Game of Life 说明文档

彭友 李映辉

2014-9-30

目录

一、Game of Life 项目的实现

Game of Life 项目的实现位于根目录下的 LifeOfGame.js 文件和 index.html 文件中。

Game of Life 项目的实现分为三个部分：逻辑功能、界面绘制以及游戏控制。

逻辑功能：

逻辑功能由四个函数实现：generateCellRandom()，generateCell()，countNeighbors()以及 nextRound()。

generateCellRandom()函数的功能是随机地赋予每个细胞生或者死的状态（0 代表死，1 代表生）

generateCell()函数的功能是把每一个细胞的状态设为死

countNeighbors()函数的功能是针对指定位置的细胞，计算出当前状态下该细胞周围活细胞的数量

nextRound()函数的功能是将所有细胞的状态更新为下一时刻的状态

界面绘制：

界面由 drawGrid()函数进行绘制，每一个活细胞被绘制成一个红色的点，每一个死细胞被绘制成浅棕色的点

游戏控制：

游戏控制功能由 keyDown()函数实现。具体的功能请参见 README。

二、部署情况

部署地址: http://pengyou12.github.io/GameOfLife

部署过程：

先把之前的 github 的库 clone 到本地，然后使用 sourcetree 的命令行模式用以下命令配置 gh-pages 分支:

git branch gh-pages

git checkout gh-pages

然后手动新建一个 GameOfLife 文件夹，并且把最后所有的文件都拷贝到下面

然后到命令行里面使用

git add .

git commit –m “commit the game”

git push orgin gh-pages

来提交整个文件夹

遇到问题

Q1：刚提交时网页打不开

A1：push 之后大概要 10 分钟之后才会有显示

Q2：sourcetree 下不会切换分支？

A2：打开命令行模式，使用 checkou 命令切换

三、单元测试运行方法

测试源代码位于根目录下的 test.js 文件中，在浏览器中打开 test.html 将会自动运行程序。

在本次项目的单元测试只针对逻辑功能部分进行了测试，对界面绘制以及游戏控制部分用走查的方式进行了检查。

在逻辑功能的测试过程中我们没有使用测试框架，而是用 alert 的方式判断函数是否通过了测试样例。由于 generateCellRandom()函数和 gerateCell()函数中没有逻辑运算，所以我们只测试了 countNeighbors()函数和 nextRound()函数。在测试中我们设计了 10 个测试样例，其中前 9 个是边界测试，第 10 个是一般性测试。同时，我们还针对一般情况（行数、列数均不小于 3）的情况用随机化的方法对上述两个函数进行了进一步验证。对各个测试的描述如下：

test1 和 test2 测试的是只有一个细胞的情况。其中 test1 测试的是细胞为活的情况，test2 测试的是细胞为死的情况。

test3 测试的是细胞为 1 行 2 列的情况，test4 测试的是细胞为 2 行 1 列的情况。

test5，6，7，8，9 分别测试了细胞排列为 2 行 2 列，1 行 3 列，3 行 1 列，3 行 2 列和 2 行 3 列的情况。

test10 测试的是一般情况下的逻辑测试，细胞排列为 3 行 3 列。

test11 中我们首先构造了一个对比函数，这个函数用另外一种方式计算了在特定细胞周围的活细胞数目。并且我们可以确定，在细胞的行数、列数均大于 2 的情况下是正确的。我们利用这个函数，用随机化的方法进行测试。测试函数将随机生成细胞的行数和列数（均不小于 3），并随机初始化每个细胞的生死状态。然后利用对比函数判断我们之前写的函数是否正确。在 test11 中前述过程会被重复 1000 次，相当于对一般情况进行了 1000 次验证。

最终程序通过了全部 11 个测试。

四、分工情况

本次项目中我们使用了结对编程的编程方法，在编程的过程中不管是负责写代码的人还是负责在一旁审核的人对程序都有贡献，所以我们认为本次项目中的每一行程序都是有我们两个人共同编写的。具体编程时，项目的主体部分由彭友负责编写，李映辉负责审核；测试部分由李映辉负责编写，彭友负责审核。我们也都曾少量地参与到对方的编写过程中。

五、结对编程的感受：

李映辉：

在结对编程的过程中，我感觉到编程工作没有过去那么乏味。有队友在身边，我可以更加专注于程序的编写，而过去我可能写上 1 个半小时的程序以后就无法集中精力了。同时，结对编程帮助我们减少了很多 bug，一些拼写的错误以及逻辑上的缺陷被及时地避免了，减少了很多由于修改错误的时间。而且，在结对编程的过程中，我们都参与到了对方的工作中，这使得当有一方需要另一方协助他进行开发时，另一方不需要进行额外的学习就可以轻松地接管对方的工作，这在传统编程的时候是无法想象的（在过去，我非常讨厌替别人审查代码，因为大部分情况下我根本看不懂对方到底写了些什么）。总之，这次结对编程是一次非常愉快的经历，我喜欢结对编程。

彭友：

结对编程最大的感受就是在队友的注视下编程，思路会比一个人编程更加的清晰。而且一个很大的好处就是队友可以及时的指出自己写的一些 bug，这使得整个编程过程前所未有的流程，结对编程让你有编程牛人一样的 coding 体验。同时结对编程也是一次

很好的队友之间的磨合，对之后的大作业的团队合作有着莫大的帮助。结对编程过程中两个人可以互换角色，这使得两个人的总的工作强度下降了不少，但是工作效率却不比两人分开写要低。

六、单元测试之我见

先偷百度上面的一句话：“单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。”对于我们的程序而言，最小的测试单元就是一个函数。同时，并不是所有的函数都是可以通过设计测试样例的方法进行验证的，比如，对图形界面的绘制、控制操作等，就只能通过走查代码的方式检查或者在使用过程发现软件的缺陷。能够通过样例进行测试的函数主要是进行逻辑计算的函数。

在测试过程中，边界测试是很重要的。在实际的编程过程中，一般程序员脑子里想到的都是一般性的情况，所以一般情况下出错的概率是不大的，而错误往往会出现在边界情况下。对边界的测试要仔细，认真分析每一种特殊情况。

在测试中，覆盖所有情况的测试是理想的，但是巨大的工作量使得我们没有办法实现这种测试。所以，替代的方法是把输入空间分成若干等价类，在每个类中选取一组进行测试。我们这次测试就采取了这种方法。