

**JAVA大作业**

**题目：**  屏保（气泡）

物联网工程 **学院** 计算机科学与技术专业 **专业**

学 号： 1030414414

学生姓名： 阎覃

班 级： 计科1404

成 绩：

**2016年 6月**

# 目录

[目录 2](#_Toc453460977)

[第一章 作业要求 4](#_Toc453460978)

[第二章 程序简介 5](#_Toc453460979)

[2.1主要文件 5](#_Toc453460980)

[2.1.1 MainFrame.java 5](#_Toc453460981)

[2.1.2 SettingDialog.java 5](#_Toc453460982)

[2.1.3 PaintingPanel.java 6](#_Toc453460983)

[2.1.4 DebugPanel.java 6](#_Toc453460984)

[2.1.5 Ball.java 7](#_Toc453460985)

[2.1.6 Bubbles.java 7](#_Toc453460986)

[2.1.7 开源库beautyeye\_lnf.jar 7](#_Toc453460987)

[2.2 线程 7](#_Toc453460988)

[2.2.1 气泡线程 7](#_Toc453460989)

[2.2.2 FPS计算线程 8](#_Toc453460990)

[2.2.3 添加小球线程 8](#_Toc453460991)

[2.3 功能简介 8](#_Toc453460992)

[2.3.1 多线程编程 8](#_Toc453460993)

[2.3.2 小球自行绘制 8](#_Toc453460994)

[2.3.3内置四种小球样式 9](#_Toc453460995)

[2.3.4 屏幕截图 10](#_Toc453460996)

[2.3.5 自定义背景 10](#_Toc453460997)

[2.3.6 自动扫描文件夹内背景图片 10](#_Toc453460998)

[2.3.7 碰撞真实 10](#_Toc453460999)

[2.3.8 模拟重力 10](#_Toc453461000)

[2.3.9 碰撞变色 10](#_Toc453461001)

[2.3.10 持续变色 11](#_Toc453461002)

[2.3.11 FPS计算 11](#_Toc453461003)

[第三章 核心算法 12](#_Toc453461004)

[3.1 碰撞后速度算法 12](#_Toc453461005)

[3.1.1 坐标系定义 12](#_Toc453461006)

[3.1.2 推算速度 12](#_Toc453461007)

[3.1.3 代码实现 14](#_Toc453461008)

[3.2 颜色生成算法 16](#_Toc453461009)

[3.2.1随机生成小球颜色 16](#_Toc453461010)

[3.2.2小球颜色渐变动画 16](#_Toc453461011)

[3.3.3小球碰撞颜色改变 17](#_Toc453461012)

[3.3 碰撞检测算法 17](#_Toc453461013)

[3.3.1获取小球下一帧的位置 17](#_Toc453461014)

[3.3.2检测小球是否碰撞 17](#_Toc453461015)

[第四章 程序代码 19](#_Toc453461016)

[第五章 心得体会 20](#_Toc453461017)

[参考文献 20](#_Toc453461018)

