

# DSAAb report

2022.5.22

小组成员：郭永杉，杨瀚博，简欣瑶

Github : <https://github.com/curcurSheep/CellProj>

## 一、问题描述

本次project的目的是模拟大量细胞在二维平面上的动态运动，主要包括细胞的移动，细胞的碰撞以及细胞颜色的改变三个部分。

在本过程中有两次需要查找细胞附近范围内的细胞：一次为细胞在运动时判断是否会与周围的细胞发生碰撞；另一次为在改变颜色的时候根据perception range中的细胞改变颜色。如果使用暴力法进行判断的时间复杂度为  $O(n)$ ，但是此方法在细胞数过多的情况下效率不佳。

该问题的难点主要在于数据结构的选择以及如何充分发挥该数据结构的优势上面。我们小组最后选择了**四叉树**（理论上的查找复杂度为  $O(\ln(n))$ ）并充分发挥了树结构查找迅速的优势，该系统最终可以通过GUI演示程序或者利用命令行输出与标准结果进行对比得到正确的结果。

## 二、RGBY Cell 系统

### （一）主体思路和拟态过程 (Simulation process)

我们的思路是将细胞的数据存入四叉树结构并初始化四叉树。之后依照细胞的ID对所有细胞进行移动并同时改变细胞在四叉树中的位置，最后统一改变细胞的颜色实现一次循环。

QTree.java

```
public void moveOneStep() {
    for (Cell cell : this.cells) {
        move(cell);           移动细胞
        cellShouldChange(cell); 并且改变细胞在树上的位置
    }
    detect_and_set_color(root);
}

public void detect_and_set_color(Node root){
    for (Cell cell : cells) {...} 得到细胞旁边的细胞颜色
    for (Cell cell : cells){
        cell.setColor(cell.perception_colors); 改变细胞的颜色
    }
}
```

### 移动和时间的关系：

在进程中，细胞是每1/15秒移动一次。举例而言，在0-1/15秒内，细胞不动，在1/15时刻，细胞进行移动。普遍而言，如果在  $[k/15, (k+1)/15), k \in N, k \geq 0$  时间查询，将查询到

细胞运动  $k$  次后的信息。

## 主要方法说明

拟态过程的具体三个步骤实现思路如下。

QTree.java

### move (cell) :

首先根据细胞（以下称为细胞a）的不同颜色创建反映细胞运动轨迹的矩形，这个矩形内所有的其它细胞都可能与细胞a发生碰撞。利用二叉树搜索迅速的优势可以快速检索到矩形内所有可能发生碰撞的细胞,同时大大减少了需要判断可能碰撞的细胞的数量。

之后判断这些可能与细胞a发生碰撞的细胞是否真的会发生碰撞，如果不是则细胞a依照自己的颜色正常移动；反之则移动到与碰撞细胞相切的位置。

### cellShouldChange (cell) :

遍历所有细胞，首先判断它是否移出过其原本所在的节点，如果没有则不改变它在树中的位置；若有，则首先查找其原本所在节点的brother，若仍没有找到则递归式地向父节点查找，直到找到包含该细胞的节点。之后直接在这个节点处向下进行广度优先查找找到对应子节点，将细胞从原节点删除，同时插入新节点。

### detect\_and\_set-color:

遍历所有细胞，根据perception range里面的其它细胞数目来改变自己的颜色。同样这里查找perception range内的细胞的方法也充分发挥了二叉树查找迅速的优势。

## (二) 实现和代码说明

### Cell

cell是需要动态模拟的对象，每一个Cell有如下属性。

```
1 public double radius ;//cell的半径
2 public double x ;//cell的横坐标
3 public double y ;//cell的纵坐标
4 public double perception_range ;//cell检测决定color的周边环境的范围大小
5 public Color color;//cell的颜色
6 public int ID = num; //cell的ID
7 public QTree.Node node;//cell在二叉树中的节点
8 public Rectangle perception_rectangle;//cell检测决定color的周边区域
```

我们对每一个Cell主要实现了如下功能。

```
1 public boolean check_if_overlapped(Wall wall)
2 //检查cell是否与墙壁碰撞
3 public boolean check_if_overlapped(Cell b)
4 //检查cell是否与Cell b碰撞
5 public void move(double x, double y)
6 //将cell移动到指定位置
7 public boolean move();//根据cell的颜色不同让cell移动1/15
8 public void change_color(Color color_changed)
```

```

9 //改变cell的颜色
10 public void setColor(Color[] perception_color)
11 //cell根据传入的颜色比例改变自身颜色
12 public String standard_output()
13 //标准化输出结果
14 public static Cell queryID(int num)
15 //根据ID返回对应的cell

