TTS 7.0 COOKBOOK

（JAVA SHOOT UNIT 01）

版本编号 7.0

2013-12

达内IT培训集团

JAVA SHOOT Unit01

1. Shoot项目

* 问题

Shoot游戏是一款十分有趣的射击类小游戏，流畅的画面，高难度的挑战。游戏中，玩家驾驶英雄机，在空中进行战斗。点击并移动自己的英雄机，发射炮弹，打掉敌飞机以及蜜蜂，来获得分数和奖励，打掉一架敌飞机赢得5分，打掉一只蜜蜂赢得1条命或是获得20次双倍火力，如果撞上敌飞机或小蜜蜂，将减少命、双倍火力清零。每撞到一次蜜蜂或是敌飞机命减1，当命数为0时，则游戏结束。初始界面如图-1所示。

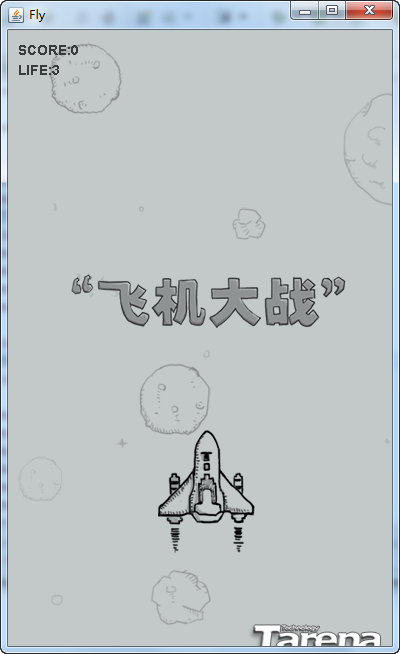


图 – 1

从图-1可以看出，默认分数为0，默认3条命，请看如图-2所示具体介绍。



图 – 2

玩家在如图-1所示的界面的任意位置，按下鼠标左键，开始游戏。界面效果如图-3所示。



图 – 3

开始游戏后，天空中不断有敌飞机和蜜蜂出现，英雄机发射子弹打掉敌飞机和蜜蜂以获取分数、增命或是双倍火力。如果英雄机与飞机或蜜蜂发生碰撞则减少命并且双倍火力清零，直至寿命为0则游戏结束。界面效果如图-4所示。



图 – 4

此时点击鼠标左键，可以重新进入开始状态。

另外，在游戏进行过程中，鼠标离开游戏界面，游戏将进入暂停状态，界面效果如图-5所示。



图 – 5

当鼠标再次移入界面时，游戏将继续进行。

* 方案

软件的开发过程如下：

1. 需求(软件功能的文字描述)

2. 业务需求分析: 找对象，以及对象之间的关系。本项目中对象如下所示：

ShootGame

|-- 英雄机 Hero

|-- 敌飞机 Airplane

|-- 蜜蜂Bee

|-- 子弹Bullet

3. 软件概要设计

数据建模：使用一个数据模型，描述对象的关系。使用绘图坐标系作为参考模型，英雄机、敌飞机、蜜蜂、子弹都是矩形区域，如图-6所示。

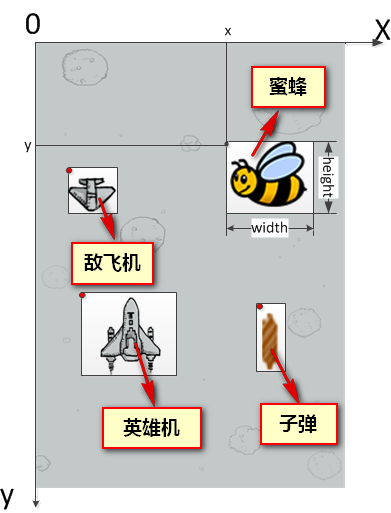


图 – 6

图-6中以蜜蜂为例，标识出了数据x、y、width以及height所表示的位置。英雄机、敌飞机、子弹与蜜蜂的这四个属性类似的。

4. 类的设计

本案例中的类、及类之间的关系如图-7所示。

图 – 7

* 步骤

实现此案例需要按照如下步骤进行。

步骤一：新建工程和包

首先，新建名为Shoot的Java工程； 然后，在工程下的src目录下新建包com.tarena.shoot； 最后，将该工程所需的图片拷贝到该包下，工程结构如图-8所示：

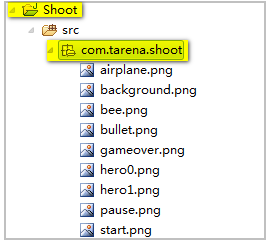


图- 8

在项目中，包的名字一般为公司域名倒过来，再加上项目名称，即为包名。如图-8中的包名为com.tarena.shoot，其中，com.tarena是达内公司的域名倒过来，shoot为本项目的名称。

步骤二：创建抽象父类FlyingObject

由图-6可以分析出英雄机、敌飞机、子弹以及蜜蜂都有x、y、width以及height属性，因此，将这些属性抽象到父类FlyingObject中。另外，它们在界面上都以图片的形式显示，因此在父类FlyingObject中，添加image属性，表示它们的贴图，并提供上述5个属性的getter和setter方法，FlyingObject类的代码如下所示：

package com.tarena.shoot;

import java.awt.image.BufferedImage;

public abstract class FlyingObject {

protected int x; //x坐标

protected int y; //y坐标

protected int width; //宽

protected int height; //高

protected BufferedImage image; //图片

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public int getWidth() {

return width;

}

public void setWidth(int width) {

this.width = width;

}

public int getHeight() {

return height;

}

public void setHeight(int height) {

this.height = height;

}

public BufferedImage getImage() {

return image;

}

public void setImage(BufferedImage image) {

this.image = image;

}

}

步骤三：创建接口Enemy，实现该接口的类为敌人

创建接口Enemy，表示敌人。如果子弹击中敌飞机，英雄机可以获取分数，因此，在Enemy接口中提供获取分数的方法，代码如下所示：

/\*\*

\* 敌人，可以有分数

\*/

public interface Enemy {

/\*\* 敌人的分数 \*/

int getScore();

}

步骤四：创建接口 Award，实现该接口的类表示奖励

创建接口Award，表示奖励。如果子弹击中了蜜蜂，英雄机可以获取奖励。奖励有两种形式，分别是双倍火力或增命，因此，提供获取的奖励类型的方法，代码如下所示：

package com.tarena.shoot;

/\*\*

\* 奖励

\*/

public interface Award {

int DOUBLE\_FIRE = 0; //双倍火力

int LIFE = 1; //1条命

/\*\* 获得奖励类型(上面的0或1) \*/

int getType();

}

上述代码中，如果奖励类型为0，则表示奖励双倍火力；如果奖励类型为1，则表示奖励1条命。

步骤五：新建类Airplane,表示敌飞机

新建类Airplane，表示敌飞机。敌飞机属于飞行物，因此，继承自FlyingObject类；敌飞机也属于敌人，因此，需要实现Enemy接口。敌飞机可以向下移动，因此有移动的速度作为属性，代码如下所示：

package com.tarena.shoot;

import java.util.Random;

import com.tarena.shoot.ShootGame;

