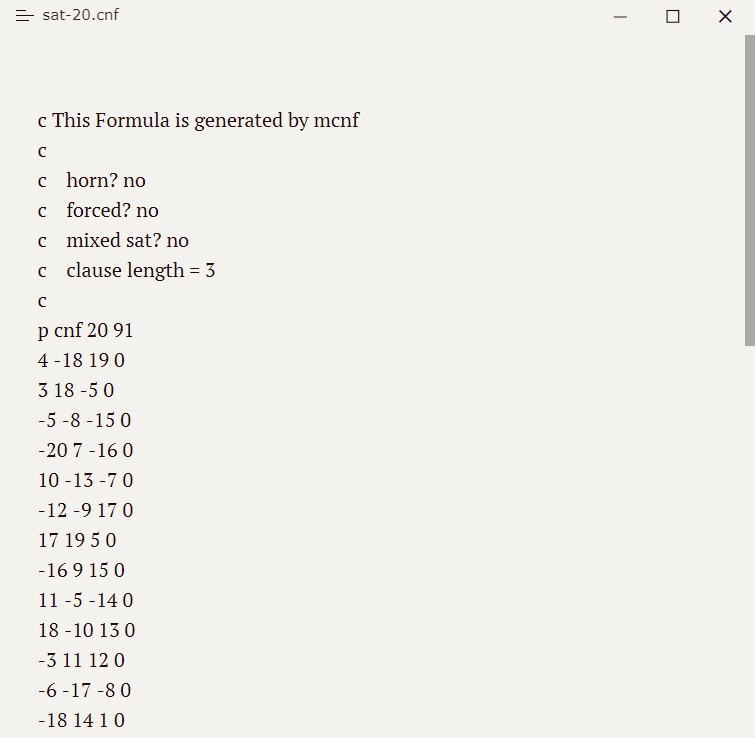
* SAT求解器部分

进入SAT求解器后，首先会提示用户输入将要读取的CNF文件，用户只需输入文件名，无需输入.cnf后缀。如图3：



**图3 SAT起始菜单**

以可满足小算例sat-20.cnf作为示例，该文件部分内容如图4：



**图4 sat-20.cnf部分内容**

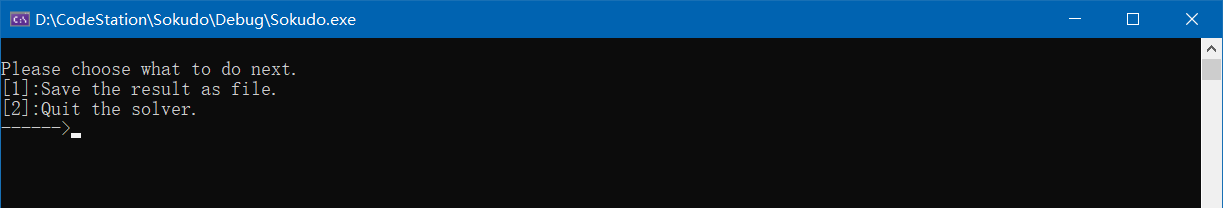
输入文件名sat-20后，系统便会进行DPLL算法求解。

求解完成后，系统会显示求解结果， 如图5：



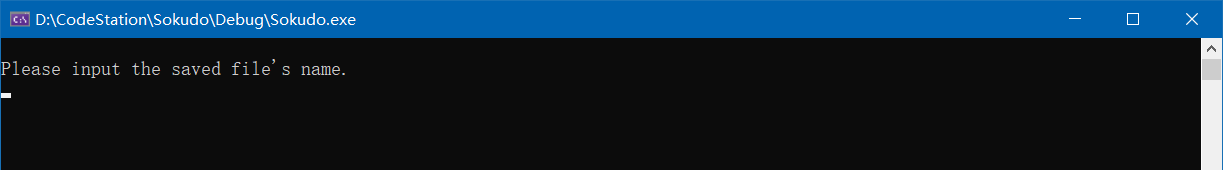
