**Sudo详细设计**

1. **文档概述**

此概要说明对Sudo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. **Sudo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。

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：

* 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。

* + 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，为用不到Sudoku类内的元素，所以写在了主函数类中。

* 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。

1. **数据结构定义**