/\*\*

\* 敌飞机: 是飞行物，也是敌人

\*/

public class Airplane extends FlyingObject implements Enemy {

private int speed = 2;

public int getScore() {

return 0;

}

}

步骤六：新建类Bee,表示蜜蜂

新建类Bee，表示蜜蜂。蜜蜂属于飞行物，因此，继承自FlyingObject类；击中蜜蜂可以获得奖励，因此，需要实现Award接口，并且有奖励类型作为属性。蜜蜂可以左右移动、向下移动，因此有移动的速度作为属性，代码如下所示：

package com.tarena.shoot;

/\*\* 蜜蜂 \*/

public class Bee extends FlyingObject implements Award{

private int xSpeed = 1; //x坐标移动速度

private int ySpeed = 2; //y坐标移动速度

private int awardType; //奖励类型

public int getType() {

return 0;

}

}

步骤七：新建类Bullet,表示子弹

新建类Bullet，表示子弹。子弹属于飞行物，因此，继承自FlyingObject类；子弹可以向上移动，因此有移动的速度作为属性，代码如下所示：

package com.tarena.shoot;

/\*\*

\* 子弹类:是飞行物

\*/

public class Bullet extends FlyingObject {

private int speed = 3; //移动的速度

}

步骤八：新建类Hero,表示英雄机

新建类Hero，表示英雄机。英雄机属于飞行物，因此，继承自FlyingObject类；英雄机发出子弹，击中蜜蜂可以获取双倍火力或增命，因此，将双倍火力的子弹数量和命的数量作为该类的属性，代码如下所示：

package com.tarena.shoot;

import java.awt.image.BufferedImage;

/\*\*

\* 英雄机:是飞行物

\*/

public class Hero extends FlyingObject{

protected BufferedImage[] images = {};

protected int index = 0;

private int doubleFire;

private int life;

}

上述代码中，还有images属性和index属性，其中images属性表示Hero的贴图，Hero的贴图由两张图片组成，因此使用数组类型；index属性是使两张图片进行交替显示的计数。

步骤九：新建类ShootGame，加载图片

新建类ShootGame，该类继承自JPanel，在该类中，使用静态常量定义面板的宽和高，并使用ImageIO的read方法加载图片，代码如下所示：

package com.tarena.shoot;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JPanel;

public class ShootGame extends JPanel {

public static final int WIDTH = 400; // 面板宽

public static final int HEIGHT = 654; // 面板高

public static BufferedImage background;

public static BufferedImage start;

public static BufferedImage airplane;

public static BufferedImage bee;

public static BufferedImage bullet;

public static BufferedImage hero0;

public static BufferedImage hero1;

public static BufferedImage pause;

public static BufferedImage gameover;

static {// 静态代码块

try {

background = ImageIO.read(ShootGame.class

.getResource("background.png"));

airplane = ImageIO

.read(ShootGame.class.getResource("airplane.png"));

bee = ImageIO.read(ShootGame.class.getResource("bee.png"));

bullet = ImageIO.read(ShootGame.class.getResource("bullet.png"));

hero0 = ImageIO.read(ShootGame.class.getResource("hero0.png"));

hero1 = ImageIO.read(ShootGame.class.getResource("hero1.png"));

pause = ImageIO.read(ShootGame.class.getResource("pause.png"));

gameover = ImageIO

.read(ShootGame.class.getResource("gameover.png"));

start = ImageIO

.read(ShootGame.class.getResource("start.png"));

} catch (Exception e) {

e.printStackTrace();

}

}

}

步骤十：为Bee类添加构造方法，初始化属性

在Bee类中添加构造方法，将属性进行初始化，请看图-9。



图 – 9

从图-9可以看出，image属性初始化为ShootGame类加载的图片；width初始化为图片的宽度、height初始化为图片的高度、x坐标的范围为0到（屏幕的宽度 - 蜜蜂的宽度），因此，x坐标初始化为这个范围的随机数；y坐标初始化为蜜蜂的负高度；奖励的类型为2以内的随机数，即为0或者1，代码如下所示：

public Bee(){

this.image = ShootGame.bee;

width = image.getWidth();

height = image.getHeight();

y = -height;

Random rand = new Random();

x = rand.nextInt(ShootGame.WIDTH - width);

awardType = rand.nextInt(2);

}

步骤十一：为Airplane类添加构造方法，初始化属性

在Airplane类中添加构造方法，将属性进行初始化， Airplane的初始化与Bee类似，代码如下所示：

public Airplane(){

this.image = ShootGame.airplane;

width = image.getWidth();

height = image.getHeight();

y = -height;

x = (int)(Math.random()\*(ShootGame.WIDTH - width));

}

步骤十二：为Bullet类添加构造方法，初始化属性

在Bullet类中添加构造方法，将属性进行初始化，代码如下所示：

package com.tarena.shoot;

/\*\*

\* 子弹类:是飞行物

\*/

public class Bullet extends FlyingObject {

private int speed = 3; //移动的速度

public Bullet(int x,int y){

this.x = x;

this.y = y;

this.image = ShootGame.bullet;

}

}

步骤十三：为Hero类添加构造方法，初始化属性

在Hero类中添加构造方法，将属性进行初始化，英雄机的出场位置如图-10所示。



图 – 10

代码如下所示：

public Hero() {

life = 3;

doubleFire = 0;

this.image = ShootGame.hero0;

images = new BufferedImage[]{ShootGame.hero0, ShootGame.hero1};

width = image.getWidth();

height = image.getHeight();

x = 150;

y = 400;

}

步骤十四：编写main方法

在ShootGame类中添加main方法，在该方法中设置窗口的大小、居中、点击窗口的右上角“X”关闭窗口以及设置窗口可见，代码如下所示：

public static void main(String[] args) {

JFrame frame = new JFrame("Fly");

ShootGame game = new ShootGame(); // 面板对象

frame.add(game); // 将面板添加到JFrame中

frame.setSize(WIDTH, HEIGHT); // 大小

frame.setAlwaysOnTop(true); // 其总在最上

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // 默认关闭操作

frame.setLocationRelativeTo(null); // 设置窗体初始位置

frame.setVisible(true); // 尽快调用paint

}

运行ShootGame类，运行效果如图-11所示。

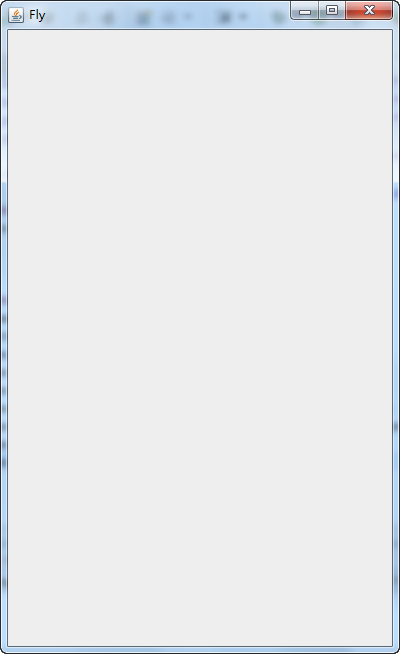


图 – 11

步骤十五：绘制界面

绘制界面的过程如下：

1. 在ShootGame类中，添加FlyingObject[]类型的属性flyings，用于存储射击游戏中的所有敌飞机和蜜蜂；

2. 在ShootGame类中，添加Bullet[]类型的属性bullets，用于存储射击游戏中的所有的子弹；

3. 在ShootGame类中，添加Hero类型的属性hero，表示英雄机；

4. 在ShootGame类中，添加paintHero方法、paintBullets方法、paintFlyingObjects方法，分别用于实现在面板上画英雄机、子弹、敌飞机、蜜蜂；并重写paint方法，在该方法中调用上述三个方法；