```

## Rectangle

Rectangle代表了系统中任意一块矩形区域，它的主要作用是作为四叉树节点的边界以及判断细胞移动过程中是否会与其它细胞发生碰撞。每一个Rectangle有如下属性：

```

1 public double x;//中心横坐标
2 public double y;//中心纵坐标
3 public double w;//宽度
4 public double h;//高度
5 public double north;//上底的纵坐标
6 public double south;//下底的纵坐标
7 public double east;//左边的横坐标
8 public double west;//右边的横坐标

```

对于rectangle我们主要实现了如下功能

```

1 public boolean isOverlap(Rectangle rectangle)
2 //检查两个rectangle是否有重叠
3 public boolean isContainCell(Cell cell, boolean isCritical)
4 //检查rectangle是否包含cell的中心
5 public boolean isContainCellPart(Cell cell)
6 //检查rectangle是否部分包含cell

```

## Wall

Wall表示了一个长方形，其主要作用是限制cell的移动边界，其大小与四叉树根节点的边界大小相同。我们默认根节点左下角的坐标是（0，0），所以wall并不需要确定中心位置，只需确定宽和高。

```

1 public double width ;
2 public double height;

```

## QTree

QTree是储存Cell的四叉树，也是本系统的主体部分，通过调用moveOneStep就能让系统前进一帧。以下是它的属性。

```

1 public Wall wall;//限定该四叉树的边界
2 public Node root;//四叉树的根节点
3 public ArrayList<Cell> cells//四叉树中所有的细胞

```

基于node我们实现了四叉树的很多功能，以下是四叉树的方法。

```

1 public static ArrayList<Cell> cellOverlap(ArrayList<Cell> cells,
2 Cell cell)
3 //检查cell和cells中的细胞是否有重合

```

```

4 insert(Cell cell)
5 //在树中自动将cell插入到合适的位置
6 public void cellShouldChange(Cell cell)
7 //在cell移动过后检查它在树中是否需要移动，如果是则移动到相应位置
8 private ArrayList<Cell> dfs(Node node)
9 //遍历node下的所有cell并返回遍历到的cell
10 public boolean move(Cell cell)
11 //根据cell的颜色自动让cell移动1/15步长
12 public void detect_and_set_color(Node root)
13 //遍历所有cell并更改颜色
14 public void moveOneStep()
15 //整个树经历1/15秒并自动更新

```

## Node

Node是四叉树的内部类，代表四叉树的节点。几何上每一个节点表示一块区域。我们设定了两种Node，一种树节点，一种为叶节点。对于树节点而言，它是已经分叉的(divided)，同时它具有四个子节点；而对于叶节点，它是没有分叉的，所以不具有四个子节点，但是它会记录在包含在该节点对应区域的细胞。属于叶节点的细胞个数不会超过预设的容量 (capacity=4)，当细胞数超过容量，叶节点就会分类成子节点，同时将它所包含的细胞分配给四个子节点。

它具有如下属性。

```

1 public Rectangle boundary;
2 //界定node边界的长方形
3 public boolean divided;
4 //判断该node是否发生了分裂
5 final static private int capacity = 4;
6 //设定阈值，当节点中的cell超过capacity时发生分裂
7 public Node father;
8 // 记录该节点的父节点
9 public ArrayList<Node> brother ;
10 // 记录与该节点同级的节点
11
12 public Node ne;
13 public Node nw;
14 public Node se;
15 public Node sw;
16 // node : 该节点的四个子节点，当且仅当该节点分裂时才有这个属性
17
18 public ArrayList<Cell> cells;
19 // leaf : 该节点下的细胞，当且仅当该节点没有分裂时才有这个属性
20

```

以下是我们实现的node方法

```

1 public boolean insert(Cell cell)
2 //在当下节点插入一个cell
3 public boolean divide()
4 //分裂当下节点
5 public boolean isContain(Cell cell, boolean critical)
6 //判断cell是否在当下节点
7 public ArrayList<Cell> cellInRange(Rectangle range, boolean critical)

```

```

8 //判断cell是否在range中
9 public ArrayList<Cell> cellInPerception(Rectangle searchRange,
10 Rectangle judgeRange)
11 //返回perception range里的所有点

```

### (三) GUI和可视化

GUI使用了OpenGL,其中主要使用了Renderer和EventListener两个类。

关于可视化的改进想法:

(1) 颜色 (2) 结束控制 (3) 调用?

