Sudoku Game Report

计 64　陶东来　学号:2016011322

September 4, 2017

1　概述

本次大作业，我本来想使用传统的 Qt 来完成。但我在实现过程中发现，我难以简洁地达成我想要的效果。并且，Qt 的 QSS 并不像 CSS 那样有方便的工具（jQuery）来辅助处理。因此我选择转换思路，把工程分解成了由 html+css+js 管理的前端，和用 Qt 管理的后端，通过 QWebEngine 和 QWebChannel 来进行前后端的交互。

必须说明的是，本次所实现的版本仅仅是一个“原型”。更多的视觉效果，如动画，可以在前端中较为方便地扩展出来（使用 keyframe），比起单纯的 Qt 要简单得多。这也是我使用这种设计的重要原因之一。

2　前端

前端使用了 jQuery 和 bootstrap 来辅助实现，通过 js 来管理游戏的整体运行时逻辑。运行效果如图 1：

可以调节难度，在难度为 10 时，效果如图 2：

可以看到，支持了相同数字高亮，标记格子，提示当前行列的功能。而一行中填入多个数字，计时，撤销，恢复，删除，重新开始，这些功能这些功能也不在话下。

最后，是一张游戏胜利的截图。

Figure 1: 图 1. 难度为 1

Figure 2: 图 2. 难度为 10

Figure 3: 图 3. 一行中填入多个数字

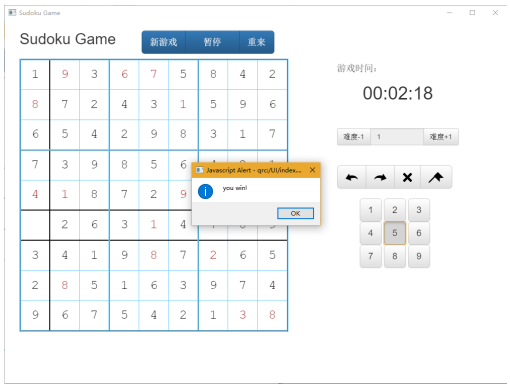

Figure 4: 图 4. 胜利！

3 后端

后端的 Qt 主要负责数独盘面的产生，并通过 QWebChannel 传输至前端。并且，通过 Qt 后端，游戏进度的保存也成为可能。

4 数独的求解及生成

数独的求解使用了经典的 DLX(Dancing Link X) 算法。数独的生成则参考了同学交流时所提到的 Las Vegas+ 贪心挖洞算法。而难度则直观地表现在盘面空格的个数上。难度大于等于 x 时，盘面上的空格数应至少为 $10+5x$ 个。

4.1 数独的求解——DLX

DLX 算法是由 Donald Knuth 提出的，用以高效解决精确覆盖问题的搜索算法。所谓精确覆盖问题的一个表述是，给定一个 01 矩阵，求得一个行的集合，使得每列有且仅有一个 1，或报告无解。

首先，我们不难想到一个深度优先搜索算法（被 Knuth 称为算法 X）。从第一列开始，对于第 i 列，我们依次考虑当下可以用来覆盖这一列的行，并把之后将与之冲突的所有行排除在之后的考虑范围之内，接着考虑第 i+1 列。为了高效地实现算法 X，我们可以使用二维的循环双向链表来存储这个 01 矩阵，这个链表就被称作 Dancing Links。通过 Dancing Links，我们可以高效地删除/恢复行列，并且更好地利用稀疏矩阵的特性，从而获得极佳的速度。

DLX 的具体实现这里不进行细致的描述。我们主要要谈的是如何将数独转化为精确覆盖问题。我们构造一个 $9\times9\times9$ 行，$4\times9\times9$ 列的矩阵。行表示在格子 (i,j) 中填入 k，而列表示其对应的 4 条限制：

1. (i,j) 有且仅有一个数字。
2. 数独的第 i 行有且仅有一个数字 k。
3. 第 j 行有且仅有一个数字 k。
4. (i,j) 所在的 3×3 大方格有且仅有一个数字 k。

转化完毕之后，直接使用 DLX 算法求解即可。

4.2 数独的生成——Las Vegas+ 贪心挖洞

数独的生成分为两步：1. 生成一个合法的终盘；2. 在终盘上挖出一些空格。

4.2.1 生成合法终盘——Las Vegas

一个简单易行的方法，是在全空数独上随机地选取 MN 个格子，随机填入 1-9 的数字，在调用数独求解器得到终盘，如果无解则重新随机。这里根据实践，MN 取 11。

4.2.2 挖洞——贪心法

在获得了一个合法的终盘之后，我们根据难度选择对应的遍历方法 g_i，并对遍历过程中的每个格子，尝试将其变为空格。如果在变为空格之后盘面有唯一解，就保留这个空格，否则将其恢复为原来的数字。重复这个过程知道空格数达到难度要求为止。

```
g[level-1](bd,[](int *bd,int pos){
                int cur=bd[pos];
                for(int i=1;i<=N;i++) if(cur!=i){
                    bd[pos]=i;
                    if(sudokuSolver(bd,nullptr)){
                        bd[pos]=cur;return false;
                    }
                }
                bd[pos]=0;return true;
        });
```

Figure 5: 图 5. 实现

具体实现时，考虑到不同的遍历方法 g_i，实际上只是遍历顺序有区别，而对于遍历的每个格子实际上做的是同样的事情，我使用了 FP 的思想，让 g_i 接受一个 function 作为参数，这个 function 执行了对于单个格子进行的操作，如图 5。

5 值得加入的改进

通过加入动画让 UI 更为炫酷；
保存最佳成绩；
等等。

6 吐槽

等等，还有吐槽环节吗？第二周怎么还是 Qt？真是个灾难！

话说回来，我这次的 UI 是基于高二的一个（尚未完成的）project 的（半成品）web_ui，现在还挂在 github 上面（https://github.com/NagiNikaido/Cell-Game）。真是久违了啊，在我想起它的时候，有一种“啊，兜兜转转又回到了这里”的感觉。我刚开始写这个 project 的时候，想着能不能搞个 arena，让 AI 无监督自进化。再后来为了可视化，参考当时正火的 2048 开始用 bootstrap 来做 web_ui，服务器还打算用 node.js 来写。再后来，就因为种种原因被搁置起来了。

或许什么时候我又会把它拿出来，精心擦拭一番，换上新的零部件，加上全新的扩展——特修斯之船。

不过，那应该也是以后的事情了吧。