**图5 求解结果**

按下任意键后进入次级菜单，次级菜单提供两个选项，如图6所示：



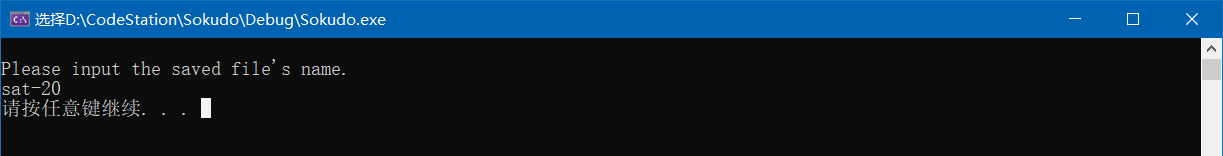
**图6 SAT次级菜单**

* + 若选择【1】，将会进入到文件保存操作，用户输入文件名，系统会将执行结果生成相应的.res文件到库文件夹内，相对路径为 ./Out/。文件保存界面如图7：



**图7 文件保存界面**

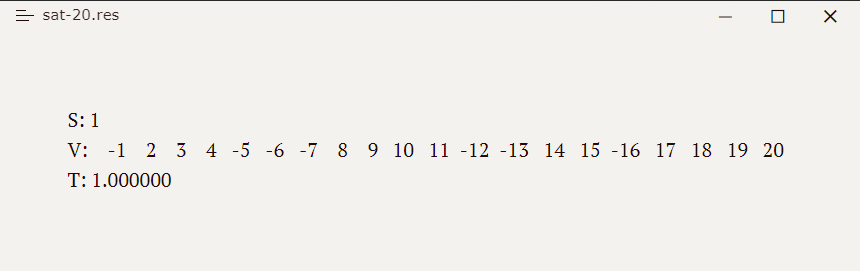
输入sat-20作为保存文件名，执行结果如下图：



**图8 文件保存执行结果**

按任意键结束SAT求解器，返回系统初始界面。

保存的sat-20.res文件内容如图9：

