CS203B Project - RGBYCellLife

Author

12012128 孙逸涵(Yihan Sun)

12012336 龚雪(Xue Gong)

12010137 李璇(Xuan Li)

Catalogue

Introduction

在本次project中，我们在一个二维平面区域中对一些简化后的虚拟单细胞生命进行模拟,进行了移动的模拟，以及对自身和周围细胞的检验，并模拟在检验后进行变换颜色、碰撞感知、行动和停止等活动，更新自身的位置、颜色等属性，并将结果通过GUI和Terminal模式分别输出。

System simulation

Cell类

Cell的参数包括id，半径r，x、y坐标，颜色以及感知范围。在Cell类中设置细胞基本的属性，以及对周围细胞的判断和对自己的更改。

其中，error变量用于在该类的方法中比较double类型数据是否相等时，作为精度来使用。

如：`if (Math.abs(d1 - d2) < error)`, d1和d2均为double类型数据。

相应代码如下所示：

```
//  Does NOT change during the life cycle of the cell
private final int identity;// An integer represent the uniqueness of cell
private final double radius;// Unit is meter

private Rect rect;

private final double error = 0.0001;

/*
Position: Represented by a real number pair (x,y)
x: Distance between the left border of the plain and the center of the cell
(in meters)
y: Distance between the bottom border of the plain and the center of the cell
(in meters)
*/
private double xPosition;
private double yPosition;

private char color;
private final double perceptionRange;// 感知范围，正方形边长的一半

private int numOfR;
private int numOfG;
private int numOfB;
private int numOfY;
private int numOfC;

public Cell(int id, double x, double y, double r, double perceptionRange, char
color) {
    identity = id;
    radius = r;
    xPosition = x;
    yPosition = y;
    this.color = color;
    this.perceptionRange = perceptionRange;
}
```

Cell中的isOverlap方法通过计算当前细胞和入参细胞的圆心之间的距离与半径之和作比较，来判断两个细胞是否重叠。相应代码如下所示：

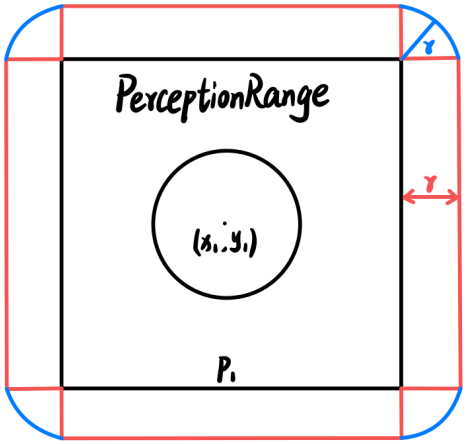
```
// Decide if two cells are overlapping
public boolean isOverlap(Cell c1, Cell c2) {
    double x1 = c1.xPosition;
    double y1 = c1.yPosition;
    double r1 = c1.radius;
    double x2 = c2.xPosition;
    double y2 = c2.yPosition;
    double r2 = c2.radius;
    return Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2)) < r1 + r2 ||
Math.abs(Math.sqrt(Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2)) - (r1 + r2)) <
error;
}
```

isPerceived方法通过计算当前细胞和入参细胞的圆心之间的距离与当前细胞的感知范围作比较，来判断入参细胞是否在当前细胞的感知范围内。在本次project中，我们通过两种不同的计算方式实现了isPerceived方法，相应代码如下所示：

```
// Decide if current cell(调用此方法的cell) is in cell c's(入参的cell) perception
rage
public boolean isPerceived(Cell cell) {
    double xLength = Math.abs(cell.xPosition - xPosition);
    double yLength = Math.abs(cell.yPosition - yPosition);
    double p = cell.perceptionRange;
    if (xLength > yLength && (xLength < p + radius || Math.abs(p + radius -
xLength) < error))
        return true;
    else if (xLength < yLength && (yLength < p + radius || Math.abs(p + radius
- yLength) < error))
        return true;
    else if ((xLength > p || Math.abs(xLength - p) < error) && (yLength > p ||
Math.abs(yLength - p) < error)) {
        return Math.sqrt(Math.pow(yLength - p, 2) + Math.pow(xLength - p, 2))
< radius || Math.abs(Math.sqrt(Math.pow(yLength - p, 2) + Math.pow(xLength -
p, 2)) - radius) < error;
    }
    return false;
}
```

在本次project中，我们主要使用了isPerceived方法的第二种实现。该方法将需要感知周围细胞数量的细胞的perceptionRange的外围部分划分为8个部分，即四个矩形区域和四个四分之一圆区域，区域的宽度为需要判断是否被当前细胞感知的其他细胞的半径，当周围其他细胞的圆心落入此区域中，则判定该细胞被感知。

划分区域如图所示：



相应代码如下：

```
public boolean isPerceived(Cell cell) {
    double x = xPosition;
    double y = yPosition;
    double r = radius;
    double x1 = cell.xPosition;
    double y1 = cell.yPosition;
    double p1 = cell.perceptionRange;

    if ((x1 - p1 < x || Math.abs(x1 - p1 - x) < error) && (x < x1 + p1 ||
Math.abs(x1 + p1 - x) < error)) {
        if (y >= y1 && (y < y1 + p1 + r) || Math.abs(y1 + p1 + r - y) < error)
{
            return true;
        } else return y <= y1 && (y > y1 - p1 - r || Math.abs(y1 - p1 - r - y)
< error);
    } else if ((y1 - p1 < y || Math.abs(y1 - p1 - y) < error) && (y < y1 + p1
||        Math.abs(y1 + p1 - y) < error)) {
        if (x <= x1 && (x > x1 - p1 - r || Math.abs(x1 - p1 - r - x) < error))
{
            return true;
        } else return x >= x1 && (x < x1 + p1 + r || Math.abs(x1 + p1 + r - x)
< error);
    } else
        return (((x > x1 + p1 || Math.abs(x1 + p1 - x) < error) && (y > y1 +
p1 || Math.abs(y1 + p1 - y) < error)) || ((x > x1 + p1 || Math.abs(x1 + p1 -
x) < error) && (y < y1 - p1 || Math.abs(y1 - p1 - y) < error)) || ((x < x1 -
p1 || Math.abs(x1 - p1 - x) < error) && (y < y1 - p1 || Math.abs(y1 - p1 - y)
< error)) || ((x < x1 - p1 || Math.abs(x1 - p1 - x) < error) && (y > y1 + p1
|| Math.abs(y1 + p1 - y) < error))) && (Math.pow(x - x1, 2) + Math.pow(y - y1,
2) <= Math.pow(r + Math.sqrt(2) * p1, 2) || Math.abs(Math.pow(x - x1, 2) +
Math.pow(y - y1, 2) - Math.pow(r + Math.sqrt(2) * p1, 2)) < error);

}
```

Cell中的change方法是在当前细胞计算完周围各种颜色的细胞数量以后，若符合变色的条件，就进行颜色变化，反之则不变。相应代码如下所示：

```
// Cell changes its color
public void change() {
    switch (color) {
        case 'r':
            if (numOfR >= 3 && (double) numOfR / numOfC > 0.7) {
                color = 'g';
            } else if (numOfY >= 1 && (double) numOfY / numOfC < 0.1) {
                color = 'y';
            }
            break;
        case 'g':
            if (numOfG >= 3 && (double) numOfG / numOfC > 0.7) {
                color = 'b';
            } else if (numOfR >= 1 && (double) numOfR / numOfC < 0.1) {
                color = 'r';
            }
            break;
        case 'b':
            if (numOfB >= 3 && (double) numOfB / numOfC > 0.7) {
                color = 'y';
            } else if (numOfG >= 1 && (double) numOfG / numOfC < 0.1) {
                color = 'g';
            }
            break;
        case 'y':
            if (numOfY >= 3 && (double) numOfY / numOfC > 0.7) {
                color = 'r';
            } else if (numOfB >= 1 && (double) numOfB / numOfC < 0.1) {
                color = 'b';
            }
    }
}
```