#### Controller

控制二叉树与GUI的交互。具体控制方法是通过GUI调用controller进而使得二叉树前进一帧,它具有如下属性。

```

1 public static double TIME_STEP = 1 / 15.0;
2 //系统演化时间步长
3 public static double w;
4 //墙的宽度
5 public static double h;
6 //墙的高度
7 public static int _n_cell;
8 //细胞的数目

```

以下是controller中实现的方法

```

1 public static void init()
2 //根据读取的文件创建系统,包括生成树并插入细胞
3 public static QTree build_QTree_from_samplefile(File file)
4 //根据文件创建树
5 public static void init_diy_guo(int cell_num_to_test)
6 //随机生成初始化数据并初始化二叉树

```

#### Renderer

```

1 public static void init()
2 //初始化GUI窗口并设置窗口属性
3 public static double getWindowH()
4 //获得当前窗口的高度
5 public static double getWindowW()
6 //获取当前窗口的宽度

```

#### EventListener

```

1 public void init(GLAutoDrawable drawable)
2 //每次初始化窗口都会调用一次,可以在其中设置背景色等
3 public void display(GLAutoDrawable drawable)
4 //每一帧调用一次
5 public void reshape(GLAutoDrawable drawable, int x, int y,
6 int width, int height)
7 //每次改变窗口大小调用一次

```

```

8 public void drawSingleCircle(GL2 gl, double x, double y, double r,
9 Cell.Color c)
10 //在窗口中绘制一个圆

```

## 三、测试和结果

### (一) 结果正确性检验

本项目的测试部分由多种测试组成。根据难度可以分为简单测试与大数据量测试，根据呈现方式可以分为可视化直观测试与命令行输出自动测试。虽然测试种类多样，但核心原理都是根据正确答案比较输出结果。由于时间有限，不能通过基于不同原理的实现方法进行交叉验证，在本项目中，我们仅基于实验课中给定的sample2参考图形样例与sample.zip中的txt文件对输出结果进行检验。因此，我们的测试主要可分为两个部分。

#### 第一部分：静态测试

- a. 目标：检验检测变色程序；
- b. 实现方式：根据老师给出的sample2.txt生成四叉树，并填满整个空间。禁用 `move()`，仅测试 `detect_and_set_color()`；
- c. 检测手段：颜色变化需要检查生成过程与最终稳态两个部分，生成过程只能基于命令行输出测试，最终稳态既可以基于命令行进行检测，也可以根据输出的最终稳态图像进行检测；

#### 第二部分：动态测试

- d. 目标：检验检测移动与变色综合程序；
- e. 实现方式：
  - i. 根据老师给出的sample.zip中的txt文件生成四叉树，并读取对应Query清单进行查询，将查询结果输出到自建的txt文件中，再读取相应的sample\_out.txt正确答案文件进行逐行比对；[简]
  - ii. 直接生成随机sample文件进行四叉树生成与查询操作，小样本可以手动粗略计算进行检验，大样本只能直观查看图形渲染效果，作为图像生成程序的测试与检验；
- f. 检测手段：命令行输出自动比对与输出图像的直观比对；

### (1) 过程的正确性检验

#### 测试思路 and 实现：

在此部分测试中，测试数据为提供的sample，我们将查询的结果与提供的答案进行比较，三个sample全部通过测试（在5%的误差内）。

Test.java

```

1 // 读入sample数据和query,进行查询,并且将查询的结果存入txt,调用compareResult和给
  定的结果进行比较
2 public static void query_test(int num_sample)
3 public static void CompareResult(int fileNum, int numOfQuery)

