Game of Life (Cellular Automata 细胞自动机)

说明文档

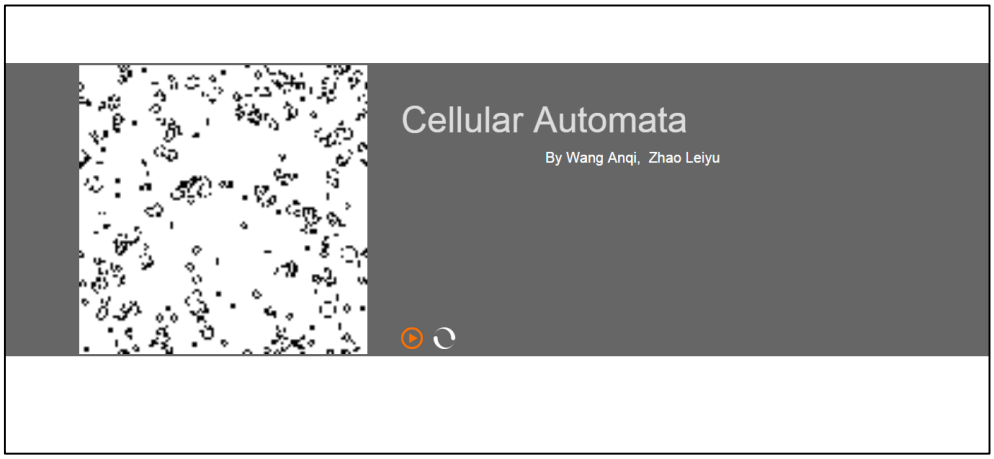
软 21. 王安琪 2012013282

软 21. 赵雷彧 2012013285

I. 实现情况

本实验目标为利用 Javascript 实现名为 Game of Life 的细胞自动机，将其在浏览器上展示并对源代码(尤其是内核部分)进行单元测试。

经过开发，程序支持在支持 HTML5-Canvas 标签的浏览器上展示细胞自动机运行状况并将其渲染至前端。界面如下:



左方展示板中黑白两色分别表示活、死细胞，右下方按钮分别控制自动机的状态演进的暂停行进以及重新随机细胞分布。

在源代码 Kernel.js 中的常量定义类中，可以轻易调整一切预制参数(截图如下)，如细胞高度、宽度、每秒刷新帧率、刷新活细胞几率等。

```
var GLOBAL_CONST=   //a class contains all the global storage used
{
    WIDTH: 120,     //the width of the arena
    HEIGHT: 120,    //the height of the arena
    NEIGHBOR: [[-1,-1],[-1,0],[-1,1],[0,1],[1,1],[1,0],[1,-1],[0,-1]],
    FPS: 30,    //the fps to refresh the arena
    CANVAS_SIZE: 400,   //canvas size is x*x
    PROBLEBILITY: 0.2,  //the probability to spawn a living cell
    ARENA: {},  //global arena to hold those cells
    TIMER: -1,   //the refresher-function timer
    CANVAS: 0
}
```

由于程序速度瓶颈在于 Canvas 渲染效率而非细胞矩阵处理效率，故重点优化 Canvas 绘制命令数，通过建立 Arena.Changed 指示量每次只重绘状态更改的细胞对应的画布区域，可较大幅度优化细胞自动机的最大执行能力。

	优化前	优化后
FPS=30 时最大渲染尺寸	110*110	240*240

注： 1. 以上测试均在 Chrome 37 下进行，CPU: Intel Core i7-4500U。

2. 受限于 Canvas 尺寸，为达到相对较好的展示效果，细胞阵列规模最终确定在 120*120。

本程序在 Chrome，IE11 上均能正常运行。

II. 部署情况

根据要求，本程序全部源代码部署于 Github 上。仓库网址为

https://github.com/publicres/cells

SSH 克隆地址为

git@github.com:publicres/cells.git

程序源码以及对于内核的单元测试代码分别位于 master、unitTest 两大分支，可分别拉取并本机运行。对于正常版本程序，运行 main.html 即可；而对于单元测试版程序，配置及查看结果方式参见下一章节。建立的 gh-pages 分支用于发布网页，发布结果可直接由网址 http://publicres.github.io/cells/ 查看。

III. 单元测试运行方法

根据要求，单元测试全部源代码部署于 Github 上。见“部署情况”。

本测试程序采用了 Qunit 测试框架，最小测试单元为函数，仅对核心部分 kernel.js 中定义的函数进行了单元测试，而对于界面代码进行了走读测试。

单元测试运行方法，直接运行位于 unitTest 文件夹下的 test.html 即可看到测试结果。

IV. 分工情况

王安琪	**赵雷彧**
单元测试	内核、前端编写
辅助编程	

V. 感受 & 总结

王安琪：

首先，结对编程是一种高效的编程方式。同一段代码由双人监督完成，有一些小 bug 在走读阶段就能被发现，而在编译运行时发现的 bug 也能在短时间内就得

到解决。而且由于同伴的监督，使得双方都集中精神，提高了工作效率。

第二，单元测试更便捷。进行单元测试时，因为有了协作编程时沟通的基础，我已经对雷彧兄的代码很了解了，于是也就很快知道了如何设置测试数据。

第三，单元测试取决于函数功能的划分。雷彧兄将函数功能划分的很明确，让写测试的时候思路清晰了许多。

第四，交流学习。这点只是针对我个人而言的，以前自己单打独斗的时候，遇到了问题也会去问其他同学，得到的只是零星的碎片。但是这次协作编程，通过与雷彧兄的交流以及看他 coding 的过程，说是大开眼界也不为过了。

赵雷彧：

由于我原来参加过信息竞赛，协作编程的确是和之前单人想思路、coding 以及 debugging 完全不同的经历。

在结对编程中，从程序逻辑框架到具体算法的实现均由双人讨论而成，即使是个人突发奇想的点子也需要务必给搭档讲清楚，否则在实现过程中势必招致不必要的质疑。另外，由于每个想法都经过两个人的大脑，其理论正确性可以被再次确保，如果有潜在的问题也必然有很大的几率被发现，从而大幅降低写好程序发现不可行而返工的事故出现率。另外，由于编辑代码的时候有人作为监督者，同时编写者需要把代码的意图同步给监督者，对编写者本身也是一个警醒，将大幅度降低程序出错概率。例如在和@王安琪编程的过程中，由于经历一个假期，我本身对 js 已经出现生疏，笔误亦或是小错误时有发生，可以预见到将在调试阶段带来相当多麻烦。而作为监督者的安琪在旁边上网查阅求证后能很快指出问题，也省略了相当多的调试代价。

就我个人而言，对于高密度，高出错率的 coding 工作，结对编程带来的沟通代价效率节省的调试时间，故值得采用；相反，对于简单的、编程者相当有把握的工作，则单人编程+静态查错足以顺利完成程序。

而对于单元测试，个人认为最重要的环节在于编程阶段合理的函数分割以及测试数据的合理编写，测试框架只是辅助性工作，能够真正全面了解代码以及编写出方便分析的代码才是测试是否有作用的关键。故今后编程的逻辑接口划分需要朝着更易测试的方向发展。