isIn方法用于判断即将移动的细胞cell的移动范围内有无别的细胞。相应代码如下所示：

```
// Decide if current cell(调用此方法的cell) is in the motion path of the
parameter cell(入参的cell)
public boolean isIn(Cell cell) {
    double x1 = xPosition;
    double y1 = yPosition;
    double r1 = radius;
    double x = cell.xPosition;
    double y = cell.yPosition;
    double r = cell.radius;
```

```
    switch (cell.getColor()) {
        case 'r':
            if (y1 >= y && y1 <= y + (double) 1 / 15) return Math.abs(x1 - x)
<= r + r1;
            else if (y1 > y + (double) 1 / 15)
                return Math.pow(x - x1, 2) + Math.pow(y + (double) 1 / 15 -
y1, 2) <= Math.pow(r + r1, 2);
            break;
        case 'g':
            if (y1 >= y - (double) 1 / 15 && y1 <= y) return Math.abs(x - x1)
<= r + r1;
            else if (y1 < y - (double) 1 / 15)
                return Math.pow(x - x1, 2) + Math.pow(y - (double) 1 / 15 -
y1, 2) <= Math.pow(r + r1, 2);
            break;
        case 'b':
            if (x1 >= x - (double) 1 / 15 && x1 <= x) return Math.abs(y - y1)
<= r + r1;
            else if (x1 < x - (double) 1 / 15)
                return Math.pow(x - (double) 1 / 15 - x1, 2) + Math.pow(y -
y1, 2) <= Math.pow(r + r1, 2);
            break;
        case 'y':
            if (x1 >= x && x1 <= x + (double) 1 / 15) return Math.abs(y1 - y)
<= r + r1;
            else if (x1 > x + (double) 1 / 15)
                return Math.pow(x + (double) 1 / 15 - x1, 2) + Math.pow(y -
y1, 2) <= Math.pow(r + r1, 2);
            break;
        }
    return false;
}
```

isToEdge方法用于判断细胞是否移动到了画面边界，若移动至边界则不再继续移动。相应代码如下所示：

```
public boolean isToEdge(int width, int height) {
    if (color == 'r') { // Upward
        return yPosition + (double) 1 / 15 + radius > height ||
Math.abs(yPosition + (double) 1 / 15 + radius - height) < error;
    } else if (color == 'g') { // Downward
        return yPosition - (double) 1 / 15 - radius < 0 || Math.abs(yPosition
- (double) 1 / 15 - radius) < error;
    } else if (color == 'b') { // Left
        return xPosition - (double) 1 / 15 - radius < 0 || Math.abs(xPosition
- (double) 1 / 15 - radius) < error;
    } else if (color == 'y') { // Right
        return xPosition + (double) 1 / 15 + radius > width ||
Math.abs(xPosition + (double) 1 / 15 + radius - width) < error;
    }
    return false;
}
```

CellSystem类

CellSystem类主要用于控制整个模拟细胞生存环境的进程的进行。

其中，error变量用于在该类的方法中比较double类型数据是否相等时，作为精度来使用。

如：`if (Math.abs(d1 - d2) < error)`，d1和d2均为double类型数据。

静态变量mode用于控制程序是以GUI模式运行还是以Terminal模式运行，mode默认值为0，当命令行参数为“Terminal”时，mode=1，程序通过Terminal模式运行；否则程序通过GUI模式运行。除此之外，在Main方法中，我们还使用重定向的方式读取文件，此时不同configuration的命令行参数已被提前设置。

使用LinkedList储存Cell

相应代码如下所示：

```
private int width; // Width of the window in GUI
private int height;// Height of the window in GUI
private double time;
private int numOfQ; // The number of queries(for terminal mode)
private double[][] acts;
private LinkedList<Cell> cells;
private final double error = 0.00001;

/*
type = 0 -> GUI mode
type = 1 -> Terminal mode
*/
public static int mode = 0;
```

使用Array储存Cell

相应代码如下所示：

```
private int width; // Width of the window in GUI
private int height;// Height of the window in GUI
private double time;
private int numOfQ; // The number of queries(for terminal mode)
private double[][] acts;
private Cell[] cells;
private int size;
private final double error = 0.0001;

/*
type = 0 -> GUI mode
type = 1 -> Terminal mode
*/
public static int mode1 = 0;
```

Run方法是用来控制程序运行进程的主要方法，分为两种，run1和run2，run1是用来控制gui输出，run2是用来控制terminal输出，两种方法的不同其实也就是绘制和打印的区别。其大致内容为：利用两个for循环控制1/15s，然后调用move方法移动细胞，count多种方法来计算细胞感知范围内的不同颜色的细胞数量，clear清空画布，调用change方法改变细胞颜色，重绘细胞，并且显示出来。

run1方法相应代码如下所示：

```
public void run1(CellSystem cellSystem) { // GUI mode -> 一直不停
    for (double t1 = 0; true; t1++) {
        for (double t2 = 1; t2 <= 15; t2++) {

//                try {
//                    Thread.sleep(1000);
//                } catch (InterruptedException e) {
```

```
//                    e.printStackTrace();
//                }

            time = t1 + t2 / 15.0;

            // Move cell[i] to new position.
            for (Cell c : cells) {
                cellSystem.move(c);
            }

            // Scan cell[i]'s perception range, calculate the number of cells
            for (Cell c : cells) {
                c.setNumOfC(countAll(c));
                if (c.getColor() == 'r') { // For red cells
                    c.setNumOfR(countR(c));
                    c.setNumOfY(countY(c));
                } else if (c.getColor() == 'g') { // For green cells
                    c.setNumOfG(countG(c));
                    c.setNumOfR(countR(c));
                } else if (c.getColor() == 'b') { // For blue cells
                    c.setNumOfB(countB(c));
                    c.setNumOfG(countG(c));
                } else if (c.getColor() == 'y') { // For yellow cells
                    c.setNumOfY(countY(c));
                    c.setNumOfB(countB(c));
                }
            }

            // Change cell[i]'s color with the rules if necessary
            StdDraw.clear();
            for (Cell c : cells) {
                c.change();
                StdDraw.setPenColor(c.Color());
                StdDraw.filledCircle(c.getxPosition(), c.getyPosition(),
c.getRadius());
            }
            StdDraw.show();

        }
    }
}
```

run2方法相应代码如下所示：

```
public void run2(CellSystem cellSystem) { // Terminal mode
    for (double t1 = 0; true; t1++) {
        for (double t2 = 1; t2 <= 15; t2++) {
            time = t1 + t2 / 15.0;
```

```
            if (time > acts[numOfQ - 1][0]) {
                break;
            }

            // Move cell[i] to new position.
            for (Cell c : cells) {
                cellSystem.move(c);
            }

            // Scan cell[i]'s perception range, calculate the number of cells
            for (Cell c : cells) {
                c.setNumOfC(countAll(c));
                if (c.getColor() == 'r') { // For red cells
                    c.setNumOfR(countR(c));
                    c.setNumOfY(countY(c));
                } else if (c.getColor() == 'g') { // For green cells
                    c.setNumOfG(countG(c));
                    c.setNumOfR(countR(c));
                } else if (c.getColor() == 'b') { // For blue cells
                    c.setNumOfB(countB(c));
                    c.setNumOfG(countG(c));
                } else if (c.getColor() == 'y') { // For yellow cells
                    c.setNumOfY(countY(c));
                    c.setNumOfB(countB(c));
                }
            }

            // Change cell[i]'s color with the rules if necessary
            for (Cell c : cells) {
                c.change();
            }

            new File("src\\output").mkdir();
            try (PrintWriter fout = new
PrintWriter("src\\output\\output.txt")) {
                for (int i = 0; i < numOfQ; i++) {
                    if (Math.abs(time - acts[i][0]) < 0.0001) {
                        fout.println(cells.get((int) acts[i][1]).toString());
                        System.out.println(cells.get((int) acts[i]
[1]).toString());
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
        if (time > acts[numOfQ - 1][0] || Math.abs(time - acts[numOfQ - 1][0])
< 0.0001){
            break;
        }
    }
}
```