# 第一章 作业要求

屏保（气泡），要求：气泡自行绘制，多线程编程。

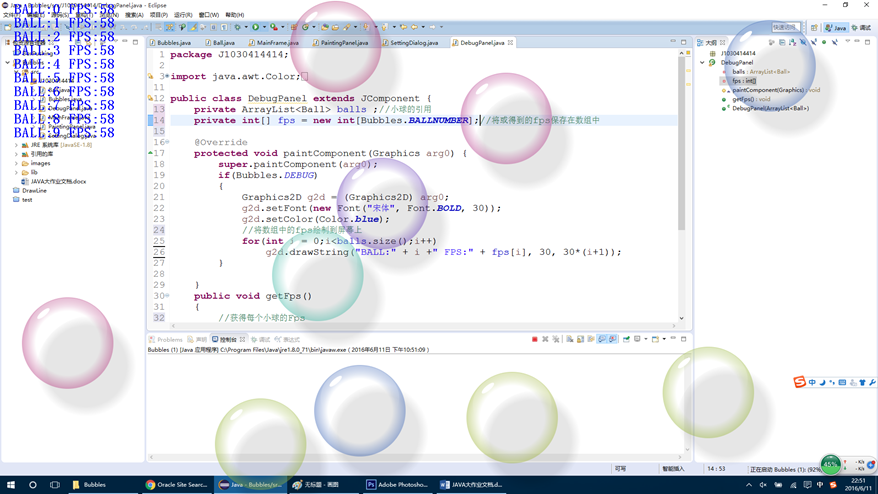
# 第二章 程序简介

本作业使用JAVA编写，基于JDK1.8。默认全屏显示，按esc或者回车退出。双击打开设置。

## 2.1主要文件

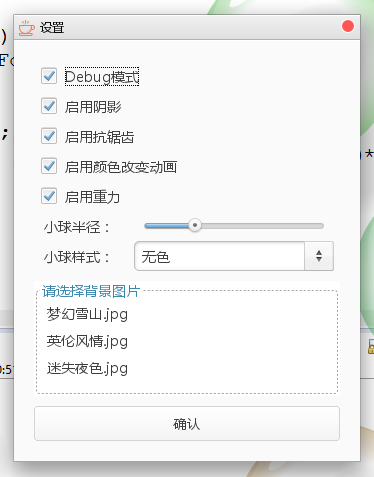
### 2.1.1 MainFrame.java

显示屏保的主窗体，默认全屏显示，按esc或者回车退出。



### 2.1.2 SettingDialog.java

设置对话框，双击主界面打开。提供多种设置。

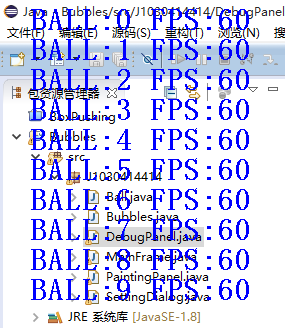


### 2.1.3 PaintingPanel.java

包含于MainFrame中，是程序的绘图面板。

### 2.1.4 DebugPanel.java

继承自JComponent。这个组件会显示所有小球的Fps值。



### 2.1.5 Ball.java

小球类，每个小球独占一个线程，每个小球包含各自的颜色，位置，速度等信息。

### 2.1.6 Bubbles.java

程序的入口，并且保存了一些静态常量。

### 2.1.7 开源库beautyeye\_lnf.jar

[beautyeye](https://github.com/JackJiang2011/beautyeye)是一个界面库。

## 2.2 线程

### 2.2.1 气泡线程

代码位于**Ball.java**第70行

每个气泡独占一个线程，有N个气泡则有N个线程。

每个线程负责对应气泡与其他气泡的碰撞检测，对应气泡的下一帧位置的计算。

生存周期：直至程序退出

### 2.2.2 FPS计算线程

代码位于**PaintingPanel.java**第55行

若启动Debug模式，则定时计算FPS。

生存周期：直至程序退出

### 2.2.3 添加小球线程

代码位于**PaintingPanel.java**第74行

因为每添加一个球要延时，若在主线程会造成阻塞，所以单独开辟一个线程添加小球。

生存周期：添加所有小球后线程结束。

## 2.3 功能简介

### 2.3.1 多线程编程

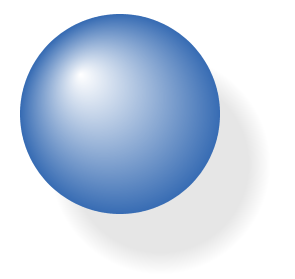
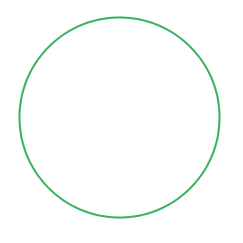
后台持续多线程工作：每个气泡独占一个线程，Fps计算占一个线程。