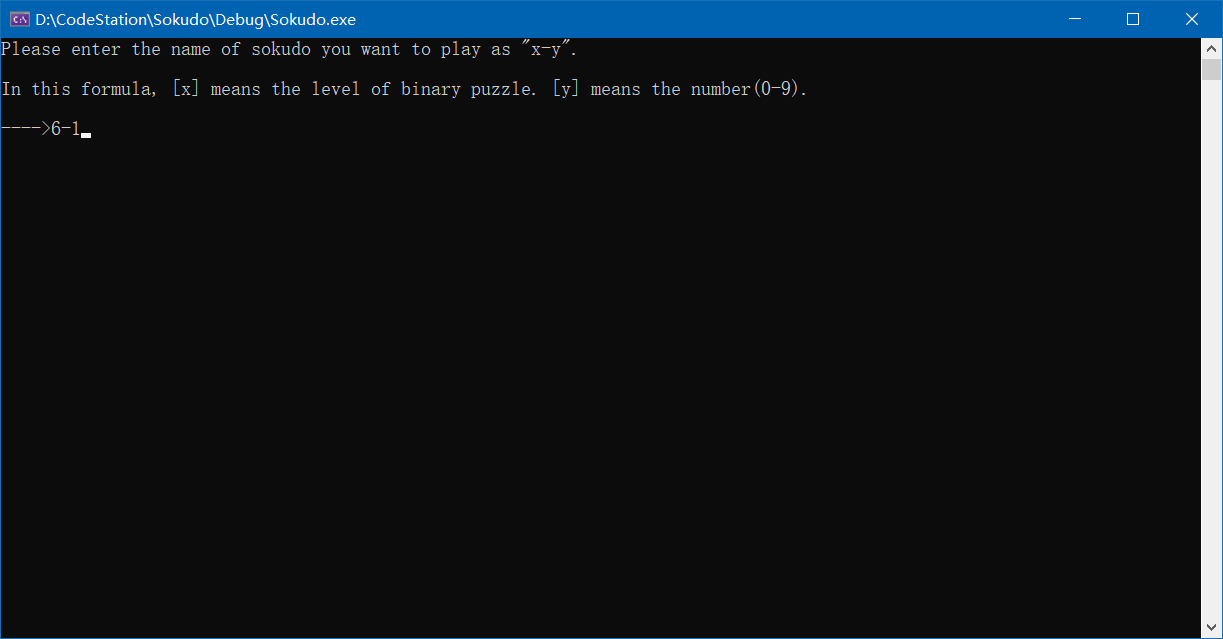


**图9 sat-20.res内容**

S代表运行结果，1为可满足，0为不可满足；V代表各变量的取值，正数为正文字，负数为负文字，未显示的变量说明被化简，对结果无影响；T为运行时间，单位ms。

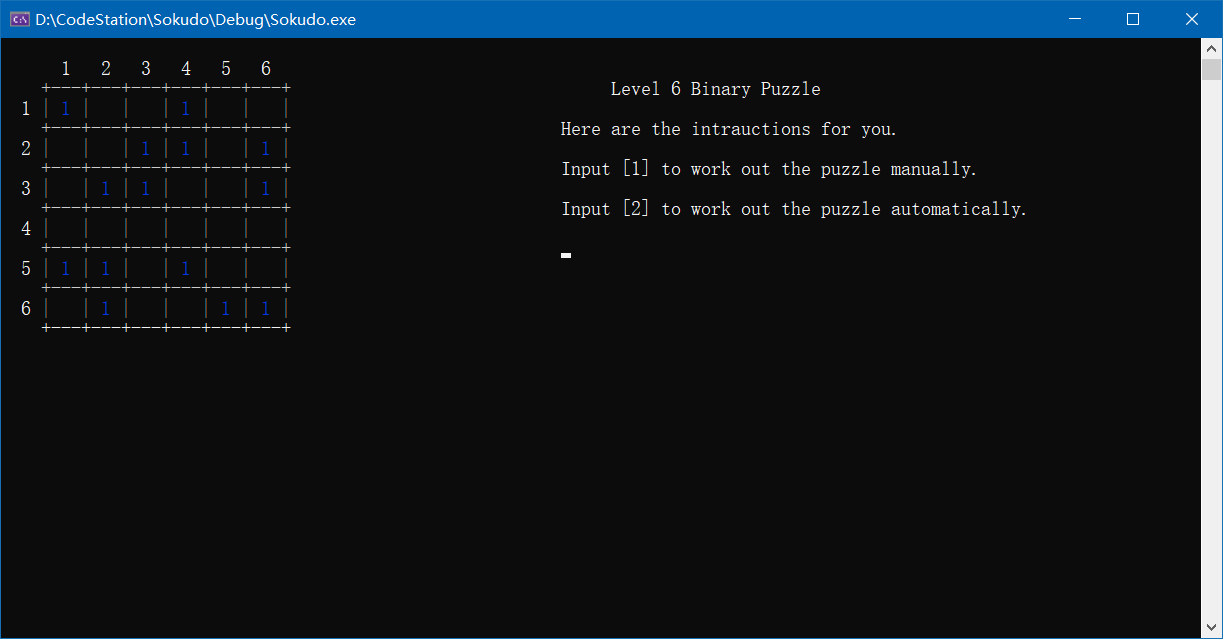
* + 若选择【2】，则将会直接退出求解器，返回系统起始菜单。
* Binary Puzzle游戏部分

进入Binary Puzzle游戏后，显示初始化界面，用户需输入将要被读取的初始棋盘的文件名。格式为x-y，x代表阶数，y表示棋盘编号。这里输入6-1，即6阶1号棋盘作为测试，如图10：