```

#### 细胞状态查询对比结果：

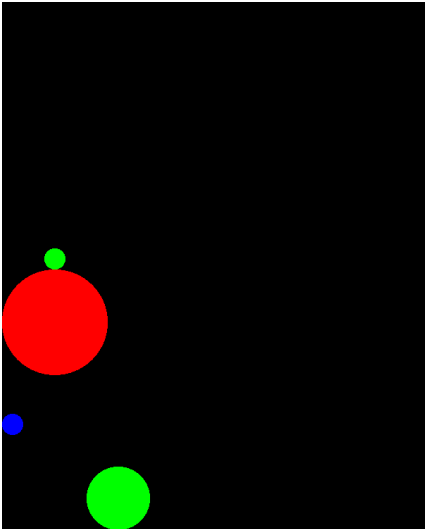
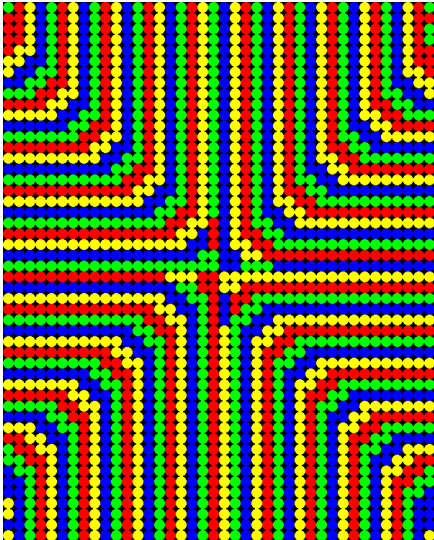
```
Test x
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
----- test sample 1 -----
-- results of query are in ../res/output/test_sample1.txt
all correct!
----- test sample 2 -----
-- results of query are in ../res/output/test_sample2.txt
all correct!
----- test sample 3 -----
-- results of query are in ../res/output/test_sample3.txt
all correct!

Process finished with exit code 0
```

## (2) 图像显示检验

### 1. 简单测试

- 测试数据: sample 1 & sample 2
- 测试方法: 调用Main.java的GUI模式 (默认模式)
- 测试结果 (此处)

sample 1	sample 2
	

### 2.大数据量测试

用户可以自主设定需要生成的Cell数量cell\_num\_to\_test, 函数预设每个Cell的最大半径为0.5, 并根据cell\_num\_to\_test的大小生成能够容纳所有Cell的最小方框以生成合宜大小的四叉树。值得注意的是, 为了防止Cell数量太小导致函数异常, 方框的最小边长为10。假设cell\_num\_to\_test为1000, 函数将根据标准格式生成diy\_sample1000.txt文件, 并产生1000个标准格式的查询输出附于文件中四叉树数据之后。

Controller.java

```
1 // 输出图像
2 public static void init_diy_guo(int cell_num_to_test)
```



```

3 // 调用显示
4 public static void main(String[] args) {
5     init_diy_guo(30);
6     Renderer.init();
7 }

```

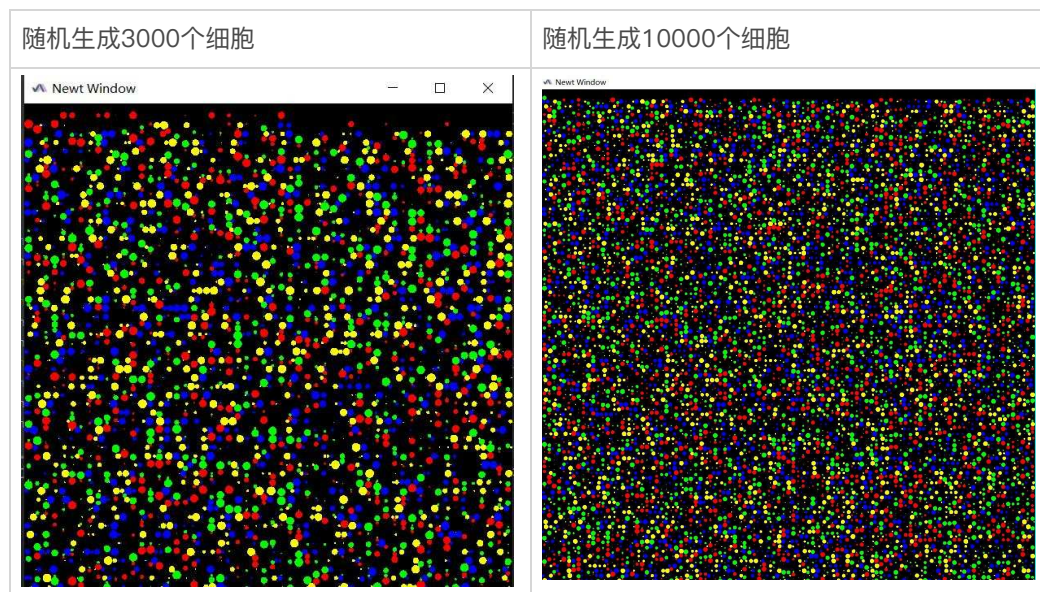
Test.java (init\_diy\_guo中调用的方法)

```

1 // 生成随机数据与导出Sample文件
2 public static void GenBigSquare(int cell_num_to_test)
3 // 读取Sample文件，自动生成二叉树。
4 public static QTree build_QTree_from_samplefile(File file)

