**生命游戏代码详解**

目录

**一．应用背景及国内外发展** 2

**二．相关技术简介** 2

**三．分析和概要设计** 2

**四．详细设计**2

**A.中期答辩时的代码详解** 2

**B.下面是中期答辩之后的工作** 3

**版本一 001MyLifeOpenCV.py** 3

**版本三 003MyLifeOpenCVMatConv.py**4

**版本四 004MyLifeColor.py** 4

**版本五 005MyLifeWithoutFor.py** 5

**版本六 006MyLifeWithoutForWithColor.py** 5

**其他**5

**五．测试(如何使用我的代码)** 6

**六．总结与展望** 7

**一．应用背景及国内外发展**

生命游戏这个名词已经出现很久了，好像由此发展出来元胞自动机的学科，而且python还有一个用来构建元胞自动机的包，可见生命游戏不仅有趣还很有用。

**二．相关技术简介**

中期答辩使用的是pygame，期末答辩使用的是opencv-cpu。二者都是很好的库，因为期末临近我四门实验课结束，而实验课正好被迫熟练掌握opencv，所以换成opencv来实现生命游戏代价不高。

**三．分析和概要设计都在详细设计里了。**

观赏性游戏，主要亮点在于代码实现方面，采用矩阵运算的内核，opencv的显示，最终达到的效果很好，cpu运算速度很快，可以运算的图片大小可以很大，1920\*1080个像素点的图片迭代速度优良。

**四．详细设计**

**A.中期答辩时的代码详解**

**中期版本000MyLife.py(基于life.py修改)(烂尾)**

模块化编程，程序开始运行，从导入包开始，然后进入全局变量初始化，包括实例化一个矩形，细胞大小的初始化，细胞数量的初始化，细胞状态的定义，三个颜色模式的定义，画布鼠标等颜色的定义。之后遇到函数定义暂时跳过。直到末尾。



以此为入口函数，直接进入Life唯一一个实例化。

在类初始化函数中，pygame初始化一次，按照实例化好的矩形的大小创建建一层画布，画布标题，画布字体，初始化画布单元格为0。

实例化一个时钟，用来前期调试代码时不让图像闪退。

在画布上画出两块矩形，用来区别游戏区域与游戏状态区域。

使用第一种颜色模式载入颜色。

进入while循环

clock.tick(60)用来让画布不会闪退，即暂停游戏画面。只在写代码前期起作用，现在这句注释掉对整体没有影响。

进入函数update()，如果布尔型变量running为真进入函数next()，

next()函数，新建单元格值全为0的矩阵用来存放下一代细胞的状态，两层for循环内首先数当前单元格及周围8个单元格共九个单元格中活着的细胞的数量。如果当前单元格的状态是活着的1或者在本次计算中被杀死的5，则它在本次计算中会被赋值为2，如果在表示细胞活着和死亡的矩阵只包含0,1的9个单元格中有两个细胞活着，那么当前单元格的下一代状态为活着的。如果9个单元格中有三个细胞活着，那么首先判断用来表示细胞生死的矩阵中该细胞是否是死去的，如果是，那么在用来表示细胞在本次计算中被复活和被杀死的矩阵中，给该单元格标记为复活4，并标记当前单元格的下一代状态为活着。如果9个单元格中活着的细胞个数不为2个和3个，如果该单元格细胞原来状态为活着的，那么在表示细胞在本次计算中被复活和被杀死的矩阵中，给该单元格标记为杀死5，并标记当前单元格的下一代状态为死亡。

将计算for循环结果赋值给细胞状态矩阵。迭代次数加一。

update()函数运行完成进入draw(screen)函数，两层for循环中

对矩阵表示细胞生死的单元格用ALIVE\_COLOR与DEAD\_COLOR表示

对矩阵表示细胞本次计算后的状态的单元格用NEW\_ALIVE\_COLOR，DIED\_OUT\_COLOR，MARKED\_COLOR表示

其中ALIVE\_COLOR，NEW\_ALIVE\_COLOR颜色一样，其他的颜色一样。

然后绘制鼠标，用小方格代替

覆盖绘制一个矩形，在该矩形上绘制游戏的状态。

pygame.display.update()函数用来将上预绘制的元素真正更新到画布上。

下面是按键中断检测：鼠标点击检测；键盘按下检测。

**B.下面是中期答辩之后的工作**

**版本一 001MyLifeOpenCV.py**

基于对中期代码的理解，使用opencv-cpu进行简单的功能实现。opencv-gpu用cmake+wingw的opencv总是不适用java、python2、python3。所以我的opencv使用不了CUDA。现阶段不能解决。