5. 在ShootGame类中，添加构造初始化属性flyings、bullets以及hero；

6. 重构Airplane类和Bee类，设置固定的x、y坐标位置，以便显示查看。

ShootGame类中添加的代码如下所示：

package com.tarena.shoot;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class ShootGame extends JPanel {

public static final int WIDTH = 400; // 面板宽

public static final int HEIGHT = 654; // 面板高

public static BufferedImage background;

public static BufferedImage start;

public static BufferedImage airplane;

public static BufferedImage bee;

public static BufferedImage bullet;

public static BufferedImage hero0;

public static BufferedImage hero1;

public static BufferedImage pause;

public static BufferedImage gameover;

private FlyingObject[] flyings = {}; // 敌机数组

private Bullet[] bullets = {}; // 子弹数组

private Hero hero = new Hero(); // 英雄机

public ShootGame(){

//初始化一只蜜蜂一架飞机

flyings=new FlyingObject[2];

flyings[0]=new Airplane();

flyings[1]=new Bee();

//初始化一颗子弹

bullets=new Bullet[1];

bullets[0]=new Bullet(200,350);

}

static {// 静态代码块

... ...

}

@Override

public void paint(Graphics g) {

g.drawImage(background, 0, 0, null); // 画背景图

paintHero(g); // 画英雄机

paintBullets(g); // 画子弹

paintFlyingObjects(g); // 画飞行物

}

/\*\* 画英雄机 \*/

public void paintHero(Graphics g) {

g.drawImage(hero.getImage(), hero.getX(), hero.getY(), null);

}

/\*\* 画子弹 \*/

public void paintBullets(Graphics g) {

for (int i = 0; i < bullets.length; i++) {

Bullet b = bullets[i];

g.drawImage(b.getImage(), b.getX() , b.getY(),null);

}

}

/\*\* 画飞行物 \*/

public void paintFlyingObjects(Graphics g) {

for (int i = 0; i < flyings.length; i++) {

FlyingObject f = flyings[i];

g.drawImage(f.getImage(), f.getX(), f.getY(), null);

}

}

public static void main(String[] args) {

... ...

}

}

Bee类修改的代码如下所示：

package com.tarena.shoot;

import java.util.Random;

/\*\* 蜜蜂 \*/

public class Bee extends FlyingObject implements Award{

private int xSpeed = 1; //x坐标移动速度

private int ySpeed = 2; //y坐标移动速度

private int awardType; //奖励类型

public Bee(){

this.image = ShootGame.bee;

width = image.getWidth();

height = image.getHeight();

//y = -height;

Random rand = new Random();

//x = rand.nextInt(ShootGame.WIDTH - width);

x=100;

y=200;

awardType = rand.nextInt(2);

}

public int getType() {

return 0;

}

}

Airplane修改的代码如下所示：

package com.tarena.shoot;

import java.util.Random;

import com.tarena.shoot.ShootGame;

/\*\*

\* 敌飞机: 是飞行物，也是敌人

\*/

public class Airplane extends FlyingObject implements Enemy {

private int speed = 2;

/\*\* 初始化数据 \*/

public Airplane(){

this.image = ShootGame.airplane;

width = image.getWidth();

height = image.getHeight();

//y = -height;

//x = (int)(Math.random()\*(ShootGame.WIDTH - width));

y=100;

x=100;

}

public int getScore() {

return 0;

}

}

步骤十六：运行

运行ShootGame类，显示的界面效果如图-12所示。

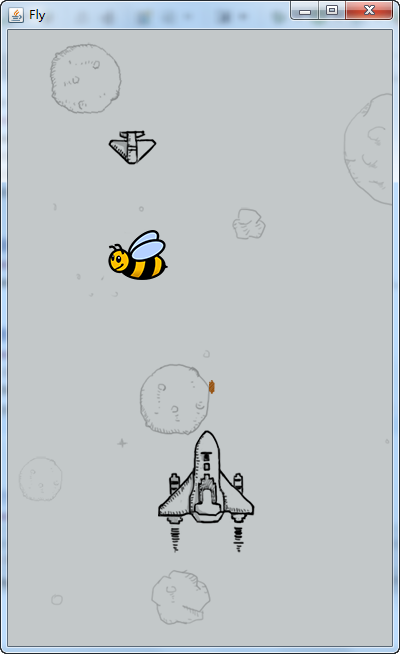


图 – 12

从图-12可以发现，在界面上显示了英雄机、敌飞机、蜜蜂以及子弹。

步骤十七：实现英雄机、敌飞机、蜜蜂、子弹的移动

1. 由于英雄机、敌飞机、蜜蜂以及子弹都是可以移动的，因此在FlyingObject类中添加抽象方法step，声明飞行物移动一步的方法，代码如下所示：

/\*\*

\* 飞行物移动一步

\*/

public abstract void step();

3. 在Airplane类中，实现父类FlyingObject的step方法，实现的代码如下所示：

@Override

public void step() { //移动

y += speed;

}

4. 在Bee类中，实现父类FlyingObject的step方法，实现的代码如下所示：

@Override

public void step() { //可斜飞

x += xSpeed;

y += ySpeed;

if(x > ShootGame.WIDTH-width){

xSpeed = -1;

}

if(x < 0){

xSpeed = 1;

}

}

由于蜜蜂可以左右移动，因此，当移动到屏幕的最右端时，使其向左移动；当移动到屏幕的最左端时，使其向右移动。蜜蜂左右移出屏幕的临界状态如图-13所示。



图 – 13

从图-13可以看出，当蜜蜂的x坐标小于0或大于屏幕的宽度 – 蜜蜂的宽度时，蜜蜂移出游戏界面。

5. 在Bullet类中，实现父类FlyingObject的step方法，实现的代码如下所示：

@Override

public void step(){ //移动方法

y-=speed;

}

6. 在Hero类中，实现父类FlyingObject的step方法，实现的代码如下所示：

@Override

public void step() {

if(images.length>0){

image = images[index++/10%images.length];

}

}

英雄机的step方法，实现了图片的更换，有动画效果。

7. 在ShootGame类中，添加nextOne方法，该方法用于随机产生蜜蜂和敌飞机，代码如下所示：

/\*\*

\* 随机生成飞行物

\*

\* @return 飞行物对象

\*/

public static FlyingObject nextOne() {

Random random = new Random();

int type = random.nextInt(20); // [0,19)

if (type==0) {

return new Bee();

