数独终盘生成的算法研究与实现

李 冰 李 林

四川大学锦城学院 四川 成都 611731

【摘 要】数独游戏风靡一时，受到广大青少年的喜爱。本文使用 JS 实现数独，从生成数独终盘、挖洞到解题，给出了详细的代码演示。并且深入探究了数独终盘生成的两种方法：模板法和回溯法。

【关键词】JS；数独；终盘生成；模板法；回溯法；

1 引言

数独源自于 18 世纪瑞士的一种数学逻辑推理趣味游戏，它的雏形为瑞士数学家欧拉等人研究的拉丁方阵，逐渐发展至今。它由 9 行 9 列小方格组成，每个 3*3 方格又被称为一个 9 宫格，它的规则很简单，在小方格内填写 1~9 之间的数字，并且每行、每列以及每个 9 宫格都不允许有重复的值。

规则虽然简单，但数独终盘却有很多种，2005 年 Bertram Felgenhauer 和 Frazer Jarvis 计算出 6,670,903,752,021,072,936,960 种组合，不计等价的数独终盘也有 5,472,730,538 种组合。那么该如何利用计算机来实现符合规则的数独终盘并且使其生成的终盘数量更多？这里用到了两种不同的方法——模板法和回溯法，而每种方法都有自己亮点与不足。

研究生成数独终盘方法的最终目的是为了能够更好的实现数独游戏，像上面说的，数独终盘有很多，那么由其衍生出的数独初盘就更加不计其数了，这里也要考虑到同一初盘在不同玩家手里的解法或许不止一种。

2 相关研究

对于一个完整的数独游戏，玩家首先看到的是一个数独初盘，然后进行填空，最后完成数独成为终盘。

而对于设计者来说，实现数独主要包括两个方面：

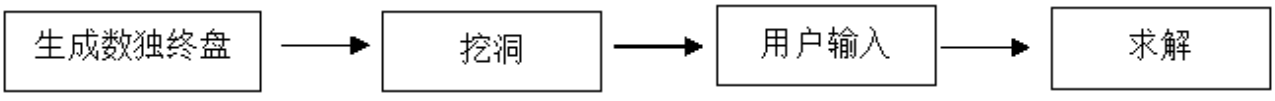
生成数独初盘。

生成数独初盘又包括了两个方面：生成数独终盘以及挖洞，即在生成的数独终盘上进行挖洞，挖去一部分方格的数据，这样便形成了一个合格的数独初盘。在数独初盘的实现上，挖洞并不难，难点主要在于生成数独终盘部分，目前研究数独的论文中常用的算法之一便是回溯法[1]，回溯法是一种算法思想，按深度优先的算法，从根结点开始搜索解空间树[2]，回溯法有递归回溯和迭代回溯两种方法，不同的方法采用的算法不同，但解决的思想都是在解空间中进行深度搜索[3]。这里通过用递归实现的回溯法找出满足数独规则约束条件的所有解，在解集中查找合适的值，若找到就继续按深度优先算法搜索；如果解集中不存在合适的值，就向上回溯，以此类推。另一种方法就是模板法，顾名思义就是有一个现成的数独终盘模板，这个模板已经满足了数独的所有规则，在这个基础上再随机生成一个用 1~9 随机排列的数组，然后把数组中的元素插入到模板中对应的位置，便可以生成一个数独终盘了。

这两种方法，模板法的实现比回溯法简单得多，但回溯法生成的数独终盘数量却远多于模板法，模板法只能生成 9! 种数独终盘，而回溯法约有 6.67×10^{21} 种。

（2）求解用户完成的数独初盘。

人手求解数独的方法是基于人类的思维[1]，用到的策略一般都基于直观法和候选数法。而计算机求解数独，一般采用回溯法，一个初盘的解并不是指被挖洞后生成初盘的原始数独终盘，被挖洞后的初盘的解有可能不止一种，通过这个初盘或许能够解出几种不用的终盘。