```

结果显示：



### 3. 动态展示

我们随机生成了100个细胞的数据，并将GUI的动态结果录制在以下视频中

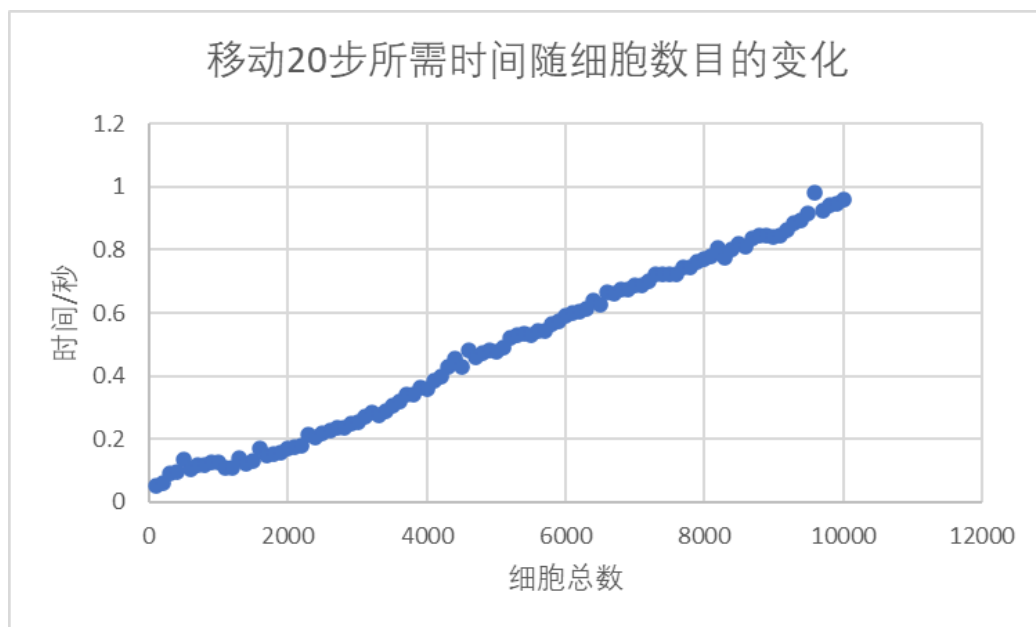
链接：<https://pan.baidu.com/s/1ttN-KzW8uRBx7kyt6cZaJg>

提取码：u96u

## (二) 时间复杂度检测

我们利用自己随机生成的测试数据描绘了系统运行100步('moveOneStep' 遍历100次)所需的时间随细胞数的变化，结果如下图所示。





系统每移动一帧包括三步，即遍历细胞并移动每一个细胞，遍历细胞并更改其在树上的位置，遍历细胞并改变所有细胞的颜色。

如果是暴力算法，则遍历本身贡献 $n$ 的复杂度，查找周围的粒子以决定是否碰撞及更改颜色也会贡献 $n$ 的复杂度。所以总体上复杂度会是  $n^2$ 。

而采用四叉树，其遍历贡献 $n$ 的复杂度后，根据四叉树的结构其查找的时间复杂度为  $\log(n)$ ，所以总复杂度应为  $n \log(n)$ 。从图上看，检测时间的复杂度非常接近线性，可见四叉树令查找周边细胞的时间大大缩小，从而让整体趋势接近线性。

## 四、调用方法

以下标明使用命令行调用主函数的方法，以及在使用Idea时遇到可能困难的解决方法。其中

### (一) 命令行

仅展示在Windows系统下的调用方法

编译：

```
1 javac -cp .;lib\algs4.jar;natives\gluegen-rt.jar;natives\jogl-all.jar src  
  \*.java -d bin
```

运行：

(1) 默认无参数（默认调用GUI）

```
1 java -cp bin;lib\algs4.jar;natives\gluegen-rt.jar;natives\jogl-all.jar Mai  
  n < res\sample\sample1.txt
```

(2) 使用参数 terminal

```
1 java -cp bin;lib\algs4.jar;natives\gluegen-rt.jar;natives\jogl-all.jar Mai  
  n terminal < res\sample\sample1.txt
```

(3) 使用参数 gui

```
1 java -cp bin;lib\algs4.jar;natives\gluegen-rt.jar;natives\jogl-all.jar Mai  
  n gui < res\sample\sample1.txt
```

注：重定向文件需为cmd的相对路径

## (二) IDEA

如果遇到类似于以下问题（找不到类），可以通过 file -> Invalidate Caches -> Invalidate and Restart 解决。