}else{

return new Airplane();

}

}

从代码中可以发现，产生蜜蜂的几率会小一些，只有当产生的随机数为0时，才产生蜜蜂。

8. 在ShootGame类中，添加enterAction方法，该方法用于实现每调用40次该方法，将随机生成的一个蜜蜂或是敌飞机放入flying数组中，代码如下所示：

int flyEnteredIndex = 0; // 飞行物入场计数

/\*\* 飞行物入场 \*/

public void enterAction() {

flyEnteredIndex++;

if (flyEnteredIndex % 40 == 0) { // 400毫秒--10\*40

FlyingObject obj = nextOne(); // 随机生成一个飞行物

flyings = Arrays.copyOf(flyings, flyings.length + 1);扩容

flyings[flyings.length - 1] = obj;//放到最后一位

}

}

9. 在ShootGame类中，添加stepAction方法，该方法用于实现所有飞行物的移动，代码如下所示：

public void stepAction() {

/\*\* 飞行物走一步 \*/

for (int i = 0; i < flyings.length; i++) {

FlyingObject f = flyings[i];

f.step();

}

/\*\* 子弹走一步 \*/

for (int i = 0; i < bullets.length; i++) {

Bullet b = bullets[i];

b.step();

}

hero.step();

}

10. 在ShootGame类中，添加如下两个属性：

private Timer timer; // 定时器

private int intervel = 1000/100; // 时间间隔(毫秒)

另外，添加action方法，该方法使用Timer实现每隔10毫秒入场一个飞机或是蜜蜂，并使所有飞行物移动一步，最后重绘页面，代码如下所示：

public void action() { // 启动执行代码

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

enterAction(); // 飞行物入场

stepAction(); // 走一步

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

11.在main方法中调用action方法，代码如下所示：

public static void main(String[] args) {

JFrame frame = new JFrame("Fly");

ShootGame game = new ShootGame(); // 面板对象

frame.add(game); // 将面板添加到JFrame中

frame.setSize(WIDTH, HEIGHT); // 大小

frame.setAlwaysOnTop(true); // 其总在最上

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // 默认关闭操作

frame.setLocationRelativeTo(null); // 设置窗体初始位置

frame.setVisible(true); // 尽快调用paint

game.action(); // 启动执行

}

12.重构Airplane类的构造方法，代码如下所示：

/\*\* 初始化数据 \*/

public Airplane(){

this.image = ShootGame.airplane;

width = image.getWidth();

height = image.getHeight();

y = -height;

x = (int)(Math.random()\*(ShootGame.WIDTH - width));

// y=100;

// x=100;

}

13.重构Bee类的构造方法，代码如下所示：

public Bee(){

this.image = ShootGame.bee;

width = image.getWidth();

height = image.getHeight();

y = -height;

Random rand = new Random();

x = rand.nextInt(ShootGame.WIDTH - width);

// x=100;

// y=200;

awardType = rand.nextInt(2);

}

此时，运行ShootGame类，会发现敌飞机一直向下移动、子弹一直向上移动、蜜蜂是斜着飞的、英雄机的尾翼是有动画效果的。

步骤十八：实现英雄机发射子弹

1.在Hero类中添加shoot方法，实现发射子弹，英雄机发射子弹的位置如图-14所示。



图 – 14

从图-14可以看出，将英雄机的宽度分成了四分，在1/2处发出的子弹是单倍火力的发射点；在1/4和3/4处发出的子弹是双倍火力的发射点，shoot方法的代码如下所示：

public Bullet[] shoot() { // 发射子弹

int xStep = width / 4;

int yStep = 20;

if (doubleFire>0) {

Bullet[] bullets = new Bullet[2];

bullets[0] = new Bullet(x + xStep, y - yStep);

bullets[1] = new Bullet(x + 3 \* xStep, y - yStep);

doubleFire -= 2;

return bullets;

} else { // 单倍

Bullet[] bullets = new Bullet[1];

// y-yStep(子弹到飞机的位置)

bullets[0] = new Bullet(x + 2 \* xStep, y - yStep);

return bullets;

}

}

2.在ShootGame类中添加shootAction方法，实现每调用30次该方法发射一次子弹，并将发射的子弹存储到bullets数组中，shootAction方法的代码如下所示：

int shootIndex = 0; // 射击计数

/\*\* 射击 \*/

public void shootAction() {

shootIndex++;

if (shootIndex % 30 == 0) { // 100毫秒发一颗

Bullet[] bs = hero.shoot(); // 英雄打出子弹

bullets = Arrays.copyOf(bullets, bullets.length + bs.length); // 扩容

System.arraycopy(bs, 0, bullets, bullets.length - bs.length,

bs.length); // 追加数组

}

}

3. 在ShootGame类中的action方法调用shootAction方法，代码如下所示：

public void action() { // 启动执行代码

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

4. 重构ShootGame类的构造方法，将其中的代码注释掉，注释的代码如下所示：

public ShootGame(){

//初始化一只蜜蜂一架飞机

// flyings=new FlyingObject[2];

// flyings[0]=new Airplane();

// flyings[1]=new Bee();

//初始化一颗子弹

// bullets=new Bullet[1];

// bullets[0]=new Bullet(200,350);

}

此时，运行ShootGame类，会发现界面上实现了连续发射子弹。

步骤十九：添加鼠标移动事件处理 ，当鼠标移动时，英雄机跟随着移动

1. 在Hero类中，添加moveTo方法，该方法有两个参数x、y，分别表示鼠标的x坐标位置和y坐标位置，如图-15中的红点位置表示鼠标所在的位置，英雄机的中心点。



图 – 15

从图-15可以看出，要实现英雄机跟随着移动而移动，那么英雄机的坐标算法如下：

hero.x=鼠标的x坐标-width/2；

hero.y=鼠标的y坐标-height/2

moveTo方法的代码如下所示：

/\*\*

\* 当前物体移动了一下，相对距离， x,y鼠标位置

\*/

public void moveTo(int x, int y) {

this.x = x - width / 2;

this.y = y - height / 2;

}

2.在ShootGame类的action方法，添加鼠标的移动事件处理，代码如下所示：

public void action() { // 启动执行代码

// 鼠标监听事件

MouseAdapter l = new MouseAdapter() {

@Override

public void mouseMoved(MouseEvent e) { // 鼠标移动

int x = e.getX();

int y = e.getY();

hero.moveTo(x, y);

}

};

this.addMouseMotionListener(l); // 处理鼠标滑动操作

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

bangAction();

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

步骤二十：实现子弹打敌飞机和蜜蜂

1. 由于蜜蜂和敌飞机都可以被子弹击中，因此在FlyingObject类中添加shootBy方法，该方法的参数为子弹类型。图-16以蜜蜂为例，说明了被子弹击中的算法。



