**Sudoku详细设计**

1. **文档概述**

此概要说明对Sudoku软件的功能做了详细设计的说明，为代码实现奠定了基础。

1. **模块详细概述**
   1. **Sudoku类模块**
      1. **求解数独函数**

最开始的想法无论是求解数独还是生成终局，全部暴力+回溯，但是在1e6的大数量情况下显然会很慢。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 函数名 | SolveSudoku | | | | | | |
| 文件名 | Mysudoku/Mysudoku/Sudoku.cs | | | | | | |
| 功能 | 求解数独 | | | | | | |
| 记述形式 | public bool SolveSudoku( int position) | | | | | | |
| 参数 | | | | | | | |
| 名称 | | | 类型 | | | 含义 |  |
| Position | | | Int | | | 递归序号 |  |
| 返回值 | | 类型 | | Bool | 说明 | | |
|  | | 值 | | True | 表示有可行解 | | |
|  | |  | | False | 表示没有可行解 | | |
| 详细说明 | | | | | | | |
| 通过递归算法来进行回溯，外加预处理来剪枝。 | | | | | | | |

经过思考，想了想还是用递归，不过要大量剪枝，先做预处理。给Sudoku类加了三个数组分别是hang[9,9]，lie[9,9]，sansan[9,9]，即牺牲内存减少时间，第一个参数分别代表了第i行，第i列，第i个小3×3矩阵，第二个参数代表这行出现的数字，如果出现则标记为true，没有出现则标记false，这个方法极大地缩短了时间。具体设计如下：

* + 1. **生成终局函数**

最开始想暴力+回溯，外面再套一个大循环，显然在1e6的大数量情况下会很慢。

在网上搜索了很多的数独相关资料，发现最快的是模板法，即生成某一行或某一列或某个3\*3小矩阵填入模板，即可构成一个完整的数独终局，由于左上角的数字固定为后两位模9+1（我的是（3+5）mod 9 + 1 =9），故共有8！= 40320种，另外23行（第一行固定不动），456行，789行也可互相交换，数独性质仍成立，故一共有40320 × 2！× 3！× 3！约等于3e6，超过了要求的1e6。

这个需要写一个全排列函数，C++有直接的函数可以调用，考虑到C#没有，而且如果之后做UI（学期内应该是没有时间做了）可以游玩，这种方法生成的数独矩阵很固定可玩性不高，于是想到了生成随机数，思路又回归到了回溯法上。

先思考我们也可以实现填上一部分数字，然后用回溯方法，这里可以调用求解数独的函数。一共有9个3×3矩阵，对角线上的3个矩阵是互不干扰的，所以可以随机生成这三个矩阵，然后用优化后的回溯来生成数独解，回溯方法生成的是固定解，所以种子相同，解就会相同，所以这种方法原则上可以生成8！× 9！× 9！约等于1e15，所以添加一个随机函数来打乱1-9的数组填进去就可以了，推测重复的概率会非常小。同时，每次生成完一个终局后要初始化Sudoku类的对象，重置预处理的标记等。具体函数如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数名 | GenerateSudokuEnding | | | |
| 文件名 | Mysudoku/Mysudoku/Sudoku.cs | | | |
| 功能 | 生成数独终局 | | | |
| 记述形式 | public void GenerateSudokuEnding() | | | |
| 参数 无 | | | | |
| 返回值 | | 类型 | 无 | 说明 |
|  | | 值 | 无 | 无 |
| 详细说明 | | | | |
| 通过一定的算法生成数独终局。 | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 函数名 | Generate33 | | | | | |
| 文件名 | Mysudoku/Mysudoku/Sudoku.cs | | | | | |
| 功能 | 随机生成第1 5 9个3×3矩阵 | | | | | |
| 记述形式 | private void Generate33(int[] gene, int num) | | | | | |
| 参数 无 | | | | | | |
| 名称 | | | 类型 | | | 含义 |
| gene | | | Int[] | | | 生成随机数组的初始数组 |
| num | | | Int | | | 要生成的小矩阵的序号 |
| 返回值 | | 类型 | | 无 | 说明 | |
|  | | 值 | | 无 | 无 | |
| 详细说明 | | | | | | |
| 由于第一个矩阵第一个数字不能动，所以将1与5，9区别开来，考虑到分开写两个函数重复度很大，故将其重构成一个函数，用标号加以区分，首先调用void Program.Random159（int[]）函数生成随机数组，然后将数字填进去，并且预处理。 | | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数名 | Initialize | | | |
| 文件名 | Mysudoku/Mysudoku/Sudoku.cs | | | |
| 功能 | 初始化对象内部的字段和属性 | | | |
| 记述形式 | public void GenerateSudokuEnding() | | | |
| 参数 无 | | | | |
| 返回值 | | 类型 | 无 | 说明 |
|  | | 值 | 无 | 无 |
| 详细说明 | | | | |
| 将数独初始化，除了第一个位置为固定数字外，其余的均为0，并且预处理的数组也相应初始化。 | | | | |

* 1. **控制模块（输入输出模块）**
     1. **主函数**

主控模块（Main）主要用来接收参数并判断命令，由于只有两个命令所以判断都写在了主函数中，日后有时间可以进行重构，使代码看起来清晰美观，若增加命令也可复用。

控制台接收命令参数，主要从args这个字符串数组传入，以空格分割，一个参数存入一个单元，由于两个命令都是双目操作，所以数组长度如果不等于2则判断命令有误。

当命令为-c 时，第二个参数我们试图将其变为整型int.TryParse(args[1], out parseInt)，如果合法则调用生成数独函数，生成终局写入txt文件中，不合法则报错。