方法move是用来移动细胞到下一个位置，首先判断传入细胞的颜色，然后判断isStop方法和isToEdge方法的返回值，若均返回false则说明细胞下一步既不会碰壁也不会撞到别的细胞，则相应的x，y值做相应改动；若有一个返回true，则调用increment方法判断具体的移动距离，然后对x，y值做相应改动。相应代码如下所示：

```
// Cell moves to next place
public void move(Cell cell) {
    if (cell.getColor() == 'r') { // Upward
        if (!isStop(cell) && !cell.isToEdge(width, height)) {
            cell.setyPosition(cell.getyPosition() + (double) 1 / 15);
        } else {
            cell.setyPosition(cell.getyPosition() + increment(cell));
        }
    } else if (cell.getColor() == 'g') { // Downward
        if (!isStop(cell) && !cell.isToEdge(width, height)) {
            cell.setyPosition(cell.getyPosition() - (double) 1 / 15);
        } else {
            cell.setyPosition(cell.getyPosition() - increment(cell));
        }
    } else if (cell.getColor() == 'b') { // Left
        if (!isStop(cell) && !cell.isToEdge(width, height)) {
            cell.setxPosition(cell.getxPosition() - (double) 1 / 15);
        } else {
            cell.setxPosition(cell.getxPosition() - increment(cell));
        }
    } else if (cell.getColor() == 'y') { // Right
        if (!isStop(cell) && !cell.isToEdge(width, height)) {
            cell.setxPosition(cell.getxPosition() + (double) 1 / 15);
        } else {
            cell.setxPosition(cell.getxPosition() + increment(cell));
        }
    }
}
```

方法isStop是用来判断目标细胞在下一步移动中是否会碰到阻碍细胞，我们利用一个for循环遍历所有细胞，首先判定为非目标细胞，然后调用Cell类里面的isIn方法来判断遍历细胞是否位于目标细胞的移动路径中，若存在一个细胞isIn判定返回true，则我们返回true，意为目标细胞在下一步移动中会碰到阻碍细胞，否则则返回false。相应代码如下所示：

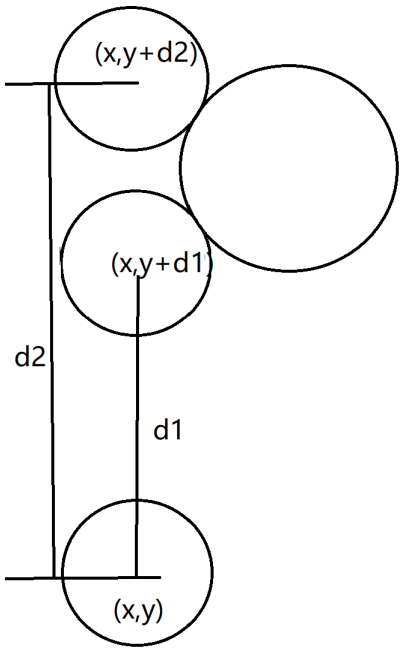
```
/*
```

```
Decide if cell will meet other cells in its path
cell -> 移动的细胞
*/
public boolean isStop(Cell cell) {
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isIn(cell)) {
                return true;
            }
        }
    }
    return false;
}
```

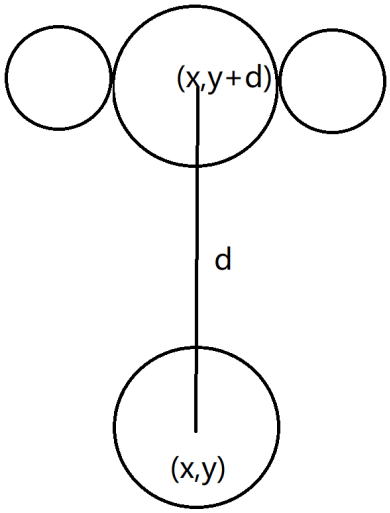
increment方法用于计算当细胞无法正常走完1/15距离时所移动的距离，即此时细胞移动路径上可能会与其他细胞发生碰撞或是会触碰窗口边界。

在此方法中，以红色细胞作为运动细胞为例，我们假设运动细胞的坐标为(x, y)，在该细胞移动路径上的另一细胞的坐标为(x1, y1)，运动细胞最终停止的位置为(x, y+d)，d即为细胞移动的增量。当满足$(x_1 - x)^2 + [y_1 - (y + d)]^2 = (r_1 + r)^2$时，运动细胞和路径上的细胞相切，将上式化简可得$d^2 + (2y - 2y_1)d + [(x_1-x)^2 + (y_1 - y)^2 - (r_1+r)^2] = 0$，此时我们需通过该一元二次方程求出d的解。由一元二次方程的性质可知：

当delta>0时，细胞移动距离d有两个解，如图所示：



当delta=0时，细胞移动距离d仅有一个解，如图所示：



故当delta>0时，我们需要选取较小的那个解，若选取较大的解，则对应着在实际情况中两细胞会发生重叠，违背了规则；当delta=0时，d即为我们想求得的增量；delta<0时，两细胞相离，无解。除此之外，在运动细胞的移动路径中可能会存在多个其他细胞，此时我们通过比较d和dmin找出最短移动路径。

若先发生细胞与窗口边界相接触，则d为细胞与窗口边界相切时所移动的距离。

相应代码如下所示：

```
 /*
(x, y), r: 入参细胞(移动中的细胞)
(x_1, y_1), r_1: 被创的那个细胞
d^2 + (2y - 2y_1)d + [(x_1 - x)^2 + (y_1 + y)^2 - (r_1 + r)^2] = 0

delta < 0 -> d无解 -> 两圆相离(不会出现相切) -> d = 1/15
delta = 0 -> d一解
delta > 0 -> d两解 -> 一正一负 -> 输出正确移动路径上对应的d
*/
public double increment(Cell cell) {
    double dmin = (double) 1 / 15;
    double d = (double) 1 / 15;
    double x = cell.getxPosition();
    double y = cell.getyPosition();
    double r = cell.getRadius();

    for (Cell c : cells) {
        if (c != cell) {
            if (c.isIn(cell)) {
                if (cell.getColor() == 'r') { // Upward
                    double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getxPosition() - x, 2));
                    if (delta > 0) {
                        d = c.getyPosition() - y - Math.sqrt(delta) / 2;
                    } else if (Math.abs(delta) < error) {
```

```
                        d = c.getyPosition() - y;
                    }
                    if (d < dmin) {
                        dmin = d;
                    }
                } else if (cell.getColor() == 'g') { // Downward
                    double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getxPosition() - x, 2));
                    if (delta > 0) {
                        d = y - c.getyPosition() - Math.sqrt(delta) / 2;
                    } else if (Math.abs(delta) < error) {
                        d = y - c.getyPosition();
                    }
                    if (d < dmin) {
                        dmin = d;
                    }
                } else if (cell.getColor() == 'b') { // Left
                    double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getyPosition() - y, 2));
                    if (delta > 0) {
                        d = x - c.getxPosition() - Math.sqrt(delta) / 2;
                    } else if (Math.abs(delta) < error) {
                        d = x - c.getxPosition();
                    }
                    if (d < dmin) {
                        dmin = d;
                    }
                } else if (cell.getColor() == 'y') { // Right
                    double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getyPosition() - y, 2));
                    if (delta > 0) {
                        d = c.getxPosition() - x - Math.sqrt(delta) / 2;
                    } else if (Math.abs(delta) < error) {
                        d = c.getxPosition() - x;
                    }
                    if (d < dmin) {
                        dmin = d;
                    }
                }
            } else if (cell.isToEdge(width, height)) {
                if (cell.getColor() == 'r') { // Upward
                    d = height - cell.getRadius() - cell.getyPosition();
                    if (d < dmin) {
                        dmin = d;
                    }
                } else if (cell.getColor() == 'g') { // Downward
                    d = cell.getyPosition() - cell.getRadius();
                    if (d < dmin) {
```

```
                        dmin = d;
                    }
                } else if (cell.getColor() == 'b') { // Left
                    d = cell.getxPosition() - cell.getRadius();
                    if (d < dmin) {
                        dmin = d;
                    }
                } else if (cell.getColor() == 'y') { // Right
                    d = width - cell.getRadius() - cell.getxPosition();
                    if (d < dmin) {
                        dmin = d;
                    }
                }
            }
        }
    }
return dmin;
}
```