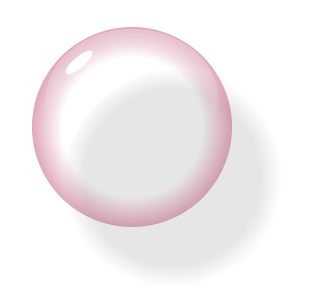
### 2.3.2 小球自行绘制

每个小球采用了中心渐变绘图，制造出高光效果和阴影效果。

### 2.3.3内置四种小球样式







### 2.3.4 屏幕截图

小球的背景默认采用当前截图

### 2.3.5 自定义背景

除了默认的截图作为背景，屏保的背景可以设置为自定义的背景图片。



### 2.3.6 自动扫描文件夹内背景图片

只要将背景图片放在.\images目录下，程序会自动扫描出，用户可以设置。



### 2.3.7 碰撞真实

小球碰撞遵守动量守恒和能量守恒定律。

### 2.3.8 模拟重力

小球会受重力的影响

### 2.3.9 碰撞变色

小球每次碰撞都会稍微变色。

### 2.3.10 持续变色

小球在任何时刻都在缓慢的改变颜色。

### 2.3.11 FPS计算

当打开调试模式后，屏幕左上角会显示当前的帧率，以便调整延时，平衡动画的流畅度和CPU负载。

# 第三章 核心算法

## 3.1 碰撞后速度算法

### 3.1.1 坐标系定义

小球的坐标系采用java的系统坐标系，记做S1，如图所示。

x

y

为了计算小球碰撞的速度，以小球连心线方向为x轴，垂直方向为y轴，建立参考坐标系。记做S2，如图所示。

x

y

### 3.1.2 推算速度

设小球1,2的质量均为，小球1以速度碰撞小球2，小球2的初速度为，碰撞后小球1的速度为，小球2的速度为,。

由于小球碰撞过程满足动量守恒和能量守恒，可以得出：

为了方便运算，将小球的速度在S2系中正交分解。

x

y

Vx

Vy

V

记做，，，，，，，。

所以我们目前的目的就是求出，，，。

由于碰撞时，受力方向仅在小球连心线方向。所以，，均不变。

即：

将③，④代入①，②，可以解出：

即，在x方向上，V1的速度和V2的速度发生了交换。

### 3.1.3 代码实现

在代码中，小球的位置是在S1坐标系中保存的，保存了x，y坐标。this为小球1，ball为小球2。首先求出小球1与小球2的距离差。

double dx = ball.x - x;

double dy = ball.y - y;

利用距离差求出夹角，便于进行坐标变换。

double angle = Math.atan(dy / dx);

坐标变换时分为两种情况：

this

dy

dx

ball

this

dy

dx

ball

在第一种情况中：dx>0,dy>0,angle>0。在第二种情况中：dx>0,dy<0;angle<0

令坐标系S1中的速度表示为vx，vy；坐标系S2中的速度表示为s2\_vx,s2\_vy。

vx

s2\_vy

s2\_vx

vy

在进行坐标变换时，两种情况投影公式存在符号上的差异。

第一种情况：

|  |  |  |
| --- | --- | --- |
| S1 | S2 | 符号 |
| vx | s2\_vx | 同号 |
| vx | s2\_vy | 异号 |
| vy | s2\_vx | 同号 |
| vy | s2\_vy | 同号 |

第二种情况：

|  |  |  |
| --- | --- | --- |
| S1 | S2 | 符号 |
| vx | s2\_vx | 同号 |
| vx | s2\_vy | 同号 |
| vy | s2\_vx | 异号 |
| vy | s2\_vy | 同号 |

无论是S1投影至S2，还是S2投影至S1，符号变化的规律均为上表中所示。

使用如下公式即可同时满足上述两种情况：

将S1中的量投影至S2：

|  |  |  |
| --- | --- | --- |
| S1 | S2 | 公式 |
| vx | s2\_vx | s2\_vx = vx \* Math.cos(angle) |
| vx | s2\_vy | s2\_vy = -vx \* Math.sin(angle) |
| vy | s2\_vx | s2\_vx = vy \* Math.sin(angle) |
| vy | s2\_vy | s2\_vy = vy \* Math.cos(angle) |

将S2中的量投影至S1：

|  |  |  |
| --- | --- | --- |
| S2 | S1 | 公式 |
| s2\_vx | vx | vx = s2\_vx \* Math.cos(angle) |
| s2\_vx | vy | vy = s2\_vx \* Math.sin(angle) |
| s2\_vy | vx | vx = -s2\_vy \* Math.sin(angle) |
| s2\_vy | vy | vy = s2\_vy \* Math.cos(angle); |

根据上表的公式进行坐标变换。将速度投影后，即可按照3.1.2中的推论求出碰撞后的速度。

碰撞后，ball在S2系中x轴速度记做s2\_vx2，y轴速度记做s2\_vy2，this在S2系中x轴速度记做s2\_vx1，y轴速度记做s2\_vy1。

ball的x轴速度即为this的x轴速度。将this两个分量vx，vy投影到S2系中即可。

ball的y轴速度不变，将ball两个分量vx，vy投影至S2系中即可。

This的速度和ball的速度公式类似，交换变量即可。

double s2\_vx2 = vx \* Math.cos(angle) + vy \* Math.sin(angle);

double s2\_vy2 = -ball.vx \* Math.sin(angle) + ball.vy \* Math.cos(angle);

double s2\_vx1 = ball.vx \* Math.cos(angle) + ball.vy \* Math.sin(angle);

double s2\_vy1 = -vx \* Math.sin(angle) + vy \* Math.cos(angle);

最后将S2系中的变量投影至S1。

ball.setVx(s2\_vx2 \* Math.cos(angle) - s2\_vy2 \* Math.sin(angle));

ball.setVy(s2\_vx2 \* Math.sin(angle) + s2\_vy2 \* Math.cos(angle));

vx = s2\_vx1 \* Math.cos(angle) - s2\_vy1 \* Math.sin(angle);

vy = s2\_vx1 \* Math.sin(angle) + s2\_vy1 \* Math.cos(angle);

