说明文档

- 耿正霖、王宇炜

实现情况

我们按照作业要求实现了Game of Life的网页游戏。为了实现无限延伸的网格效果，我们首先创建了一个大小固定的网格，然后将上边界与下边界拼接，左边界与右边界拼接，这样，细胞可以再无限大的范围内演化。细胞状态的变化完全遵守实验要求中的生存定律。

玩家可以通过鼠标点击直接修改每个细胞的生死状态，也可以使用我们提供的按钮清空或者随机生成细胞。其中，随机生成细胞时细胞的密度是可以由玩家指定的。玩家可以点击start按钮让细胞开始连续的演化，演化速度也是可以控制的。玩家也可以点击next按钮进行一步演化查看效果。为了防止出错，在细胞演化的过程中，玩家不可以自行改变细胞状态。

部署情况

网页访问

http://gengzhenglin.github.io/Game%20of%20Life/src/index.html

代码部署

https://github.com/GengZhengLin/GengZhenglin.github.io/tree/gh-pages/Game%20of%20Life

单元测试部署

https://github.com/GengZhengLin/GengZhenglin.github.io/tree/gh-pages/Game%20of%20Life/test

单元测试的运行方法

测试方法

单元测试主要是对Grid.js中的Grid.next()函数进行测试，测试代码在`\test\test.js`中，运行单元测试的方法为：打开`\test\test.html`即会自动进行单元测试。

测试数据

打开`\test\testdoc\testData.md`可以查看自动生成的测试数据。（注：这个文件是事先生成好的）

测试覆盖率

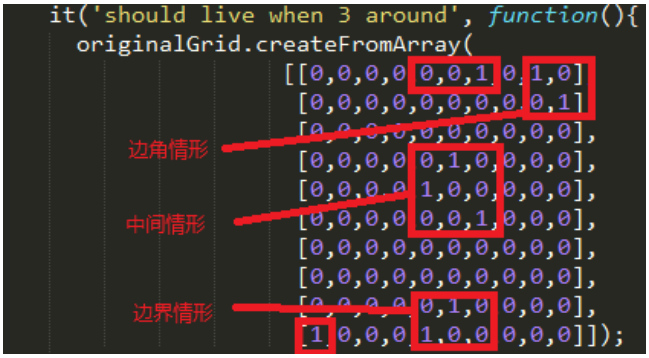
打开`\test\testdoc\testReport.html`可以查看测试过程中Grid.js代码的测试覆盖率。（注：这个文件是

事先生成好的）

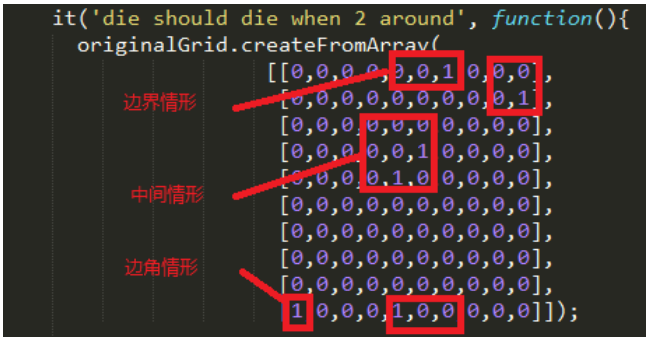
测试数据设计

我们把细胞单元的状态进行了分类。首先，按照其周围活细胞的数量进行分类，可分为2，3和其他三类。其次，按照细胞自己的状态，分为活和死两类。按照细胞的位置，分为网格内部，边上和角上三类。如此排列组合一共有十八类。因为棋盘足够大，我们可以在一次运行中利用网格的不同位置测试多组数据。

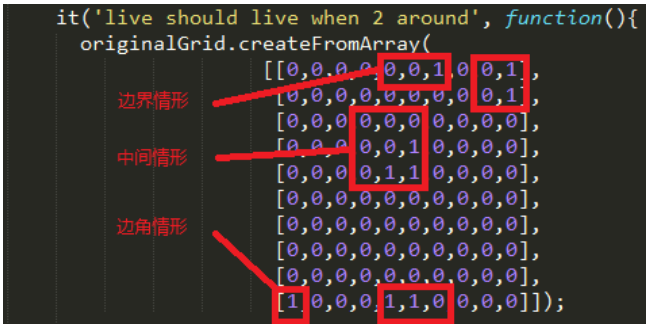
下图中，我们测试了细胞周围有三个活细胞时，细胞的状态变化。



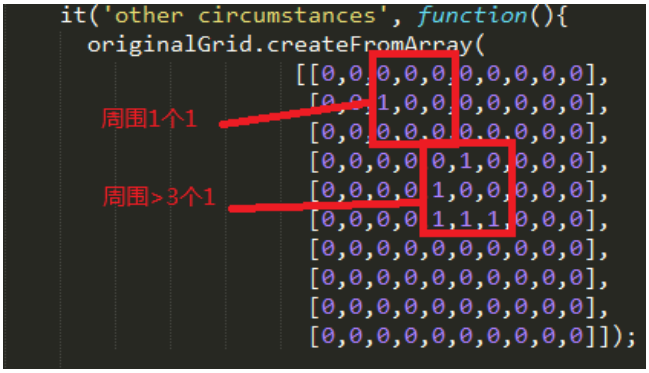
下图中，我们测试了细胞周围有两个活细胞且细胞本身死亡时，细胞的状态变化。



下图中，我们测试了细胞周围有两个活细胞且细胞本身生存时，细胞的状态变化。



下图中，我们测试了细胞周围有其他数目的活细胞时，细胞的状态变化。



分工情况

几乎所有代码文件为我们两位同学所共同完成，共同负责，但是如果一定要列出分工情况，就是这个样子的：

- 逻辑代码：
  - ‘驾驶员’：王宇炜
  - ‘领航员’：耿正霖
- 单元测试：
  - ‘驾驶员’：耿正霖
  - ‘领航员’：王宇炜
- 交互界面：王宇炜与耿正霖两名同学各自写了一部分。
- 文档编写：王宇炜与耿正霖两名同学各自写了一部分。

