数据结构课程报告

软件工程1504 刘博谦 U201517070

一、开发目标

使用Html/CSS/Javascript编写数独游戏。能按照不同难度级别设置随机的初始数独，允许用户在空格处填入数字完成数独游戏；在用户输入数字时，判断冲突，给出错误提示；计时，记录用户完成数独所用时间。

二、开发计划

使用jQuery以及其timer、blockUI插件完成数独游戏的设计与编写。首先进行数独算法的编写，然后编写基本的html页面，如果还有时间的话就使用CSS把页面再美化一下。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模块名称 | 主要文件 | 预计完工 | 实际完工 | 完成情况 |
| 数独算法 | sudoku.js | 12.28 | 12.30 | 良好 |
| 网页 | index.html | 12.29 | 01.01 | 良好 |
| 层叠样式表 | style.css | 12.30 | 01.01 | 良好 |

三、测试方案

1、测试不同浏览器下功能模块与页面显示情况，并与预期效果比较并做出改进

2、测试不同难度数独算法执行情况，包括速度、错误率等

四、软件设计（用方框图表示软件结构，用流程图或伪代码表示算法）

软件结构

网页项目Sudoku——回溯算法实现数独游戏

层叠样式表

文件：style.css

功能：实现网页的样式设置

Javascript文件

文件：sudoku.js

功能：实现核心数独算法与主要函数

(插件js文件详见项目)

网页

文件：index.html

功能：实现网页的样式设置并控制网页流程

下面给出乱序排列、相关20格判定与回溯解数独伪代码。

shuffle(v)

{

i←the length of v

**for** (i←theLengthOfV;v>**0**;v←v-**1**)

j←小于i的数字

swap v[i] and v[j]

end

**return**

}

check20Grid(ss,i,j)

{

//辅助变量存储

let row = {},col = {},subSudo = {}

**for** (k←**0**; k < **9**; k←k+**1**)

{

cur1←ss[i][j],cur2←ss[k][j]

**if** (ss[i][j]已经在第i行出现)

**return** **1**;

**else**

storage cur1 in row

**if** (ss[k][j]已经在第j列出现)

**return** **2**;

**else**

storage cur2 in col

**if** (subSudo[key]已经在小九宫格出现)

**return** **3**

**else**

storage key in subSudo

}

**return** **0**;//正常

}

fill()

{

**for** (let i ← **0**; i < **9**; i++)

{

**for** (let j ← **0**; j < **9**;j++) {

**if** (当前位置为待定元素的首次处理或回溯到当前位置)

{

搜索向下一个合法值迈进

**while** (k < **10**) {

填值

**if** (相关二十格判定判定合法)

{

存储回溯点，并步进

**break**;

}

k++;

}

**if** (当前位置找不到合法值，回溯)

{

回溯前归零

堆栈中取回溯信息

**if** (无解)

**return** **0**;

穿越

}

} **else** **if**(当前位置数字为题目给定)

j++;

}

}

**return** **1**; //成功

}

五、软件实现（给出关键函数的代码清单）

初始数独的生成，首先填好第一行代码，1~9随机排列（此步是为了防止填入1~9顺序排列的情况以提高难度），随后利用回溯算法解此数独，1~9从第二行开始填入空格不断尝试，成功继续，否则回退重新寻找满足条件的数独。在填入数字时，主要利用数独的性质，即每行、每列、每个九宫格只含有1~9且不重复。找到一个解后根据难度的不同，挖去相应数目的格子数，随后打印表格开始游戏。

下面列出sukudo.js数独算法和错误检测的代码及注释。

//生成初始数独

**function** InitData() {

**for** (i = **0**; i < **9**; i += **1**) {

bl[i] = **new** Array(**9**);

tn[i] = **new** Array(**2**);

s[i] = **new** Array(**9**);

numcount[i] = **9**;

}

**for** (j = **0**; j < **9**; j += **1**) {

**for** (k = **0**; k < **9**; k += **1**) {

bl[j][k] = **0**;

s[j][k] = **0**

}

s[**0**][j] = j + **1**;

}

shuffle(s[**0**]);

fill();

/\*var Alert="";

for(i=0;i<9;i++)

{

for(k=0;k<9;k++)

{

Alert+=s[i][k];

Alert+=" ";

}

Alert+="\n";

}

alert(Alert);\*/

}

**var** stack = [],

flag = **false**;

//判定填入元素是否重复

**function** check20Grid(ss, i, j) {

**let** row = {},

col = {},

subSudo = {} //辅助变量

**for** (**let** k = **0**; k < **9**; k++) {

**let** cur1 = ss[i][k],

cur2 = ss[k][j]

**if** (cur1)

{

//当前元素已经在行中出现，优化掉零的判断，key为0时值为0，不需要额外判断

**if** (row[cur1])

**return** **1**; //返回错误代码

**else**

row[cur1] = cur1 //当前元素未在行中出现，存入辅助变量中

}

**if** (cur2)

{ //列的判定与行类似

**if** (col[cur2])

**return** **2**;

**else**

col[cur2] = cur2;

}

//转化循环变量到小九宫格的坐标

**let** key = ss

[Math.floor(i / **3**) \* **3** + Math.floor(k / **3**)]

[Math.floor(j / **3**) \* **3** + Math.floor(k % **3**)]

**if** (subSudo[key]) //九宫格判定与行类似

**return** **3**

**else**

subSudo[key] = key

}

**return** **0**;

}

//回溯算法寻找数独可行解

**function** fill()

{

**for** (**let** i = **0**; i < **9**; i++)

{

**for** (**let** j = **0**; j < **9**;) {

**if** (s[i][j] === **0** || flag)

{

//当前位置为待定元素的首次处理或回溯到当前位置

flag = **false**;