图 – 16

由图-16可以看出，当子弹的x坐标在蜜蜂的x与x+width之间，并且子弹的y坐标在蜜蜂的y与y+height之间时，子弹击中了蜜蜂，即：

bee.x<bullet.x<bee.x+width

&&

bee.y<bullet.y<bee.y+height

在代码中把蜜蜂换成this就可以了，shootBy方法代码如下所示：

/\*\*

\* 检查当前飞行物体是否被子弹(x,y)击(shoot)中，

\* true表示击中，飞行物可以被击中

\* @param Bullet 子弹对象

\* @return true表示被击中了

\*/

public boolean shootBy(Bullet bullet){

int x = bullet.x; //子弹横坐标

int y = bullet.y; //子弹纵坐标

return this.x<x && x<this.x+width && this.y<y && y<this.y+height;

}

2.当英雄机击中蜜蜂时，可以获取奖励，增命或是获得双倍火力，因此在Hero类中添加addDoubleFire实现获取双倍火力；添加addLife方法增命，代码如下所示：

public void addDoubleFire(){

doubleFire += 40;

}

public void addLife() { // 增命

life++;

}

3.在Airplane类中，实现getScore方法，每击中一架敌飞机获得5分，getScore方法的代码如下所示：

public int getScore() {

return 5;

}

4. 在Bee类中，实现getType方法，获取奖励的类型，getType的代码如下所示：

public int getType() {

return awardType;

}

5. 在ShootGame类中添加属性score，用于记录得分，代码如下所示：

private int score = 0; // 得分

6. 在ShootGame类中，添加bangAction方法和bang方法，这两个方法实现了子弹与飞行物（蜜蜂或敌飞机）的碰撞检测，详细过程如下：

1） 循环遍历存储所有的子弹数组bullets；

2） 在上述循环中，再次使用循环，遍历存储所有飞行物（蜜蜂或敌飞机）的数组flyings，在该循环中判断当前子弹是否击中某个飞行物(蜜蜂或敌飞机)，如果击中则退出该循环，记录被击中的飞行物在flyings数组中的索引index；

3）在flyings数据中找到该飞行物，并将其移除;

4）判断该飞行物的类型是Enemy还是Award，如果是Enemy类型，则获取加分；如果是Award类型，则获取奖励；

5）获取奖励的类型，如果奖励的类型为DOUBLE\_FIRE，则获得20次双倍火力；如果奖励的类型为LIFE，则增命，代码如下所示：

/\*\* 子弹与飞行物碰撞检测 \*/

public void bangAction() {

for (int i = 0; i < bullets.length; i++) { // 遍历所有子弹

Bullet b = bullets[i];

bang(b);

}

}

/\*\* 子弹和飞行物之间的碰撞检查 \*/

public void bang(Bullet bullet) {

int index = -1; // 击中的飞行物索引

for (int i = 0; i < flyings.length; i++) {

FlyingObject obj = flyings[i];

if (obj.shootBy(bullet)) { // 判断是否击中

index = i; // 记录被击中的飞行物的索引

break;

}

}

if (index != -1) { // 有击中的飞行物

FlyingObject one = flyings[index]; // 记录被击中的飞行物

FlyingObject temp = flyings[index]; // 被击中的飞行物与最后一个飞行物交换

flyings[index] = flyings[flyings.length - 1];

flyings[flyings.length - 1] = temp;

// 删除最后一个飞行物(即被击中的)

flyings = Arrays.copyOf(flyings, flyings.length - 1);

// 检查one的类型 如果是敌人, 就算分

if (one instanceof Enemy) { // 检查类型，是敌人，则加分

Enemy e = (Enemy) one; // 强制类型转换

score += e.getScore(); // 加分

}

if (one instanceof Award) { // 若为奖励，设置奖励

Award a = (Award) one;

int type = a.getType(); // 获取奖励类型

switch (type) {

case Award.DOUBLE\_FIRE:

hero.addDoubleFire(); // 设置双倍火力

break;

case Award.LIFE:

hero.addLife(); // 设置加命

break;

}

}

}

}

7. 在Action方法中调用bangAction方法，代码如下所示：

public void action() { // 启动执行代码

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

bangAction();

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

步骤二十一：实现画分数和命数

1.在Hero类中，添加getLife方法，该方法用于获取英雄机的命数，代码如下所示：

public int getLife() {

return life;

}

2.在ShootGame类中，添加paintScore方法，该方法用于画分数和命数，代码如下所示：

/\*\* 画分数 \*/

public void paintScore(Graphics g) {

int x = 10;

int y = 25;

Font font = new Font(Font.SANS\_SERIF,Font.BOLD, 14);

g.setColor(new Color(0x3A3B3B));

g.setFont(font); // 设置字体

g.drawString("SCORE:" + score, x, y); // 画分数

y+=20;

g.drawString("LIFE:" + hero.getLife(), x, y);

}

3.在ShootGame类的paint方法中，调用paintScore方法，代码如下所示：

@Override

public void paint(Graphics g) {

g.drawImage(background, 0, 0, null); // 画背景图

paintHero(g); // 画英雄机

paintBullets(g); // 画子弹

paintFlyingObjects(g); // 画飞行物

paintScore(g);//画分数

}

步骤二十二：删除越界飞行物（蜜蜂和敌飞机）和子弹

1.由于蜜蜂、敌飞机、子弹都可能出现越界现象，因此，在FlyingObject类中添加抽象方法outOfBounds，根据子类不同实现具体的越界算法，代码如下所示：

/\*\*

\* 检查是否出界

\* @param width 边界宽

\* @param height 边界高

\* @return true 出界与否

\*/

public abstract boolean outOfBounds();

2.在Bee类中，实现父类FlyingObject的越界判断方法，蜜蜂是向下飞行的，下越界的临界状态如图-17所示：



图 – 17

由图-17可以看出，当蜜蜂的y坐标大于屏幕的高度时，蜜蜂超出了边界 ，代码如下所示：

@Override

public boolean outOfBounds() {

return y>ShootGame.HEIGHT;

}

3. 在Airplane类中，实现父类FlyingObject的越界判断方法，敌飞机上下越界的临界状态与蜜蜂相同，代码如下所示：

@Override

public boolean outOfBounds() { //越界处理

return y>ShootGame.HEIGHT;

}

4. 在Bullet类中，实现父类FlyingObject的越界判断方法，子弹是向上运动的，子弹上越界的临界状态如图-18所示。



图 – 18

由图-18可以看出，当子弹的y坐标小于子弹的负高度时，子弹超出了边界 ，代码如下所示：

@Override

public boolean outOfBounds() {

return y<-height;

}

5.在ShootGame类中添加outOfBoundsAction方法，该方法用于删除越界飞行物（蜜蜂和敌飞机）及子弹，详细实现过程如下：

1）创建FlyingObject类型的数组flyingLives，用于存储所有活着的飞行物（蜜蜂和敌飞机），即没有越界的飞行物（蜜蜂和敌飞机）；

2）循环遍历存储所有飞行物（蜜蜂和敌飞机）的数组flyings，并判断每一个飞行物（蜜蜂或敌飞机）是否越界，将没有越界的飞行物（蜜蜂或敌飞机）放入flyingLives数组中存储，并记录不越界飞行物的个数index；