当命令为-s 时，第二个参数我们先检验书读文件绝对路径是否存在，如果存在则按行读取直到文件尾，读入的数据存入到一个string中，再按空格进行分割到string数组中，然后每81个构成一个数独谜题求解输出，直到将所有的数字用完。

* + 1. **输入输出函数**

输入输出均采用流读写。按行读入，按字符或行写出。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 函数名 | OutputToTxt | | | | | |
| 文件名 | Mysudoku/Mysudoku/Program.cs | | | | | |
| 功能 | 输出到文本。 | | | | | |
| 记述形式 | static void OutputToTxt(Sudoku sudoku, StreamWriter streamWriter) | | | | | |
| 参数 无 | | | | | | |
| 名称 | | | 类型 | | | 含义 |
| sudoku | | | Sudoku | | | 生成的数独 |
| streamWriter | | | StreamWriter | | | 流写入 |
| 返回值 | | 类型 | | 无 | 说明 | |
|  | | 值 | | 无 | 无 | |
| 详细说明 | | | | | | |
| 按行或字符写入txt文件中 | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 函数名 | InputSudoku | | | | | |
| 文件名 | Mysudoku/Mysudoku/Program.cs | | | | | |
| 功能 | 读入txt文本并存入字符串中 | | | | | |
| 记述形式 | static string InputSudoku(string allTheText, string path) | | | | | |
| 参数 无 | | | | | | |
| 名称 | | | 类型 | | | 含义 |
| allTheTxt | | | String | | | 文件内容的载体 |
| path | | | String | | | 文件的绝对路径 |
| 返回值 | | 类型 | | String | 说明 | |
|  | | 值 | | String | 返回读入文本的字符串 | |
| 详细说明 | | | | | | |
| 按行读入文本，并将所有内容写入一个字符串并返回，行与行之间用空格衔接。 | | | | | | |

* + 1. **随机函数**

手动写了一个打乱数组的算法，即在当前数组号中随机抽一个，把该位置的元素与末尾元素交换，当前数组号减1，重复上述步骤直到数组号为1，因为用不到Sudoku类内的元素，所以写在了主函数类中。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 函数名 | Random159 | | | | | |
| 文件名 | Mysudoku/Mysudoku/Program.cs | | | | | |
| 功能 | 生成随机数组 | | | | | |
| 记述形式 | static string InputSudoku(string allTheText, string path) | | | | | |
| 参数 无 | | | | | | |
| 名称 | | | 类型 | | | 含义 |
| gene | | | Int[] | | | 生成随机数组的初始数组 |
| 返回值 | | 类型 | | 无 | 说明 | |
|  | | 值 | | 无 | 无 | |
| 详细说明 | | | | | | |
| 按照上面的算法生成一个随机数组。 | | | | | | |

* 1. **改进与提升**

输出发现有一些数独连续几个是完全相同的，查找了很多原因都没能解决，最终找到了问题的根本原因——随机函数有周期。

通过查资料了解到，所有的随机函数都是伪随机函数，默认的Random是以时间为种子的，短时间内生成的随机数是相同的（这也就解释了写完发现有好几个终局是连续相同的），后来查到了一个延时的算法来上随机函数的种子刷新，random之前加入了sleep函数，效果还算可以，但也正是因为延时，所以时间变得很慢。

后来查到了一个获得随机数的办法，用一个加密算法生成一串比特位，然后将这个比特位转换成整型数作为种子即可。

这个方法生成的终局目前来看没有重复的，而且几率很小。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 函数名 | GetRandomSeed | | | |
| 文件名 | Mysudoku/Mysudoku/Program.cs | | | |
| 功能 | 生成随机数组 | | | |
| 记述形式 | static int GetRandomSeed() | | | |
| 参数 无 | | | | |
| 返回值 | | 类型 | Int | 说明 |
|  | | 值 | Int | 生成一个随机种子 |
| 详细说明 | | | | |
| 用加密算法生成一个随机数的种子。 | | | | |

1. **数据结构定义**

class Sudoku

{

public void GenerateSudokuEnding()

{

//填充部分

}

public void SovleSudoku()

{

//填充部分

}

public void Initialize();

private void Generate33(int[] gene, int num)

private char[,] shudu = new char[9, 9];

public char[,] Shudu

{

get { return shudu; }

set { shudu = value; }

}

private bool[,] hang = new bool[9, 9];

public bool[,] Hang

{

get { return hang; }

set { hang = value; }

}

private bool[,] lie = new bool[9, 9];

public bool[,] Lie

{

get { return lie; }

set { lie = value; }

}

private bool[,] sansan = new bool[9, 9];

public bool[,] Sansan

{

get { return sansan; }

set { sansan = value; }

}

}

}

class Program

{

static void Main(string[] args)

{

//填充部分

}

static void OutputToTxt(Sudoku sudoku,StreamWriter streamWriter)

{

//填充部分

}

static string InputSudoku(string allTheText, string path);

static int GetRandomSeed();

public static void Random159(int[] gene);

}