**let** k = s[i][j] + **1**; //搜索向下一个合法值迈进

**while** (k < **10**) { //循环找到下一个合法值

s[i][j] = k; //填值

**if** (check20Grid(s, i, j) == **0**)

{ //相关二十格判定判定合法

stack.push([i, j++]) //存储回溯点，并步进

**break**;

}

k++;

}

**if** (k > **9**)

{ //当前位置找不到合法值，回溯

s[i][j] = **0**; //回溯前归零

**let** rt = stack.pop(); //堆栈中取回溯信息

**if** (!rt) //无解判断，返回0

**return** **0**;

i = rt[**0**] //穿越

j = rt[**1**]

flag = **true**;

}

} **else**//当前位置数字为题目给定

j++;

}

}

**return** **1**; //成功找到一组解

}

//挖洞策略初始化表格数据

**function** InitBlockData(level) {

**var** lt = **20** + level + Math.floor(Math.random() \* **3**);

**var** j = Math.floor(Math.random() \* **9**);

**var** k = Math.floor(Math.random() \* **9**);

**if** (level == **0**)

blockscount = **0**;

**else**

**for** (i = **0**; i < lt; i++) {

**while** (**true**) {

j = Math.floor(Math.random() \* **9**);

k = Math.floor(Math.random() \* **9**);

**if** (j == **4** && k == **4**) {

**if** ((numcount[s[j][k] - **1**] == (**6** - level) ||numcount[s[Math.abs(j - **8**)][Math.abs(k - **8**)] - **1**]

== (**6** - level)))

**continue**;

**else**

**break**;

} **else** **if** (numcount[s[j][k] - **1**] == (**5** - level) ||numcount[s[Math.abs(j - **8**)][Math.abs(k - **8**)] - **1**]== (**5** - level))

**continue**;

**else**

**break**;

}

numcount[s[j][k] - **1**]--;

s[j][k] = **0**;

**if** (s[Math.abs(j - **8**)][Math.abs(k - **8**)] != **0**) {

numcount[s[Math.abs(j - **8**)][Math.abs(k - **8**)] - **1**]--;

}

s[Math.abs(j - **8**)][Math.abs(k - **8**)] = **0**;

}

blockscount = **0**;

//计算空格数量

**for** (j = **0**; j < **9**; j++)

**for** (k = **0**; k < **9**; k++)

**if** (s[j][k] == **0**)

blockscount++;

}

//数独数据检查

**function** check(o) {

**var** id = o.attr('id'),

x = id.substr(**2**, **1**),

y = id.substr(**4**, **1**),

val = o.num(),

tx = Math.floor(x / **3**),

ty = Math.floor(y / **3**);

**for** (i = **0**; i < **3**; i++) {

**for** (j = **0**; j < **3**; j++) {

tn[i \* **3** + j][**0**] = tx \* **3** + i;

tn[i \* **3** + j][**1**] = ty \* **3** + j;

}

}

**for** (i = **0**; i < **9**; i++) {

**if** (x != i) {

**if** (s[i][y] == val && val != **0**) {

bl[i][y]++;

bl[x][y]++;

} **else** **if** (bl[i][y] > **0** && lastselectednum == s[i][y]) bl[i][y]--;

}

**if** (y != i) {

**if** (s[x][i] == val && val != **0**) {

bl[x][i]++;

bl[x][y]++;

} **else** **if** (bl[x][i] > **0** && lastselectednum == s[x][i]) bl[x][i]--;

}

}

**for** (i = **0**; i < tn.length; i++) {

**if** (tn[i][**0**] != x && tn[i][**1**] != y) {

**if** (s[tn[i][**0**]][tn[i][**1**]] == val && val != **0**) {

bl[tn[i][**0**]][tn[i][**1**]]++;

bl[x][y]++;

} **else** **if** (bl[tn[i][**0**]][tn[i][**1**]] > **0** && lastselectednum == s[tn[i][**0**]][tn[i][**1**]]) bl[tn[i][**0**]][tn[i][**1**]]--;

}

}

//重复数据红色提示

**for** (j = **0**; j < **9**; j++) {

**for** (k = **0**; k < **9**; k++) {

**if** (bl[j][k] > **0**) $("#c\_" + j + "\_" + k + "").addClass('err');

**else** $("#c\_" + j + "\_" + k + "").removeClass('err');

}

}

//填完空格且填入数据全部正确

//计时器停止，

**if** ($('.err').size() == **0** && blockscount == **0**) {

$(g.canvas).unblock().block({

message: finishedinfo,

css: {

width: '400px',

border: '3px solid #a00'

}

});

$('#timer').stopTime();

}

//更新待填方格数

$('#lefts').text('待填方格数:' + blockscount + '');

$('#leftsg').refreshnumcount();

}

六、测试结果与分析

在对半成品项目进行测试的时候，数独的生成、挖格子和打印表格在主观感觉上还是很快的，不存在延迟。简单难度连续生成多个数独后网页会崩溃，增加格子后故障消失。

在主流浏览器如Edge、Firefox和Chrome浏览器基本功能运行良好，不过Firefox和Chrome中页面布局发生了部分扭曲，可能会影响玩家视觉体验，Edge浏览器运行效果最好。

七、总结

这次课程设计使我对数据结构的编程思想以及回溯算法有了进一步的了解，对Html/CSS/Javascript都有了一定的了解，并切身的体会了一把编写网页的感觉。因为对软件结构图和伪代码不是很了解，只是参照例子大概编写了一下，可能不是特别规范。