3）将flyingLives数组中的元素，复制到flyings数组中，并重新指定flying数组的长度为index；

4）删除子弹与删除飞行物（蜜蜂和敌飞机）的过程类似。

outOfBoundsAction方法的代码如下所示：

/\*\* 删除越界飞行物及子弹 \*/

public void outOfBoundsAction() {

int index = 0;

// 存储活着的飞行物

FlyingObject[] flyingLives = new FlyingObject[flyings.length];

for (int i = 0; i < flyings.length; i++) {

FlyingObject f = flyings[i];

if (!f.outOfBounds()) {

flyingLives[index++] = f; // 不越界的留着

}

}

flyings = Arrays.copyOf(flyingLives, index); // 将不越界的飞行物都留着

index = 0; // 重置为0

Bullet[] bulletLives = new Bullet[bullets.length];

for (int i = 0; i < bullets.length; i++) {

Bullet b = bullets[i];

if (!b.outOfBounds()) {

bulletLives[index++] = b;

}

}

bullets = Arrays.copyOf(bulletLives, index); // 将不越界的子弹留着

}

6.在ShootAction类的action方法中调用outOfBoundsAction，代码如下所示：

public void action() { // 启动执行代码

// 鼠标监听事件

MouseAdapter l = new MouseAdapter() {

@Override

public void mouseMoved(MouseEvent e) { // 鼠标移动

int x = e.getX();

int y = e.getY();

hero.moveTo(x, y);

}

};

this.addMouseMotionListener(l); // 处理鼠标滑动操作

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

bangAction();

outOfBoundsAction(); // 删除越界飞行物及子弹

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

步骤二十三：判断英雄机是否与飞行物（蜜蜂和敌飞机）碰撞

1.当英雄机与飞行物（蜜蜂和敌飞机）发生碰撞时，需要减少命的数量以及将双倍火力清零，因此，在Hero类中添加subtractLife方法，用于实现减命；添加setDoubleFire用于重新设置双倍火力的值，代码如下所示：

public void subtractLife() { // 减命

life--;

}

public void setDoubleFire(int doubleFire) {

this.doubleFire = doubleFire;

}

2. 在Hero类中添加hit方法用于英雄机与飞行物（蜜蜂和敌飞机）的碰撞检测，图-19以蜜蜂为例说明了碰撞算法。



图 – 19

由图-19可以看出，x1、x2以及y1、y2的坐标算法如下：

x1=bee.x-1/2hero.width

x2=bee.x+1/2hero.width+bee.width

y1=bee.y-1/2hero.height

y2=bee.y+1/2hero.height+bee.height

英雄机中心点的x、y坐标算法如下：

hero.中心点x=hero.x+1/2hero.width

hero.中心点y=hero.y+1/2hero.height

当满足如下条件时，英雄机与蜜蜂发生碰撞：

x1<Hero.中心点x<x2

y1<Hero.中心点y<y2

hit方法的实现代码如下所示：

public boolean hit(FlyingObject other) { // 碰撞算法

int x1 = other.x - this.width / 2;

int x2 = other.x + other.width + this.width / 2;

int y1 = other.y - this.height / 2;

int y2 = other.y + other.height + this.height / 2;

return this.x + this.width / 2 > x1 && this.x + this.width / 2 < x2

&& this.y + this.height / 2 > y1

&& this.y + this.width / 2 < y2;

}

3. 在ShootGame类中，添加isGameOver方法，该方法用于判断游戏是否结束，方法实现的详细过程如下：

1） 循环遍历存储所有飞行物（蜜蜂和敌飞机）的数组flyings；

2） 在循环中，判断英雄机是否与某个飞行物（蜜蜂和敌飞机）发生碰撞，如果发生碰撞，则减命、双倍火力清零，并记录被撞飞行物在flyings数组中的索引index，该索引默认为-1，即没有发生碰撞；

3） 判断index是否为-1，如果不为-1，将该索引位置的元素从flyings数组中清除；

4）判断命数是否小于等于0，并返回比较结果。

/\*\* 检查游戏是否结束 \*/

public boolean isGameOver() {

for (int i = 0; i < flyings.length; i++) {

int index = -1;

FlyingObject obj = flyings[i];

if (hero.hit(obj)) { // 检查英雄机与飞行物是否碰撞

hero.subtractLife();

hero.setDoubleFire(0);

index = i;

}

if(index!=-1){

FlyingObject t = flyings[index];

flyings[index] = flyings[flyings.length-1];

flyings[flyings.length-1] = t;

flyings = Arrays.copyOf(flyings, flyings.length-1);

}

}

return hero.getLife() <= 0;

}

步骤二十四：实现游戏的开始、运行、暂停以及结束

游戏分为四种状态，分别为START、RUNNING、PAUSE、GAME\_OVER，表示游戏开始状态、运行状态、暂停状态以及游戏结束状态。

首先介绍一下鼠标事件对状态的影响，当执行鼠标点击事件时，会对游戏中的START状态、GAME\_OVER状态产生影响。如果点击鼠标时为START状态，则将游戏的状态设置为RUNNING，即点击鼠标游戏进入运行状态；如果点击鼠标时为GAME\_OVER状态，则将flyings数组、bullets数组、hero对象、score变量设置为初始状态，并将状态设置为START状态。代码如下：

flyings = new FlyingObject[0];

bullets = new Bullet[0];

hero = new Hero();

score = 0;

state = START;

当鼠标执行移动事件时，判断状态是否RUNNING状态，如果为RUNNING，则执行英雄机跟随鼠标移动的方法。

当鼠标执行进入事件时，判断状态是否PAUSE，如果为PAUSE状态，则状态更改为RUNNING。

当鼠标执行退出事件时，判断状态是否GAME\_OVER状态，如果不为GAME\_OVER状态，则状态更改为PAUSE。

然后，当游戏状态为RUNNING状态时，执行飞行物入场、所有飞行物走一步、射击、子弹打飞行物、删除越界飞行物及子弹、检查游戏结束这一系列动作，代码如下：

if (state == RUNNING) {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

bangAction(); // 子弹打飞行物

outOfBoundsAction(); // 删除越界飞行物及子弹

checkGameOverAction(); // 检查游戏结束

}

最后，如果判断游戏已经结束，那么将游戏状态设置为GAME\_OVER，代码如下：

/\*\* 检查游戏结束 \*/

public void checkGameOverAction() {

if (isGameOver()) {

state = GAME\_OVER; // 改变状态

}

}

具体实现步骤如下：

1.在ShootGame类中添加以下属性和常量，代码如下所示：

private int state;

public static final int START = 0;

public static final int RUNNING = 1;

public static final int PAUSE = 2;

public static final int GAME\_OVER = 3;

2. 在ShootGame类中添加checkGameOverAction方法，该方法用于判断游戏是否已经结束，如果已经结束，则将游戏状态设置为GAME\_OVER，代码如下所示：

/\*\* 检查游戏结束 \*/

public void checkGameOverAction() {

if (isGameOver()) {

state = GAME\_OVER; // 改变状态

}

}

3.修改ShootGame类的action方法，添加鼠标点击、移入、退出等操作的状态处理，代码如下所示：

public void action() { // 启动执行代码

// 鼠标监听事件

MouseAdapter l = new MouseAdapter() {

@Override

public void mouseMoved(MouseEvent e) { // 鼠标移动

if (state == *RUNNING*) { // 运行时移动英雄机

int x = e.getX();

int y = e.getY();

hero.moveTo(x, y);