## 3.2 颜色生成算法

颜色通常使用两种表示方法，RGB和HSB。RGB表示颜色时，需要提供红，绿，蓝三种颜色。这样表示颜色时，两种相近的颜色数值上相差特别大，并且不易于表示。为了满足本程序的需求，本程序使用HSB来表示颜色。

使用HSB表示颜色时需要提供色相，饱和度，亮度。程序中可以这样生成：

new Color(Color.HSBtoRGB(h, s, b));

h的取值范围不限。s，b的取值范围为0~1。

### 3.2.1随机生成小球颜色

将饱和度，亮度固定一个值，只随机产生色相即可生成一种随机颜色。使用这种随机产生颜色的方式可以保持所有小球饱和度，亮度的统一。

hue = 0.01F \* r.nextInt(100);

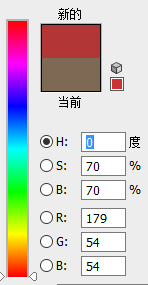
this.color = new Color(Color.HSBtoRGB(hue, 0.7F, 0.7F));

### 3.2.2小球颜色渐变动画

由于小球的色相可以是任何浮点数，并且相近的颜色具有相近的色相。所以将色相微调即可产生一个渐变色。

hue += 0.005;// 色调微调

this.color = new Color(Color.HSBtoRGB(hue, 0.7F, 0.7F));



色相的变化可以参考上图中左测Slider的颜色。

### 3.2.3小球碰撞颜色改变

碰撞时将色相调整一些即可。

this.hue = this.hue + 0.05F;

ball.hue = ball.hue + 0.05F;

this.color = new Color(Color.HSBtoRGB(hue, 0.7F, 0.7F));

ball.color = new Color(Color.HSBtoRGB(ball.hue, 0.7F, 0.7F));

## 3.3 碰撞检测算法

### 3.3.1获取小球下一帧的位置

由于在程序中，小球的位置是一帧一帧变化的。所以碰撞与否的检测，首先要获取小球当前帧和下一帧的位置。

当前帧的位置记做x，y。

根据物理常识可以得知，下一帧的位置可以通过下面的代码求出：

public double getNextX(double t) {

return x + t \* vx;

}

public double getNextY(double t) {

return y + t \* vy;

}

其中t是当前帧和下一帧的间隔时间。

由于下一帧还没发生，所以我们无法知道下一帧的时间，也就无法求出准确的间隔时间。但是我们可以知道上一帧和当前帧的间隔时间，所以用这个时间来估算下一帧的间隔时间。

double t = RATE \* (System.nanoTime() - time) / 10000000;

System.nanoTime()为当前时间，精确至10-9秒，RATE为常数，通过调整RATE的值可以控制下一帧的位置，也就可以控制小球整体的运行速度。

当碰撞检测等一系列工作做完后，当前帧就结束了。在当前帧结束前，更新时间，以便下一帧使用。

time = System.nanoTime();

### 3.3.2检测小球是否碰撞

有了下一帧的位置，就可以提前知道小球是否将要发生碰撞。如果当前帧两球未相交，下一帧两球相交，即为将要发生碰撞。若只判断下一帧，不管当前帧是否相交，就会在某些情况令小球陷入无尽的碰撞中，例如刚开始生成小球时小球是相交的，但是这种情况不是碰撞的相交。还例如有些时候估算会出现一些小误差，导致小球已经相交，这时若判定为碰撞，碰撞后小球还是相交的，这样就会进入无尽的碰撞过程中。

检测小球是否相交，可以通过判断球心距离与小球的半径关系。若下一帧小球间距小于R\*2且当前帧小球距离大于R\*2，则将要发生碰撞

if (nextDis < R \* 2 && thisDis > R \* 2)

# 第四章 程序代码

由于代码过多请

* 1. 查看.\ F:\code\java\Bubbles\src文件夹
  2. 查看<https://github.com/Ethan0w0/Bubbles>

# 第五章 心得体会

* + - 1. 学会利用网络资源，搜索需要的资料。
      2. 画图的时候调参数需要一点一点试。

参考文献

1. RadialGradientPaint (Java Platform SE 8 )

<https://docs.oracle.com/javase/8/docs/api/java/awt/RadialGradientPaint.html>

1. System (Java Platform SE 8 )

<https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#currentTimeMillis-->

1. Color (Java Platform SE 8 )

<https://docs.oracle.com/javase/8/docs/api/java/awt/Color.html>