在CellSystem中通过遍历所有细胞，通过countAll方法计算周围细胞的总数，分别通过countR、countG、countB、countY计算周围红色、绿色、蓝色、黄色的细胞数量。相应代码如下所示：

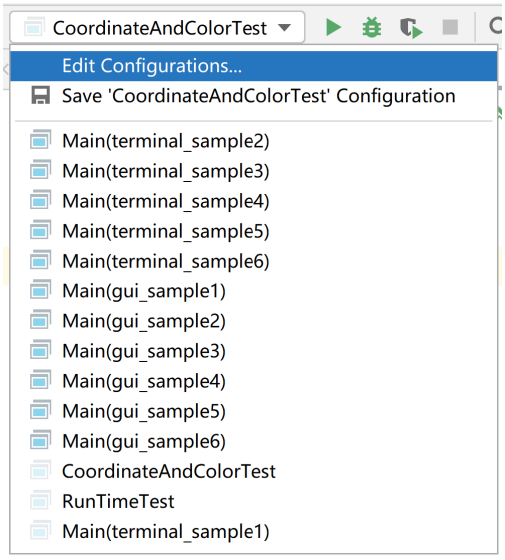
```
// Count the number of all cells in the perception range
public int countAll(Cell cell) {
    int count = 0;
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isPerceived(cell)) {
                count++;
            }
        }
    }
    return count;
}

// Count the number of red cells in the perception range
public int countR(Cell cell) {
    int count = 0;
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isPerceived(cell) && c.getColor() == 'r') {
                count++;
            }
        }
    }
    return count;
}
```

```
// Count the number of green cells in the perception range
public int countG(Cell cell) {
    int count = 0;
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isPerceived(cell) && c.getColor() == 'g') {
                count++;
            }
        }
    }
    return count;
}

// Count the number of blue cells in the perception range
public int countB(Cell cell) {
    int count = 0;
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isPerceived(cell) && c.getColor() == 'b') {
                count++;
            }
        }
    }
    return count;
}

// Count the number of yellow cells in the perception range
public int countY(Cell cell) {
    int count = 0;
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isPerceived(cell) && c.getColor() == 'y') {
                count++;
            }
        }
    }
    return count;
}
```

Redirect the StdIn

在Main方法中，我们还使用了重定向标准输入的方式读取文件，相应代码如下所示：

```
String[] data = StdIn.readAllStrings();
```



Performance optimization

在初版move方法中，isStop方法和increment方法都是通过遍历所有的细胞来找到移动路径中是否有阻碍细胞并且计算出阻碍细胞到移动细胞之间的距离，显然，这里做了很多无用功。在优化中，我们更改了isStop方法中返回值，使其返回一个`array<cell>`，在找到一个阻碍细胞后就将其加入array中，通过increment导入array，这样就可以将两个遍历减少一个。

然后，通过RunTimeTest，我们得到了利用两种版本多次运行sample2的所花时间，可以看出，运行时间有所减少。相应代码如下所示：

```
// Cell moves to next place
public void move(Cell cell) {
    ArrayList<Cell> isInCell=isStop(cell);
    if (cell.getColor() == 'r') { // Upward
        if (isInCell.size()==0 && !cell.isToEdge(width, height)) {
            cell.setyPosition(cell.getyPosition() + (double) 1 / 15);
        } else {
            cell.setyPosition(cell.getyPosition() + increment(cell,isInCell));
        }
    } else if (cell.getColor() == 'g') { // Downward
        if (isInCell.size()==0 && !cell.isToEdge(width, height)) {
            cell.setyPosition(cell.getyPosition() - (double) 1 / 15);
        } else {
            cell.setyPosition(cell.getyPosition() - increment(cell,isInCell));
        }
    } else if (cell.getColor() == 'b') { // Left
        if (isInCell.size()==0 && !cell.isToEdge(width, height)) {
            cell.setxPosition(cell.getxPosition() - (double) 1 / 15);
        } else {
            cell.setxPosition(cell.getxPosition() - increment(cell,isInCell));
```

```
        }
    } else if (cell.getColor() == 'y') { // Right
        if (isInCell.size()==0 && !cell.isToEdge(width, height)) {
            cell.setxPosition(cell.getxPosition() + (double) 1 / 15);
        } else {
            cell.setxPosition(cell.getxPosition() + increment(cell,isInCell));
        }
    }
}

/*
Decide if cell will meet other cells in its path
cell -> 移动的细胞
*/
public ArrayList<Cell> isStop(Cell cell) {
    ArrayList<Cell> isInCell=new ArrayList<>();
    for (Cell c : cells) {
        if (c != cell) {
            if (c.isIn(cell)) {
                isInCell.add(c);
            }
        }
    }
    return isInCell;
}
/*
(x, y), r: 入参细胞(移动中的细胞)
(x_1, y_1), r_1: 被创的那个细胞
d^2 + (2y - 2y_1)d + [(x_1 - x)^2 + (y_1 + y)^2 - (r_1 + r)^2] = 0

delta < 0 -> d无解 -> 两圆相离(不会出现相切) -> d = 1/15
delta = 0 -> d一解 ->
delta > 0 -> d两解 -> 一正一负 -> 输出正确移动路径上对应的d
*/
public double increment(Cell cell,ArrayList<Cell> arrayList) {
    double dmin = (double) 1 / 15;
    double d = (double) 1 / 15;
    double x = cell.getxPosition();
    double y = cell.getyPosition();
    double r = cell.getRadius();
    char color=cell.getColor();
    ArrayList<Cell> isInCell=arrayList;
    if(color=='r') {
        if (isInCell.size() != 0) {
            for (Cell c : isInCell) {
                double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getxPosition() - x, 2));
                if (delta > 0) {
```

```
                    d = c.getyPosition() - y - Math.sqrt(delta) / 2;
                } else if (Math.abs(delta) < error) {
                    d = c.getyPosition() - y;
                }
                if (d < dmin) {
                    dmin = d;
                }
            }
        } else {
            d = height - cell.getRadius() - cell.getyPosition();
            if (d < dmin) {
                dmin = d;
            }
        }
    } else if(color=='g') {
        if (isInCell.size() != 0) {
            for (Cell c : isInCell) {
                double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getxPosition() - x, 2));
                if (delta > 0) {
                    d = y - c.getyPosition() - Math.sqrt(delta) / 2;
                } else if (Math.abs(delta) < error) {
                    d = y - c.getyPosition();
                }
                if (d < dmin) {
                    dmin = d;
                }
            }
        } else {
            d = cell.getyPosition() - cell.getRadius();
            if (d < dmin) {
                dmin = d;
            }
        }
    } else if(color=='b') {
        if (isInCell.size() != 0) {
            for (Cell c : isInCell) {
                double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getyPosition() - y, 2));
                if (delta > 0) {
                    d = x - c.getxPosition() - Math.sqrt(delta) / 2;
                } else if (Math.abs(delta) < error) {
                    d = x - c.getxPosition();
                }
                if (d < dmin) {
                    dmin = d;
                }
            }
```

```
        } else {
            d = cell.getxPosition() - cell.getRadius();
            if (d < dmin) {
                dmin = d;
            }
        }
    } else if(color=='y') {
        if (isInCell.size() != 0) {
            for (Cell c : isInCell) {
                double delta = 4 * (Math.pow(c.getRadius() + r, 2) -
Math.pow(c.getyPosition() - y, 2));
                if (delta > 0) {
                    d = c.getxPosition() - x - Math.sqrt(delta) / 2;
                } else if (Math.abs(delta) < error) {
                    d = c.getxPosition() - x;
                }
                if (d < dmin) {
                    dmin = d;
                }
            }
        } else {
            d = width - cell.getRadius() - cell.getxPosition();
            if (d < dmin) {
                dmin = d;
            }
        }
    }
    return dmin;
}
```

初版：

```
Time spend: 5.3860000000 seconds
Time spend: 5.6950000000 seconds
Time spend: 3.6150000000 seconds
Time spend: 3.3850000000 seconds
Time spend: 2.9000000000 seconds
Time spend: 2.8470000000 seconds
Time spend: 2.7670000000 seconds
Time spend: 2.7970000000 seconds
Time spend: 2.9720000000 seconds
```