}

}

@Override

public void mouseEntered(MouseEvent e) { // 鼠标进入

if (state == *PAUSE*) { // 暂停时运行

state = *RUNNING*;

}

}

@Override

public void mouseExited(MouseEvent e) { // 鼠标退出

if (state != *GAME\_OVER*) {

state = *PAUSE*; // 游戏未结束，则设置其为暂停

}

}

@Override

public void mouseClicked(MouseEvent e) { // 鼠标点击

switch (state) {

case *START*:

state = *RUNNING*;

break;

case *GAME\_OVER*: // 游戏结束，清理现场

flyings = new FlyingObject[0];

bullets = new Bullet[0];

hero = new Hero();

score = 0;

state = *START*;

break;

}

}

};

this.addMouseListener(l); // 处理鼠标点击操作

this.addMouseMotionListener(l); // 处理鼠标滑动操作

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

if (state == *RUNNING*) {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

bangAction(); // 子弹打飞行物

outOfBoundsAction(); // 删除越界飞行物及子弹

checkGameOverAction(); // 检查游戏结束

}

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

4. 在ShootGame类中添加paintState方法，画出START、PAUSE以及GAME\_OVER状态显示的图片，代码如下所示：

/\*\* 画游戏状态 \*/

public void paintState(Graphics g) {

switch (state) {

case START:

g.drawImage(start, 0, 0, null);

break;

case PAUSE:

g.drawImage(pause, 0, 0, null);

break;

case GAME\_OVER:

g.drawImage(gameover, 0, 0, null);

break;

}

}

5. 在ShootGame类中paint方法中，调用paintState方法，代码如下所示：

@Override

public void paint(Graphics g) {

g.drawImage(background, 0, 0, null); // 画背景图

paintHero(g); // 画英雄机

paintBullets(g); // 画子弹

paintFlyingObjects(g); // 画飞行物

paintScore(g); // 画分数

paintState(g); // 画游戏状态

}

完整代码

Airplane类的完整代码如下所示：

package com.tarena.shoot;

import com.tarena.shoot.ShootGame;

/\*\*

\* 敌飞机: 是飞行物，也是敌人

\*/

public class Airplane extends FlyingObject implements Enemy {

private int speed = 2;

/\*\* 初始化数据 \*/

public Airplane(){

this.image = ShootGame.airplane;

width = image.getWidth();

height = image.getHeight();

y = -height;

x = (int)(Math.random()\*(ShootGame.WIDTH - width));

// y=100;

// x=100;

}

public int getScore() {

return 5;

}

@Override

public void step() { //移动

y += speed;

}

@Override

public boolean outOfBounds() { //越界处理

return y>ShootGame.HEIGHT;

}

}

Award类的完整代码如下所示：

package com.tarena.shoot;

/\*\*

\* 奖励

\*/

public interface Award {

int DOUBLE\_FIRE = 0; //双倍火力

int LIFE = 1; //1条命

/\*\* 获得奖励类型(上面的0或1) \*/

int getType();

}

Bee类的完整代码如下所示：

package com.tarena.shoot;

import java.util.Random;

/\*\* 蜜蜂 \*/

public class Bee extends FlyingObject implements Award{

private int xSpeed = 1; //x坐标移动速度

private int ySpeed = 2; //y坐标移动速度

private int awardType; //奖励类型

public Bee(){

this.image = ShootGame.bee;

width = image.getWidth();

height = image.getHeight();

y = -height;

Random rand = new Random();

x = rand.nextInt(ShootGame.WIDTH - width);

// x=100;

// y=200;

awardType = rand.nextInt(2);

}

public int getType() {

return awardType;

}

@Override

public void step() { //可斜飞

x += xSpeed;

y += ySpeed;

if(x > ShootGame.WIDTH-width){

xSpeed = -1;

}

if(x < 0){

xSpeed = 1;

}

}

@Override

public boolean outOfBounds() {

return y>ShootGame.HEIGHT;

}

}

Bullet类的完整代码如下所示：

package com.tarena.shoot;

/\*\*

\* 子弹类:是飞行物

\*/

public class Bullet extends FlyingObject {

private int speed = 3; //移动的速度

public Bullet(int x,int y){

this.x = x;

this.y = y;

this.image = ShootGame.bullet;

}

@Override

public void step(){ //移动方法

y-=speed;

}

@Override

public boolean outOfBounds() {

return y<-height;

}

}

Enemy类的完整代码如下所示：

package com.tarena.shoot;

/\*\*

\* 敌人，可以有分数

\*/

public interface Enemy {

/\*\* 敌人的分数 \*/

int getScore();

}

FlyingObject类的完整代码如下所示：

package com.tarena.shoot;

import java.awt.image.BufferedImage;

public abstract class FlyingObject {

protected int x; //x坐标

protected int y; //y坐标

protected int width; //宽

protected int height; //高

protected BufferedImage image; //图片

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

public int getWidth() {

return width;

}

public void setWidth(int width) {

this.width = width;

}

public int getHeight() {

return height;

}

public void setHeight(int height) {

this.height = height;

}

public BufferedImage getImage() {

return image;

}

public void setImage(BufferedImage image) {

this.image = image;

}

/\*\*

\* 飞行物移动一步

\*/

public abstract void step();

/\*\*

\* 检查当前飞行物体是否被子弹(x,y)击(shoot)中，

\* true表示击中，飞行物可以被击中

\* @param Bullet 子弹对象

\* @return true表示被击中了

\*/

public boolean shootBy(Bullet bullet){

int x = bullet.x; //子弹横坐标

int y = bullet.y; //子弹纵坐标

return this.x<x && x<this.x+width && this.y<y && y<this.y+height;

}

/\*\*

\* 检查是否出界

\* @param width 边界宽

\* @param height 边界高

\* @return true 出界与否

\*/

public abstract boolean outOfBounds();

}

Hero类的完整代码如下所示：

package com.tarena.shoot;

import java.awt.image.BufferedImage;

/\*\*

\* 英雄机:是飞行物

\*/

public class Hero extends FlyingObject {

protected BufferedImage[] images = {};

protected int index = 0;

private int doubleFire;

private int life;

public Hero() {

life = 3;

doubleFire = 0;

this.image = ShootGame.hero0;

images = new BufferedImage[] { ShootGame.hero0, ShootGame.hero1 };

width = image.getWidth();

height = image.getHeight();

x = 150;

y = 400;

}

@Override

public void step() {

if (images.length > 0) {

image = images[index++ / 10 % images.length];

}

}

public Bullet[] shoot() { // 发射子弹

int xStep = width / 4;

int yStep = 20;

if (doubleFire > 0) {

Bullet[] bullets = new Bullet[2];

bullets[0] = new Bullet(x + xStep, y - yStep);

bullets[1] = new Bullet(x + 3 \* xStep, y - yStep);

doubleFire -= 2;

return bullets;