本文基于两种数独终盘生成的算法实现数独游戏，并在回溯法中使用有限递推来减少循环次数。

3 生成数独终盘

3.1 模板法

该方法要先定义一个模板数组 model，在该二维数组里初始化 9*9 个满足数独规则的字母——每行、每列以及每个 9 宫格都无重复字母；其次定义一个一维数组 temp，用来保存 9 个随机数——这 9 个随机数在 1~9 的范围内且不重复；最后通过判断 model 数组里的字母，使用 temp 数组的元素向该字母对应着的 model 元素重新进行赋值，最终生成一个合格的数独终盘。因为 temp 数组里的元素是用 1~9 之间的数字随机排列的，共有 9! 种不重复排列，所以这种方法能生成 9! 种终盘。

```
for (let i = 0; i < 9; i++) {
        for (let j = 0; j < 9; j++) {
            if (model[i][j] == ‹a›) model[i]
```

```
[j] = temp[0];
                        else if (model[i]
[j] == ‹b›) model[i][j] = temp[1];
                        else if (model[i]
[j] == ‹c›) model[i][j] = temp[2];
                        else if (model[i]
[j] == ‹d›) model[i][j] = temp[3];
                        else if (model[i]
[j] == ‹e›) model[i][j] = temp[4];
                        else if (model[i]
[j] == ‹f›) model[i][j] = temp[5];
                        else if (model[i]
[j] == ‹g›) model[i][j] = temp[6];
                        else if (model[i]
[j] == ‹h›) model[i][j] = temp[7];
                        else if (model[i]
[j] == ‹i›) model[i][j] = temp[8];
            }
    }
```

3.2 回溯法

关于回溯法，本文在采用递归实现的回溯法时，还利用了“有限递推”的技巧以缩短回溯法的时间[4]。

基于有限递推的随机终盘算法逻辑如下：

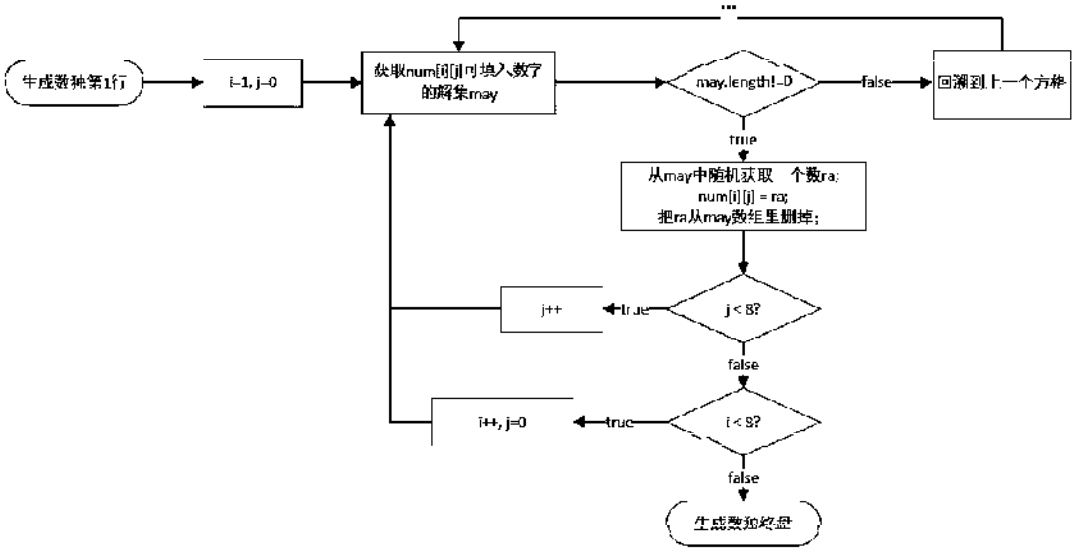
（1）按顺序从上到下、从左到右确定方格的值。因为是按顺序生成数独终盘，所以第一行受约束的条件最少，只需保证同行无重复值，所以这里先生成第一行 1~9 之间不重复的随机排列数字存入二维数组 num[0][0]~num[0][8] 中。