改进后：

```
Time spend: 4.0580000000 seconds
Time spend: 3.0060000000 seconds
Time spend: 3.4260000000 seconds
Time spend: 3.4390000000 seconds
Time spend: 3.7240000000 seconds
Time spend: 5.5510000000 seconds
Time spend: 5.3420000000 seconds
Time spend: 4.2300000000 seconds
Time spend: 3.4940000000 seconds
```

Program test

运行时间测试

在本节中，我们使用RunTimeTest类，通过计算程序运行10次每次分别所消耗的时间并进行对比，从而比较得出两种程序的性能。测试程序运行时间的代码如下：

```
import edu.princeton.cs.algs4.In;
import edu.princeton.cs.algs4.Stopwatch;

public class RunTimeTest {
    public static void main(String[] args) {
        int x = 3;// samplex.txt
        int sys = 0;// 运行第sys个系统
        for (int i = x; i <= x; i++) {
            In fin = new In("C:\\Users\\Beauty of
football\\IdeaProjects\\RGBYCellLife\\src\\sample\\sample" + i + ".txt");
            String[] data = fin.readAllStrings();
            fin.close();

            Stopwatch timer = new Stopwatch();

            double[] time = new double[10];
            double now = 0;
            double last;

            switch (sys) {
                case 0:
                    // CellSystem
                    if (args.length != 0 && args[0].equals("terminal")) {
                        CellSystem.mode = 1;
                    }

                    CellSystem system = new CellSystem(data);
```

```
                    if (CellSystem.mode == 1) { // Terminal mode
                        for (int j = 0; j < 10; j++) {
                            last = now;
                            now = timer.elapsedTime();
                            system.run2(system);
                            time[j] = now - last;
                        }
                    } else { // GUI mode
                        system.run1(system);
                    }
                    break;
                case 1:
                    // CellSystem1
                    if (args.length != 0 && args[0].equals("terminal")) {
                        CellSystem1.mode1 = 1;
                    }

                    CellSystem1 system1 = new CellSystem1(data);
                    if (CellSystem1.mode1 == 1) { // Terminal mode
                        for (int j = 0; j < 10; j++) {
                            last = now;
                            now = timer.elapsedTime();
                            system1.run2(system1);
                            time[j] = now - last;
                        }
                    } else { // GUI mode
                        system1.run1(system1);
                    }
            }

            for (double t : time) {
                System.out.printf("Time spend: %.10f seconds\n", t);
            }
        }
    }
}
```

使用LinkedList存储Cell

Sample1

实验次数	1	2	3	4	5	6	7	8	9	10
运行时间(s)	0.004	0.328	0.187	0.151	0.156	0.175	0.194	0.180	0.166	0.140

平均运行时间 = 0.1863s

由于第一次实验的结果与剩余九次结果的偏差较大，故我们将第一次实验的结果舍去，计算剩余九次实验结果的平均值，并将其作为本次测试的最终结果。

Sample2

实验次数	1	2	3	4	5	6	7	8	9	10
运行时间(s)	0.032	3.109	3.306	3.468	3.027	2.799	3.339	3.112	2.977	3.274

平均运行时间 = 3.1568s

由于第一次实验的结果与剩余九次结果的偏差较大，故我们将第一次实验的结果舍去，计算剩余九次实验结果的平均值，并将其作为本次测试的最终结果。

Sample3

实验次数	1	2	3	4	5
运行时间(s)	0.082	19.717	15.483	16.776	16.616
实验次数	**6**	**7**	**8**	**9**	**10**
运行时间(s)	16.762	16.517	17.244	17.179	17.739

平均运行时间 = 17.1148s

由于第一次实验的结果与剩余九次结果的偏差较大，故我们将第一次实验的结果舍去，计算剩余九次实验结果的平均值，并将其作为本次测试的最终结果。

使用Array存储Cell

Sample1

实验次数	1	2	3	4	5	6	7	8	9	10
运行时间(s)	0.003	0.221	0.142	0.166	0.145	0.164	0.156	0.171	0.139	0.111

平均运行时间 = 0.1572s

由于第一次实验的结果与剩余九次结果的偏差较大，故我们将第一次实验的结果舍去，计算剩余九次实验结果的平均值，并将其作为本次测试的最终结果。

Sample2

实验次数	1	2	3	4	5	6	7	8	9	10
运行时间(s)	0.041	2.739	2.308	2.514	2.547	2.572	3.026	2.457	2.714	2.969

平均运行时间 = 2.6496s

由于第一次实验的结果与剩余九次结果的偏差较大，故我们将第一次实验的结果舍去，计算剩余九次实验结果的平均值，并将其作为本次测试的最终结果。

Sample3

实验次数	1	2	3	4	5
运行时间(s)	0.072	16.880	14.365	16.111	15.981
实验次数	**6**	**7**	**8**	**9**	**10**
运行时间(s)	16.344	15.745	15.817	16.254	16.201

平均运行时间 = 15.9664s

由于第一次实验的结果与剩余九次结果的偏差较大，故我们将第一次实验的结果舍去，计算剩余九次实验结果的平均值，并将其作为本次测试的最终结果。

结论

通过对比使用LinkedList存储Cell和使用Array存储Cell的两种方式在各测试样例下的运行时间结果的平均值可以得出，使用Array存储Cell的方式可以在很小程度上加快运行速度，减少程序运行时间。

坐标误差测试

在本节中，我们使用CoordinateAndColorTest类，通过计算程序运行结果与所给的测试样例输出细胞的x和y坐标结果之间的误差，并得出所有误差中的最大误差，进而对程序的运行情况进行测试和评估。测试方法的代码如下：

```
public static void errorRateTest(int n) {
    In fin = new In("C:\\Users\\Beauty of football\\OneDrive - 南方科技大学\\桌面
\\大二春季学期\\数据结构与算法分析B\\Proj\\output\\output.txt");
    String[] out = fin.readAllStrings();
    fin.close();

    In f = new In("C:\\Users\\Beauty of football\\OneDrive - 南方科技大学\\桌面
\\大二春季学期\\数据结构与算法分析B\\Proj\\sample\\sample" + n + "_out.txt");
    String[] sample = f.readAllStrings();
    f.close();

    double errorXMax = 0;
    double errorYMax = 0;
    double errorX;
    double errorY;

    for (int i = 0, j = 1; i < out.length / 3 && i < sample.length / 3; i +=
3, j += 3) {
        errorX = Math.abs((Double.parseDouble(out[i]) -
Double.parseDouble(sample[i])) / Double.parseDouble(sample[i]));
        if (errorX > errorXMax) {
            errorXMax = errorX;
        }
        errorY = Math.abs((Double.parseDouble(out[j]) -
Double.parseDouble(sample[j])) / Double.parseDouble(sample[j]));
        if (errorY > errorYMax) {
            errorYMax = errorY;
        }
    }

    System.out.printf("Maximum error rate of xPosition: %.5f\n", errorXMax);
    System.out.printf("Maximum error rate of yPosition: %.5f\n", errorYMax);
```

```
}
```

使用LinkedList存储Cell

Sample1

在对Sample1的测试中，我们得出输出访问细胞的x坐标无误差，y坐标也无误差，结果如下图所示：

```
Maximum error rate of xPosition: 0.00000
Maximum error rate of yPosition: 0.00000
```

Sample2

在对Sample2的测试中，我们得出输出访问细胞的x坐标无误差，y坐标也无误差，结果如下图所示：

```
Maximum error rate of xPosition: 0.00000
Maximum error rate of yPosition: 0.00000
```

Sample3

在对Sample3的测试中，我们得出输出访问细胞的x坐标的最大误差为3.251%，y坐标的最大误差为3.81%，结果如下图所示：

```
Maximum error rate of xPosition: 0.03251
Maximum error rate of yPosition: 0.03810
```

使用Array存储Cell

Sample1

在对Sample1的测试中，我们得出输出访问细胞的x坐标无误差，y坐标也无误差，结果如下图所示：

```
Maximum error rate of xPosition: 0.00000
Maximum error rate of yPosition: 0.00000
```