**图10 Binary Puzzle游戏初始化界面**

之后便会产生Binary Puzzle游戏的初始棋盘以及菜单，界面如图11：



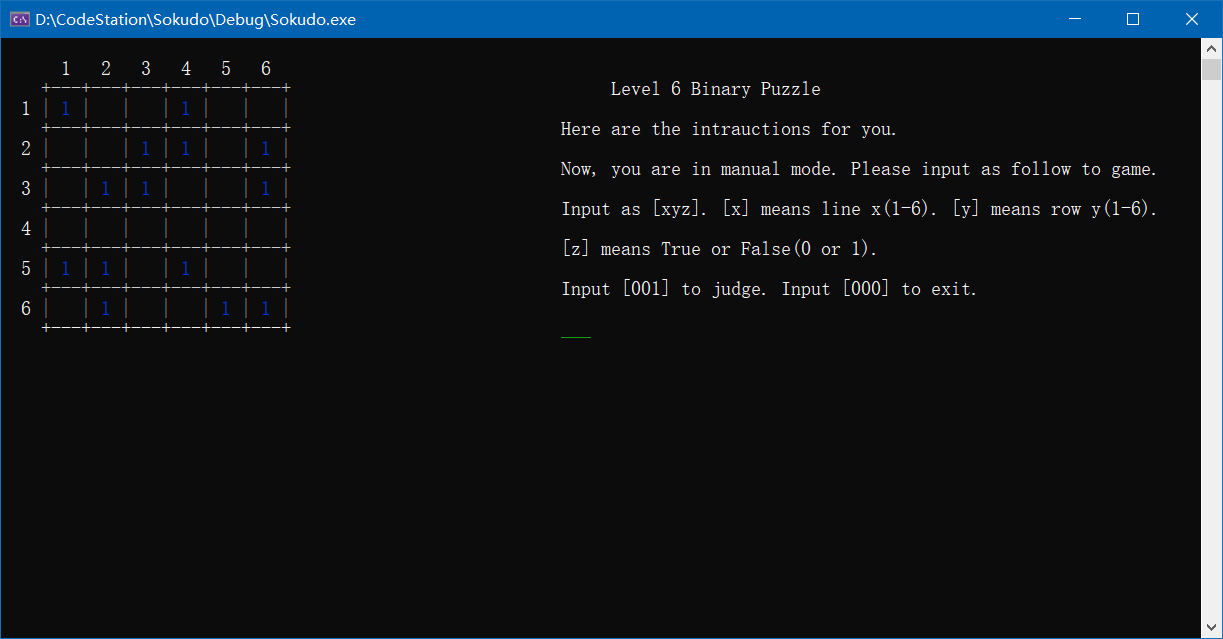
**图11 Binary Puzzle 初始棋盘及菜单**

图中左侧即为初始棋盘，在该棋盘中蓝色填充项为初始项，无法改变，其余空白的区域即为需要填充的部分。

图中右侧为提示菜单，用户选择游戏方式，输入【1】，进入手动游戏模式；输入【2】，进入自动游戏模式。

* + 手动游戏模式

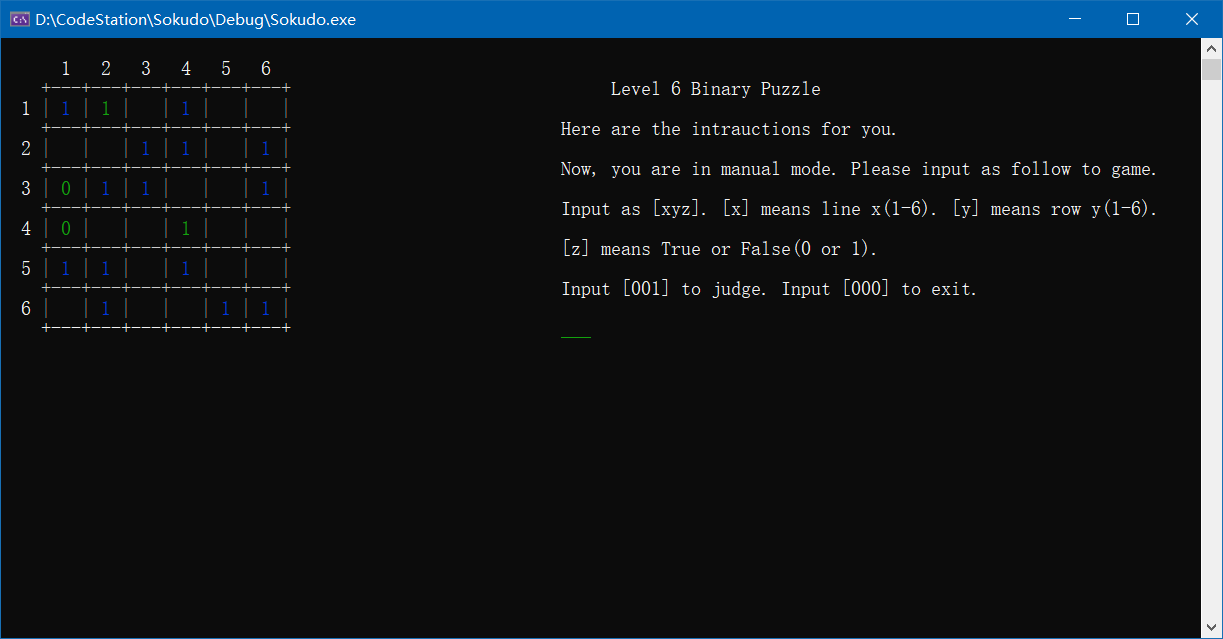
若用户选择进入手动游戏模式，右侧菜单将会更新为手动游戏模式菜单，如图12：