（2）从第 2 行第 1 列开始获取方格可填数字的一维数组解集 may。

（3）若 may.length!=0，即该方格存在可填入的数字，就从解集中随机选取一个数字填入方格，再从解集 may 里删掉该数字。然后继续步骤 2。

（4）若 may.length==0，即该方格不存在可填入的数字，则向上回溯。

按以上步骤，递归到最后一个方格并且其可填数字解集不为空，那么一个数独终盘就实现了。



因为是按从上到下、从左到右的顺序确定数独的值，所以在获取方格 num[i][j] 可填数字时只需判断该方格列的上方和行的左方。

4 挖洞

数独终盘有了，接下来就是生成初盘了，这里采用的是挖洞的方法，其实也就是随机覆盖掉一部分方格让用户进行输入。

首先要将数独终盘插入到网页中，文本通过 JS 向 table 标签中插入。

然后随机覆盖掉一部分方格。先定义一个随机数 r，表示本轮被覆盖掉的方格数量，也就是留给用户输入的空白格数量，这里随机数的范围要保证标准数独的最少提示数为 17 个；再定义一个随机数 ra，表示随机覆盖下标为 ra 的单元格，这里有 r 个空白格，就要生成 r 个不重复的下标值 ra；这里使用 input 标签进行覆盖。

```
    for (let i = 0; i < r; i++) {
      let input = document.
createElement(“input”);
        inputsetAttribute(“maxlength”, “1”);
        td[ra].textContent = “”;
        td[ra].appendChild(input);
    // 这里记得设置 input 的 css 样式，否则界面显示的时候会变形
    }
```

5 求解

挖洞完成后，数独初盘就出来了，但是还得完善一些细节：判断用户是否输入 1~9 之间的数字；空白格皆被填满则开始校验用户完成结果是否正确。

首先向所有的 input 标签添加一个 onchange 事件，当用户输入一个值，或是用户输入的值改变时就触发该事件进行一些自我设定的判断。可以直接在挖洞创建

input 标签的时候添加的事件，当然也可以通过循环来添加，知道 input 标签数量为 r，就能够通过循环依次给 r 个 input 标签绑定事件。

这里求解数独用的也是回溯法，与生成数独终盘类似，但又比它简单。当用户填写完最后一个空白格且填写数字在 1~9 之间，则从第 1 行第 1 列的数值开始判断该方格在同行、同列以及 9 宫格是否出现重复值，出现重复值则答题错误；否则继续向下判断，以此类推，直到最后一个方格也无出现重复值，则返回答题正确。

6 结论与建议

数独实现的难点就在于出题和解题，出题在于数独终盘的生成，这里用的是模板法和回溯法两种方法，但肯定还有很多其他更好更简的方法；而用回溯法解题是为了在同一初盘可能有不止一种答案的情况下，保证解题的多样性和灵活性。

本文在数独的实现上，还可以做进一步的改进，增加并完善一些功能，提高玩家的体验感，譬如：

（1）难度模式：提示数介于 17 到 27 之间属于困难，即 17 <=（81-r）<=27；介于 28 到 37 之间属于中等，即 28 <=（81-r）<=37；更多就是简单了。难度模式可以通过调整挖洞时的随机数 r 来进行选择，当然这里不是指人为的修改 JS 代码，可以通过 DOM 操作获取用户所选择的难度，然后对其进行一些判断，动态修改随机数 r，这样就可以实现难度模式的选择。

（2）检查模式：本文实现的方法只能在所有待填空白格都填满时，才会检查用户完成的结果正确与否，可以说是答题结束后检查。但其实可以添加另一种检查模式——答题过程中检查，在用户填写时，后台同步判断用户填写的值正确与否，方法是判断用户输入的值是否在同行、同列以及 9 宫格出现了重复值，重复就弹出提示框提醒用户及时修改，而这里的实现方法肯定也不止一种。在答题过程中检查比起在答题结束后检查，大大节省了玩家的时间以及精力。当然或许不同的玩家有不同的选择，所以可以设置为玩家自己选择检查的模式。

（3）提示模式：在玩家筋疲力尽之后，可以选择此模块来提示方格可填入数字的解集，以供玩家参考便于继续解答数独。

（4）重玩模式：恢复初盘，而不是重新创建数独初盘。

最后，关于这次数独的实现，在过程中收获了不少，运用了课堂学到的知识，也掌握了一些以前从未注意过的概念，更重要的是了解了更多其他方面的知识，比如数独的来源、数独的玩法、模板法以及回溯法等等。实践是检验真理的唯一标准，在实践中摸索，也在实践中学习，知道自己的不足与盲区，再逐渐的自我补足与进步。

【参考文献】

[1] 曲海平，岳峻，王飞．数独问题的生成与求解算法的研究 [J]．科技通报，2017,33(06):14-17.

[2] 石少俭．搜索算法问题的研究 [J]．电脑知识与技术，2020,16(23):216-217.

[3] 石铭杰，刘警灿，黄建昌．回溯法的概述与应用 [J]．计算机产品与流通，2020(06):243.

[4] 雷蕾，沈富可．关于数独问题的算法的设计与实现 [J]．电脑知识与技术（学术交流），2007(02):481-482+523.