对结对编程的感受（王宇炜）

这次编程我们尝试了一下结对编程。在编写代码的时候，一个人写的同时，另外一个人在旁边看。在写逻辑部分的时候，结对编程确实避免了一些小错误的发生。例如，在写for语句的时候，有时候会写错边界条件，将小于等于写成小于。在结对编程的时候，队友会提出这一错误。否则的话，可能会用很长的时间才能定位这一错误。结对编程为我们节约了不少后期调试的时间。此外，在自己思路突然有点混乱的时候，队友也可以及时提醒，有助于更快地理顺思路，降低出错的可能性。和传统的编程方式相比，其带来的好处便是减少了后期调试的时间，提高了单人的工作效率和效果。

但是，由于结对编程需要两个人的参与，使得两人不能同时编写程序，成本是很大的。而且只要在写代码时思路清晰，并经常检查的话，也可以避免很多错误。对我自己来说，因为我习惯于在写完一行之后自己检查一遍，原本就可以查出不少错误。而且，对一些结构复杂的代码，我会在编写之前在纸上理顺思路。结对编程并不比自己写的效果好多少。至于写界面的部分，出现错误的可能性较小，更多的是对界面的呈现效果进行反复调整，不适合进行结对编程，我们在写界面的部分也是只由一个人编写。总体上说，我觉得自己并不是很适合结对编程。

对结对编程的感受（耿正霖）

编程效率

在与王宇炜编程的时候，我感到编程的效率有了比自己编程有了非常明显的提升。比如在设计边界处理

代码的时候，如果是一个人的话就会开始纠结，是应该用取模的方法还是用平移的方法，取模的方法编写容易但是效率比较低，平移的方法编写比较困难但是效率比较高，这里面有权衡取舍的过程。但是两个人一起编程的时候我们就可以进行讨论，从而用尽可能快的速度找到尽可能好的方法。我们在编程的时候进行了充分的讨论，感觉比自己编程效率高多了。

编程准确性

在结对编程的实践过程中，我感到编程效率有一个非常明显的提升。在王宇炜进行逻辑部分代码编写的时候，有时确实会发生一些自己很不容易发现的错误，比如有时候忘加‘this.’什么的，我在旁边看的时候总能够及时地指出来，并提出一些解决方案。如果没有我在旁边，这些很难发现的bug很可能就被编写者忽视，只有在运行调试的时候再重新把这些问题找出来，相比之下真的会浪费非常多的时间。

debug效率

除了编程准确性debug效率也高了许多。我们在运行的时候还是会遇到一些bug，在调试走查的时候，有时我能发现一些他没发现的问题，有时他能提出一些我没想到的方法。两双眼睛同时调试一份代码，确实比一个人调试效率高出不少。

缺点

我们也有一个突出的感受，就是结对编程也并不是适用于所有的编程情景。比如交互界面的编写。我们曾经尝试交互界面也采用结对编程的方法，但是与逻辑代码编写的情况不同，在交互界面编写的时候，往往另一个人也看不出有什么值得提出的bug，浪费了时间，而且另一方面，即使界面有问题，代码编写者在运行的时候也可以非常轻易地找出错误所在。在设计界面的时候，两个人反而没有一个人自己拿主意来的效率高。比如加不加padding，用什么颜色，这种问题两个人往往都表示，我不知道该怎么办，让另一个人拿主意，这样反而降低了效率。

对单元测试的理解

软件开发的地位

我认为测试是软件开发中，保证软件质量不可缺省的一部分。虽然通过测试不能保证软件的完全正确，但是通过不断测试——修改这种方法可以不断提高软件的质量。

全面性

首先我认为为了保证单元的可靠性，单元测试应该尽可能覆盖到每一个条件分支，覆盖到代码的每一行。虽然覆盖到的地方并不能保证就一定不存在bug了，但是没有覆盖到的地方更不能保证没有bug。所以单元测试要尽可能做到全面。

等价类测试

如果对所有可能发生的情况都进行测试的话那么需要测试的情况就太多，根本不可能测完。所以设计测试数据的时候要注意，根据代码的实现情况，将所有可能的输入数据划分为等价类（这里引入了一个集合论的概念），然后从每个等价类中找一个代表测试数据进行测试就可以了。

测试成本

虽然按照等价类测试可以减少测试数据，但是随着条件分值的增加，情况的复杂，有时需要测试的等价类数目也依然会迅速增长。所以我认为有时在设计测试数据也不得不考虑在全面性与成本之间做出均衡，考虑减少一些非常明显一般不会出错的数据，而对一些边界数据，或是一些容易出错的地方，需要保证测试的全面性。

测试驱动开发

在进行测试的时候我突然在想，既然编写单元之后要进行单元测试，那么可不可以根据单元测试数据驱动代码的开发呢？后来看了老师上传的课件，竟然真的发现确实有这么一种模式。虽然我们这次并没有采用这样的方法，但是我觉得今后可以在开发的过程中进行尝试。