函数式编程，非模块化编程。

期末大作业附件——图一

初始化，创建102\*102大小的图，大小随算力可更改，我这里可以支持更高，但是后面我会介绍我将单个像素点膨胀为一块小正方形的办法。

用img和img0表示细胞死亡和存活，用img3表示细胞的被死亡被复活被标记等状态。

期末大作业附件——图二

while循环为程序主体，此前初始化参数。

delayCountNum用来控制图像更新的时间间隔，即一代存活的时间，这里10表示10个while循环的时间。实际上，不进if delayCount == delayCountNum:与进这个条件判断所花的时间呈倍数关系。因为处理图像的方式跟pygame里一样用两层for循环，因此时间复杂度为O(m^2)，这是pygame和opencv里需要解决的主要问题。

generation用来表示迭代次数

这里为了防止图像迭代次数很大了之后，图像趋于静止，所以每隔theNightTime个迭代周期进行一次图像叠加的操作，模拟一次细胞数量显著变化。

进入next\_pic函数

期末大作业附件——图三

首先数img0中本像素点及周围8个像素点，值为1的个数，

期末大作业附件——图四

数alive细胞个数的方法不好，完全可以使用np.ones((3, 3))跟以该点为中心的3\*3矩阵取矩阵卷积，结果就是为1的个数。但是这样写不如图中写的简单和快。等有优化时间再写。

两层for循环内，第一个条件判断在我的功能里不起作用，我只需要第二个条件判断加上第二的两层for循环，就可以实现功能。

期末大作业附件——图五

进入bit\_bloat函数，用来进行比特膨胀——从STM32-MDK中学来的名词，将一个像素点的状态反映成一块矩形的状态。我这里是把矩形里像素点的值赋值为一个像素点的状态值0或1。

mul表示1个像素点膨胀为mul\*mul的矩阵，prune表示图像四周剪裁prune//2的边框。

最后q键退出程序

**版本三 003MyLifeOpenCVMatConv.py**

版本二烂尾

基于前两次的基础，而且gpu还不能用，用矩阵运算直接冲

期末大作业附件——图六

ker3 = np.ones((3, 3))用作卷积核，用来跟原图做卷积，结果就是版本一num\_alive函数的结果，将两层for循环里的9个加法运算转换成了矩阵卷积运算。代价见下图

期末大作业附件——图七

由于cv2.filter2D函数会自动补齐矩阵卷积运算后损失的2倍步长的四个边框的数据，而且自动补足的数据不满足我的要求，所以我将它们统一赋值为2，这样的效果为细胞靠近边框就被杀死。后面我修改赋值为3，效果为细胞靠近边框状态不变。

使用矩阵卷积减少了一个两层for循环，速度快了很多

效果图见下图。

期末大作业附件——图八

**版本四 004MyLifeColor.py**

增加新的功能，如文件名，将单层的二值图像变更成BGR三原色图像，实际上可以覆盖BGR色彩空间的所有颜色，这样观赏效果会更好。覆盖方式为坐标元素值乘以range(100)/100，现阶段不这样做，而且调成一个好看的颜色很花时间。

期末大作业附件——图九

用来增加颜色，原理：三层图像分别进行生命游戏核心算法的运算。时间复杂度仅为单层二值图像的3倍，约为1倍。

效果图见下图。

期末大作业附件——图十

只剩这个hhh函数中的两层for循环消耗算力。

**版本五 005MyLifeWithoutFor.py**

这两层for循环作用是：

1-3\_1

0-3\_1

1-4\_1

0-4\_0

\*-else\_0

思考了一段时间

解决方案如下图

期末大作业附件——图十一

这张图是我整个工程的核心，可能是全网第一个提出来的，未考证。

life函数可以完美替代两层for循环的功能，这样会使程序运行效率有质的飞跃。也就是说时间复杂度为O(1)

**版本六 006MyLifeWithoutForWithColor.py**

期末大作业附件——图十二

所有版本都存在的功能，一直没解释，模拟一次生物大灭绝，没想好名字，就用大灭绝代替，实际效果不是大灭绝。实现方法是使用random函数随机生成一张图，然后矩阵相加到随机一层图上，模拟的效果是迭代到达一定次数时，某层图上不按生命游戏规则随机出现活的细胞，这样做会使整个图像不会因为迭代次数过多，导致图像变化不显著。