Sample2

在对Sample2的测试中，我们得出输出访问细胞的x坐标无误差，y坐标也无误差，结果如下图所示：

```
Maximum error rate of xPosition: 0.00000
Maximum error rate of yPosition: 0.00000
```

Sample3

在对Sample3的测试中，我们得出输出访问细胞的x坐标的最大误差为3.251%，y坐标的最大误差为3.81%，结果如下图所示：

```
Maximum error rate of xPosition: 0.03251
Maximum error rate of yPosition: 0.03810
```

结论

对比两种存储方式，输出结果相同，误差相同。在本测试中，对于Sample1和Sample2的坐标误差测试所得的结果与测试样例输出之间的误差为0；对于Sample3的坐标误差测试所得的结果与测试样例输出之间的x坐标的最大误差均为3.251%，y坐标的最大误差均为3.81%，满足误差小于5%的要求。

颜色误差测试

在本节中，我们使用CoordinateAndColorTest类，通过统计程序运行结果与所给的测试样例输出细胞之间对应细胞颜色不相同的细胞个数，来得出输出细胞颜色的错误率，进而对程序的运行情况进行测试和评估。测试方法的代码如下：

```
public static void countError(int n) {
    In fin = new In("C:\\Users\\Beauty of football\\OneDrive - 南方科技大学\\桌面
\\大二春季学期\\数据结构与算法分析B\\Proj\\output\\output.txt");
    String[] out = fin.readAllStrings();
    fin.close();

    In f = new In("C:\\Users\\Beauty of football\\OneDrive - 南方科技大学\\桌面
\\大二春季学期\\数据结构与算法分析B\\Proj\\sample\\sample" + n + "_out.txt");
    String[] sample = f.readAllStrings();
    f.close();

    int count = 0;
    char colorOut;
    char colorSam;

    for (int i = 0, j = 2; i < out.length / 3 && i < sample.length / 3; i +=
3, j += 3) {
        colorOut = out[j].charAt(0);
        colorSam = sample[j].charAt(0);

        if (colorOut != colorSam) {
            count++;
        }
    }
    System.out.printf("Number of error color: %d\n", count);
}
```

使用LinkedList存储Cell

Sample1

在对Sample1的测试中，我们得出输出访问细胞的颜色错误个数为0，错误率为0，结果如下图所示：

```
Number of error color: 0
```

Sample2

在对Sample2的测试中，我们得出输出访问细胞的颜色错误个数为515，错误率为1.72%，结果如下图所示：

```
Number of error color: 515
```

Sample3

在对Sample3的测试中，我们得出输出访问细胞的颜色错误个数为0，错误率为0，结果如下图所示：

```
Number of error color: 0
```

使用Array存储Cell

Sample1

在对Sample1的测试中，我们得出输出访问细胞的颜色错误个数为0，错误率为0，结果如下图所示：

```
Number of error color: 0
```

Sample2

在对Sample2的测试中，我们得出输出访问细胞的颜色错误个数为515，错误率为1.72%，结果如下图所示：

```
Number of error color: 515
```

Sample3

在对Sample3的测试中，我们得出输出访问细胞的颜色错误个数为0，错误率为0，结果如下图所示：

```
Number of error color: 0
```

结论

对比两种存储方式，输出结果相同，误差相同。在本测试中，对于Sample1和Sample3的颜色误差测试所得的结果与测试样例输出之间的误差为0，即两输出结果中相应各细胞颜色均相同；对于Sample2的颜色误差测试所得的结果与测试样例输出之间的误差为1.72%，满足误差小于5%的要求。

Bonus

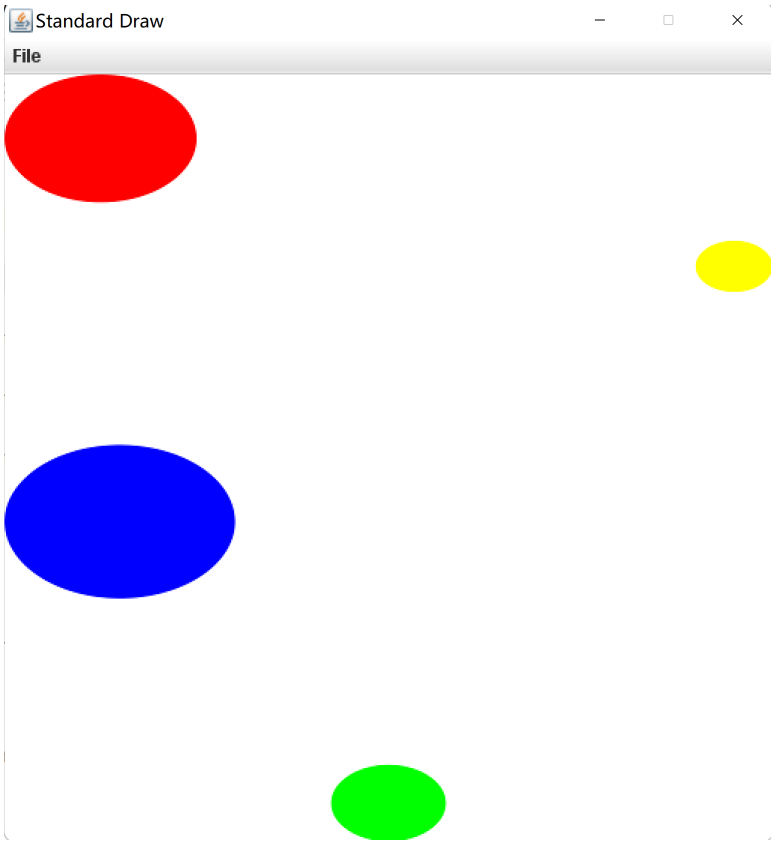
Self-designed test data

Sample4

Test Data

```
4 6
4
0.5 2 0.5 6.1 r
2 0.5 0.3 5 g
1 4.5 0.2 3 y
3.6 2.5 0.6 2 b
4
1 1
2 3
3 2
4 0
```

GUI输出结果



四个不同颜色的细胞在运动过程中不会相碰且不改变颜色，最终四个细胞会因到达窗口边界而停止运动。

Terminal输出结果

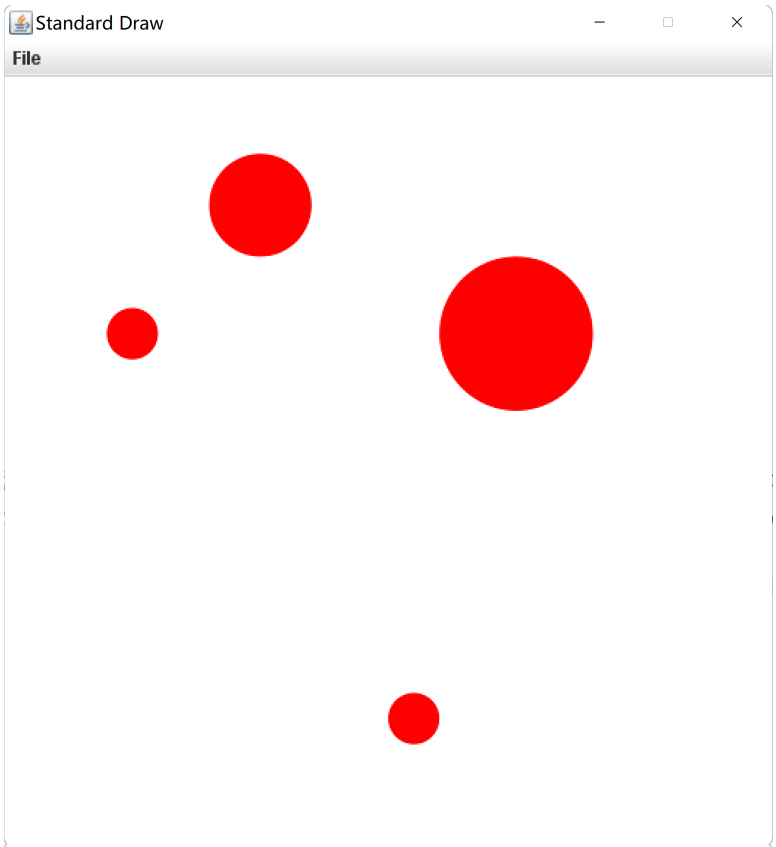
```
2.0 1.0000000000000002 g
1.5999999999999954 2.5 b
3.8 4.5 y
0.5 5.5 r
```