**图12 手动游戏模式**

手动输入格式为xyz，x代表行数，y代表列数，z代表置入的数字（0 或1）。输入的xyz如果为000，将会退出程序；如果为001，将会进入判定程序，判定当前游戏是否成功。

填充的示意图如图13，图中绿色数字为手动填入的数字，可被更改。



**图4.17 手动填充示意**

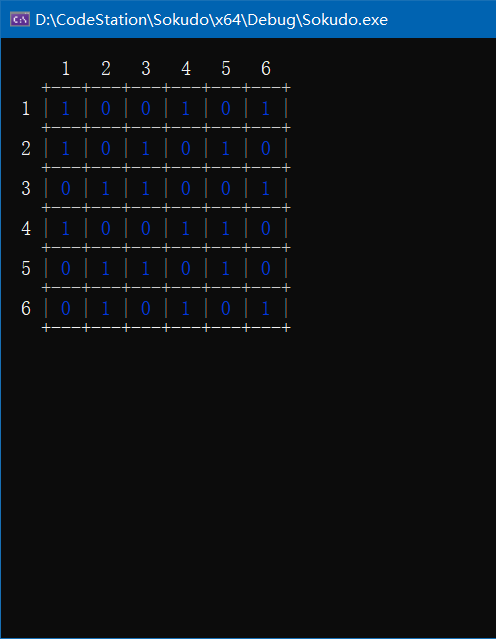
当填充完毕后可以输入001，进行判定，若判定成功，游戏结束，返回系统初始菜单。

若判定失败，则会返回游戏界面。

若输出000则会退出Binary Puzzle游戏，返回系统初始菜单界面。

* + 自动游戏

用户进入自动求解模式后，系统将会自动完成Binary Puzzle游戏的求解，并自动将棋盘填充。如图19所示：



**图19 自动游戏界面**

游戏完成后用户可按下任意键退出自动求解模式，返回系统初始菜单。

以上即为本程序的详细介绍。