相较于其他版本，这个版本的大灭绝功能改成三种颜色的生物在一次灭绝中，随机灭绝其中一种。

**其他**

中期答辩的时候说过，使用一个数组表示细胞迭代前后的生死，其中采用的策略是数组单个元素值不只为0或1，即用其他数字表示细胞的其他状态。代码占用的空间约只有一个数组的大小。

我的代码只追求了速度，没有考虑空间复杂度。可以考虑，但不是这次报告中考虑。

因为以我现在对python的理解，解决空间复杂度的问题，就是把几行代码写到一行去，这样减少了中间变量的个数也就降低了空间复杂度，不同情况可能有所不同。

我的策略，使用HEIGHT \* WIDTH \* 3的矩阵表示图像，使用HEIGHT \* WIDTH的矩阵存储每层图像跟3 \* 3全1矩阵作矩阵卷积后修改边框的单层图像的参考矩阵。这个参考矩阵的值表示对应单个像素点及其周围1的个数，根据生命游戏规则，

1-3\_1----0-3\_1----1-4\_1----0-4\_0----\*-else\_0

符号解释：原矩阵某位置值-参考矩阵中对应位置值\_下一代矩阵对应位置值，总结出生命游戏4条规则。

**五．测试(如何使用我的代码)**

**版本一 001MyLifeOpenCV.py**

**A：**4：HEIGHT, WIDTH = 602, 602——效果为细胞为像素大小的游戏窗口

**B：**取消注释5：# HEIGHT, WIDTH = 102, 102

且取消注释90：# img = bit\_bloat(img)——效果为细胞为5\*5像素大小的游戏窗口，且游戏窗口去掉了原图的3\*5个像素大小的边框

B方式的参数修改：

59：mul = 5 # 放大系数 对应1个像素

60：prune = 6 # 2的倍数 剪裁留中间去四周

**其他：修改细胞迭代周期、大灭绝周期和大灭绝开关**

78：delayCountNum = 10

79：delayCount = delayCountNum - 1

80：generation = 0 # 迭代次数

81：theNightTime = 100 # 模拟灭绝周期

82：theNightmare = 1 # 模拟生物大灭绝 theNightmare in range(theNightTime) == True 为开启大灭绝

期末大作业附件——图十三

期末大作业附件——图十四

期末大作业附件——图十五

**版本三 003MyLifeOpenCVMatConv.py**

运行速度变快了

期末大作业附件——图十六

**版本四 004MyLifeColor.py**

图比较大，运行速度慢，凸显两层for循环的弊端

期末大作业附件——图十七

**版本五 005MyLifeWithoutFor.py**

图更大了，运行速度很快

期末大作业附件——图十八

**版本六 006MyLifeWithoutForWithColor.py**

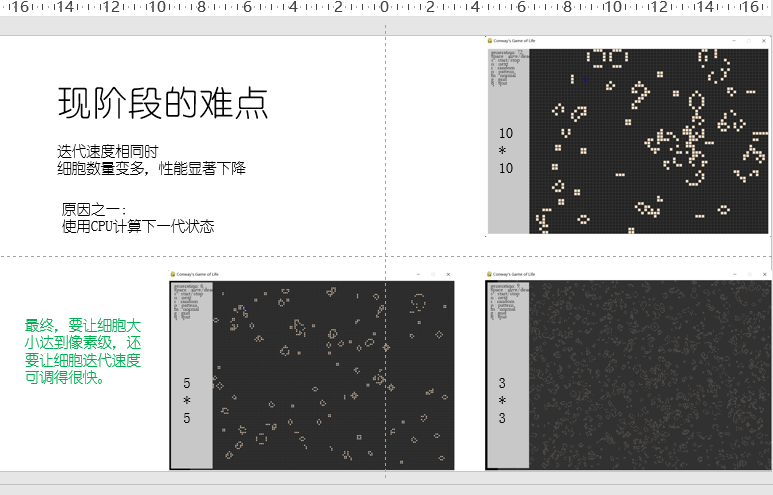
这个版本修改成了全屏游戏

修改4：times = 10——表示全屏窗口中细胞的大小为10\*10

期末大作业附件——图十九(times=10)

期末大作业附件——图二十(times=1)

**六．总结与展望**

****已经解决了中期答辩定下的目标(左图)，虽然当时想的解决方案不是现在的这个方案，但是JavaScript的WebGL包真的很值得学习。我暂时没学，一切放到考研之后。

代码已经放上了github

https://github.com/cxy-251/the-first-repo/tree/main/lifeGame