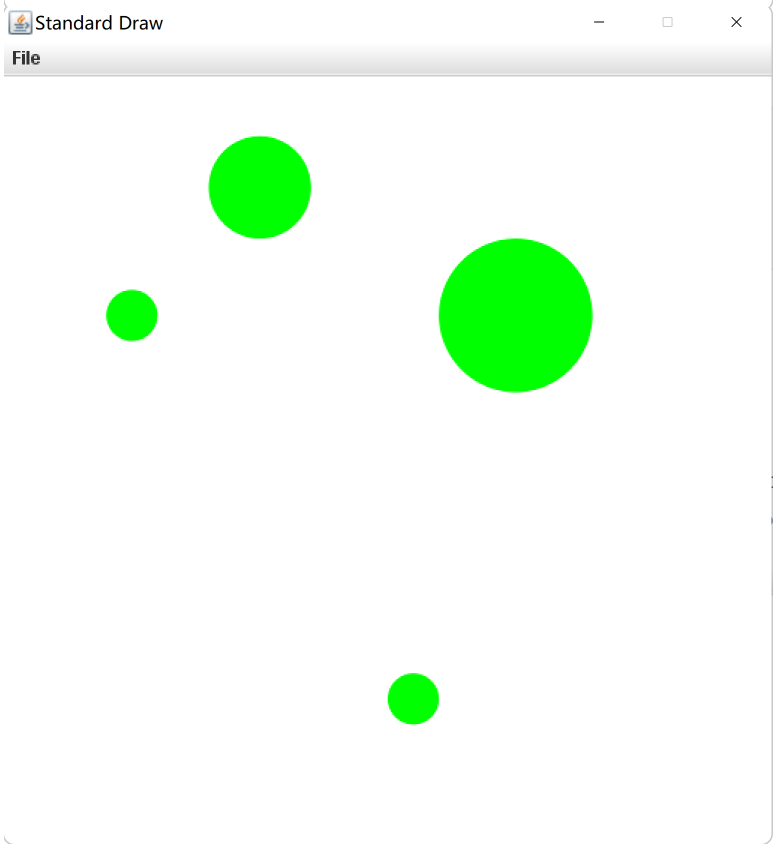
Sample5

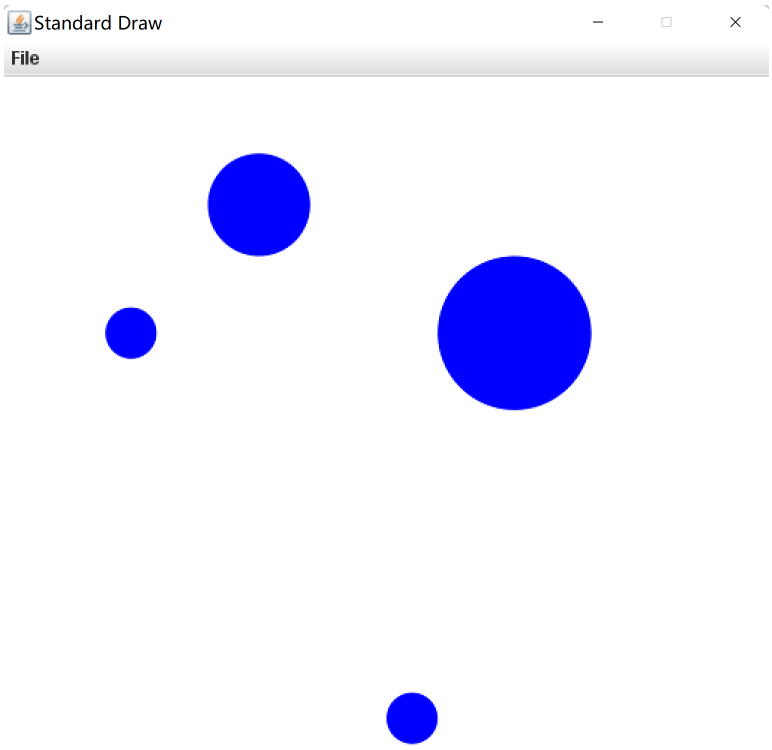
Test Data

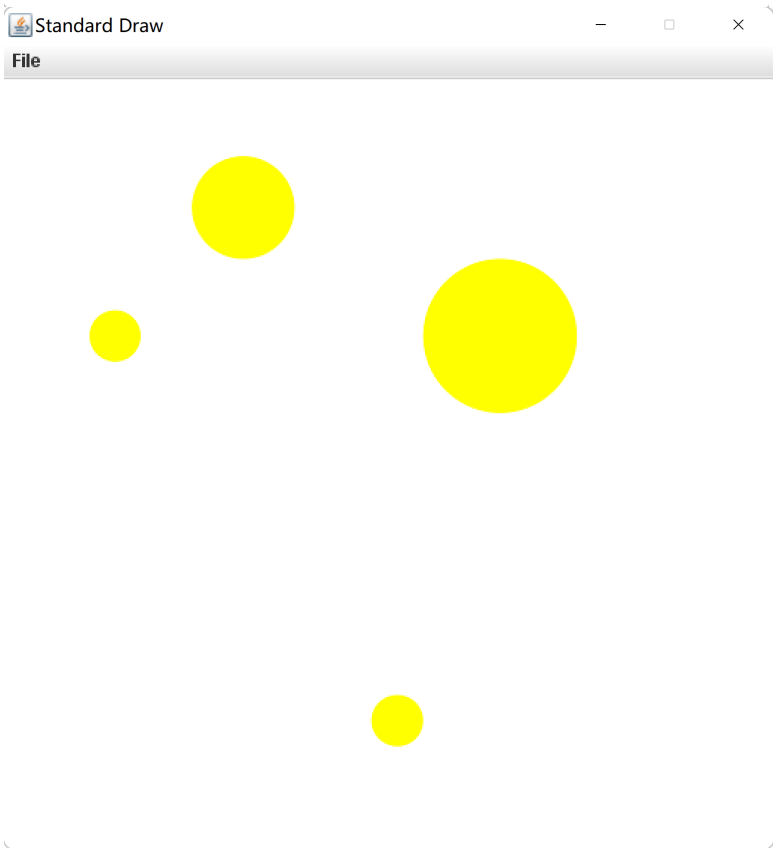
```
3 3
4
0.5 2 0.1 6.1 r
2 2 0.3 5 r
1 2.5 0.2 5.2 r
1.6 0.5 0.1 6 r
4
1 1
2 3
3 2
4 0
```

GUI输出结果


Standard Draw
File


Standard Draw
File





在Sample5输入情况下的GUI输出界面中会发生有趣的效果。四个相同颜色的细胞会按照红色、绿色、蓝色和黄色的顺序不停地改变自己的颜色，并根据当前自己的颜色同步向上、向下、向左、向右动。

Terminal输出结果

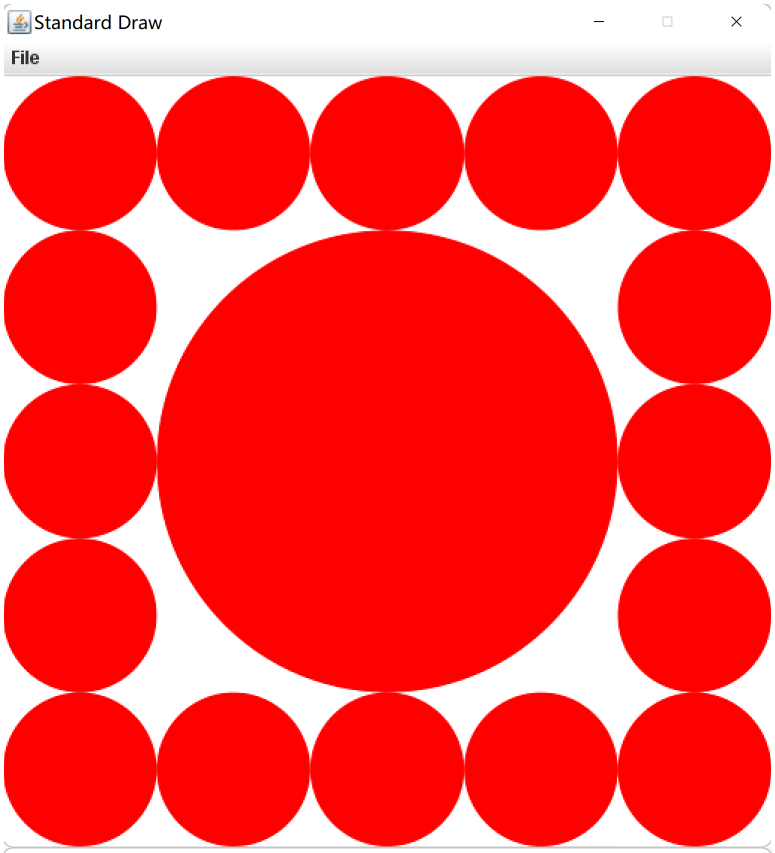
```
1.9333333333333333 2.0 y
1.6 0.5 b
1.0 2.566666666666667 g
0.5 2.0 r
```

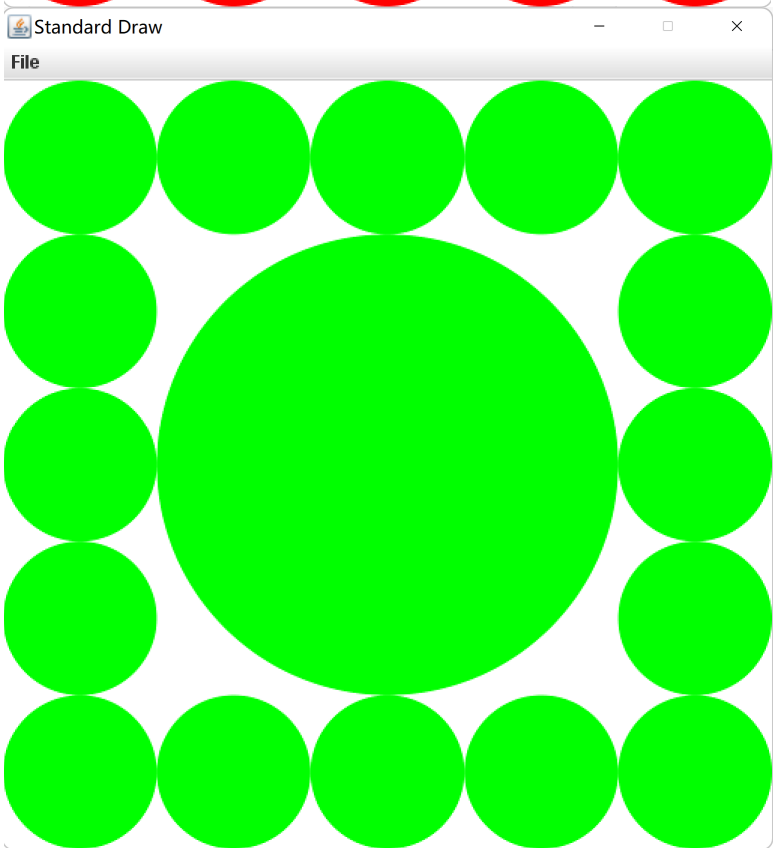
Sample6

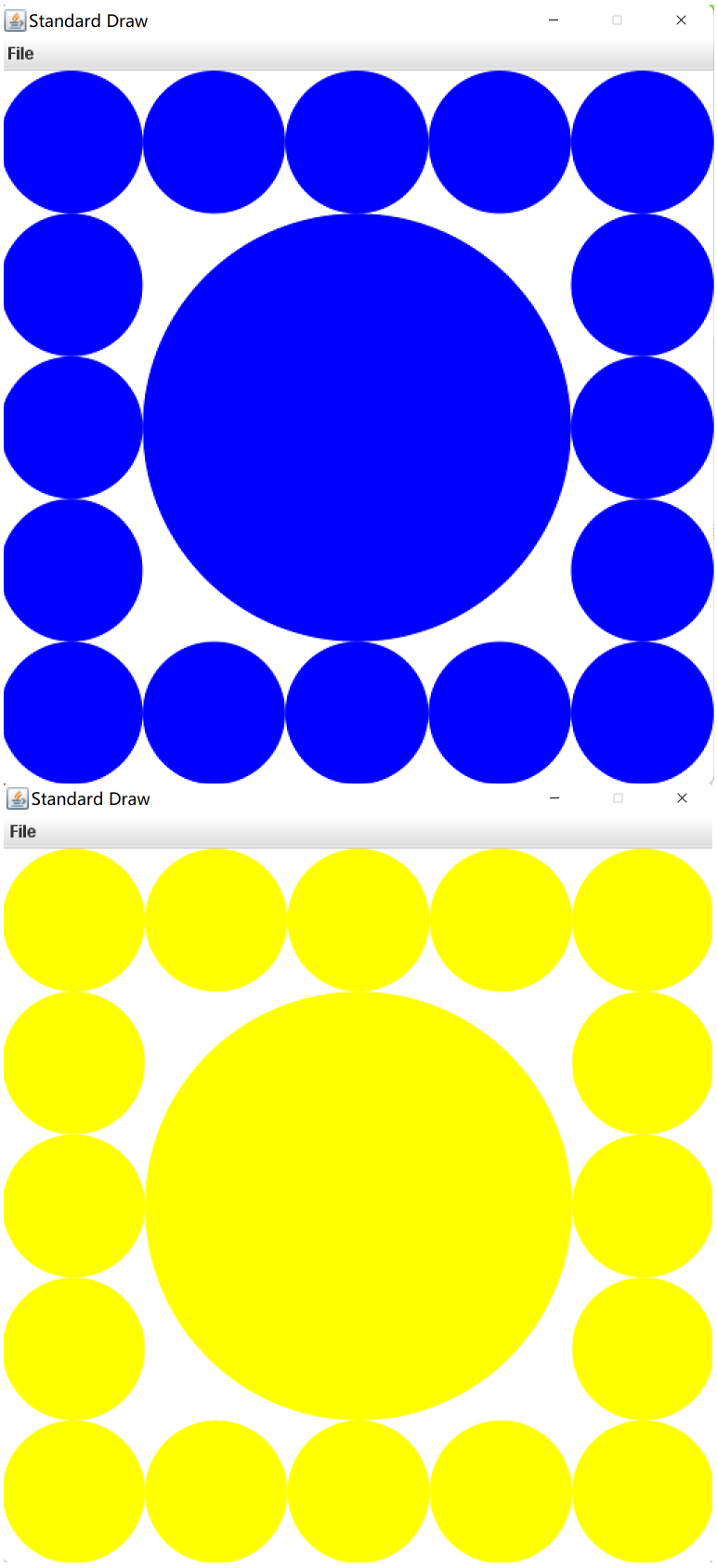
Test Data

```
5 5
17
2.5 2.5 1.5 5 r
2.5 4.5 0.5 3 r
1.5 4.5 0.5 3 r
0.5 4.5 0.5 3 r
3.5 4.5 0.5 3 r
4.5 4.5 0.5 3 r
2.5 0.5 0.5 3 r
1.5 0.5 0.5 3 r
0.5 0.5 0.5 3 r
3.5 0.5 0.5 3 r
4.5 0.5 0.5 3 r
0.5 2.5 0.5 3 r
0.5 3.5 0.5 3 r
0.5 1.5 0.5 3 r
4.5 2.5 0.5 3 r
4.5 3.5 0.5 3 r
4.5 1.5 0.5 3 r
10
1 1
2 3
3 2
4 0
5 16
6 11
7 9
8 14
9 7
10 6
```

GUI输出结果







在Sample6输入情况下的GUI输出界面中，所有相同颜色的细胞会按照红色、绿色、蓝色和黄色的顺序在原地不停地改变自己的颜色。

Terminal输出结果

```
2.5 4.5 y
0.5 4.5 b
1.5 4.5 g
2.5 2.5 r
4.5 1.5 y
0.5 2.5 b
3.5 0.5 g
4.5 2.5 r
1.5 0.5 y
2.5 0.5 b
```