} else { // 单倍

Bullet[] bullets = new Bullet[1];

// y-yStep(子弹距飞机的位置)

bullets[0] = new Bullet(x + 2 \* xStep, y - yStep);

return bullets;

}

}

public void addDoubleFire() {

doubleFire += 40;

}

public void setDoubleFire(int doubleFire) {

this.doubleFire = doubleFire;

}

public void addLife() { // 增命

life++;

}

public void subtractLife() { // 减命

life--;

}

public int getLife() {

return life;

}

/\*\*

\* 当前物体移动了一下，相对距离， x,y鼠标位置

\*/

public void moveTo(int x, int y) {

this.x = x - width / 2;

this.y = y - height / 2;

}

@Override

public boolean outOfBounds() {

return false;

}

public boolean hit(FlyingObject other) { // 碰撞算法

int x1 = other.x - this.width / 2;

int x2 = other.x + other.width + this.width / 2;

int y1 = other.y - this.height / 2;

int y2 = other.y + other.height + this.height / 2;

return this.x + this.width / 2 > x1 && this.x + this.width / 2 < x2

&& this.y + this.height / 2 > y1

&& this.y + this.width / 2 < y2;

}

}

ShootGame类的完整代码如下所示：

package com.tarena.shoot;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import java.util.Random;

import java.util.Timer;

import java.util.TimerTask;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class ShootGame extends JPanel {

public static final int WIDTH = 400; // 面板宽

public static final int HEIGHT = 654; // 面板高

/\*\* 游戏的当前状态: START RUNNING PAUSE GAME\_OVER \*/

private int state;

public static final int START = 0;

public static final int RUNNING = 1;

public static final int PAUSE = 2;

public static final int GAME\_OVER = 3;

private int score = 0; // 得分

private Timer timer; // 定时器

private int intervel = 1000 / 100; // 时间间隔(毫秒)

public static BufferedImage background;

public static BufferedImage start;

public static BufferedImage airplane;

public static BufferedImage bee;

public static BufferedImage bullet;

public static BufferedImage hero0;

public static BufferedImage hero1;

public static BufferedImage pause;

public static BufferedImage gameover;

private FlyingObject[] flyings = {}; // 敌机数组

private Bullet[] bullets = {}; // 子弹数组

private Hero hero = new Hero(); // 英雄机

public ShootGame() {

// 初始化一只蜜蜂一架飞机

// flyings=new FlyingObject[2];

// flyings[0]=new Airplane();

// flyings[1]=new Bee();

// 初始化一颗子弹

// bullets=new Bullet[1];

// bullets[0]=new Bullet(200,350);

}

static {// 静态代码块

try {

background = ImageIO.read(ShootGame.class

.getResource("background.png"));

airplane = ImageIO

.read(ShootGame.class.getResource("airplane.png"));

bee = ImageIO.read(ShootGame.class.getResource("bee.png"));

bullet = ImageIO.read(ShootGame.class.getResource("bullet.png"));

hero0 = ImageIO.read(ShootGame.class.getResource("hero0.png"));

hero1 = ImageIO.read(ShootGame.class.getResource("hero1.png"));

pause = ImageIO.read(ShootGame.class.getResource("pause.png"));

gameover = ImageIO

.read(ShootGame.class.getResource("gameover.png"));

start = ImageIO.read(ShootGame.class.getResource("start.png"));

} catch (Exception e) {

e.printStackTrace();

}

}

@Override

public void paint(Graphics g) {

g.drawImage(background, 0, 0, null); // 画背景图

paintHero(g); // 画英雄机

paintBullets(g); // 画子弹

paintFlyingObjects(g); // 画飞行物

paintScore(g);

paintState(g); // 画游戏状态

}

/\*\* 画英雄机 \*/

public void paintHero(Graphics g) {

g.drawImage(hero.getImage(), hero.getX(), hero.getY(), null);

}

/\*\* 画子弹 \*/

public void paintBullets(Graphics g) {

for (int i = 0; i < bullets.length; i++) {

Bullet b = bullets[i];

g.drawImage(b.getImage(), b.getX(), b.getY(), null);

}

}

/\*\* 画飞行物 \*/

public void paintFlyingObjects(Graphics g) {

for (int i = 0; i < flyings.length; i++) {

FlyingObject f = flyings[i];

g.drawImage(f.getImage(), f.getX(), f.getY(), null);

}

}

/\*\* 画分数 \*/

public void paintScore(Graphics g) {

int x = 10;

int y = 25;

Font font = new Font(Font.SANS\_SERIF, Font.BOLD, 14);

g.setColor(new Color(0x3A3B3B));

g.setFont(font); // 设置字体

g.drawString("SCORE:" + score, x, y); // 画分数

y += 20;

g.drawString("LIFE:" + hero.getLife(), x, y);

}

/\*\* 画游戏状态 \*/

public void paintState(Graphics g) {

switch (state) {

case START:

g.drawImage(start, 0, 0, null);

break;

case PAUSE:

g.drawImage(pause, 0, 0, null);

break;

case GAME\_OVER:

g.drawImage(gameover, 0, 0, null);

break;

}

}

public static void main(String[] args) {

JFrame frame = new JFrame("Fly");

ShootGame game = new ShootGame(); // 面板对象

frame.add(game); // 将面板添加到JFrame中

frame.setSize(WIDTH, HEIGHT); // 大小

frame.setAlwaysOnTop(true); // 其总在最上

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // 默认关闭操作

frame.setLocationRelativeTo(null); // 设置窗体初始位置

frame.setVisible(true); // 尽快调用paint

game.action(); // 启动执行

}

public void action() { // 启动执行代码

// 鼠标监听事件

MouseAdapter l = new MouseAdapter() {

@Override

public void mouseMoved(MouseEvent e) { // 鼠标移动

if (state == RUNNING) { // 运行时移动英雄机

int x = e.getX();

int y = e.getY();

hero.moveTo(x, y);

}

}

@Override

public void mouseEntered(MouseEvent e) { // 鼠标进入

if (state == PAUSE) { // 暂停时运行

state = RUNNING;

}

}

@Override

public void mouseExited(MouseEvent e) { // 鼠标退出

if (state != GAME\_OVER) {

state = PAUSE; // 游戏未结束，则设置其为暂停

}

}

@Override

public void mouseClicked(MouseEvent e) { // 鼠标点击

switch (state) {

case START:

state = RUNNING;

break;

case GAME\_OVER: // 游戏结束，清理现场

flyings = new FlyingObject[0];

bullets = new Bullet[0];

hero = new Hero();

score = 0;

state = START;

break;

}

}

};

this.addMouseListener(l); // 处理鼠标点击操作

this.addMouseMotionListener(l); // 处理鼠标滑动操作

timer = new Timer(); // 主流程控制

timer.schedule(new TimerTask() {

@Override

public void run() {

if (state == RUNNING) {

enterAction(); // 飞行物入场

stepAction(); // 走一步

shootAction(); // 射击

bangAction(); // 子弹打飞行物

outOfBoundsAction(); // 删除越界飞行物及子弹

checkGameOverAction(); // 检查游戏结束

}

repaint(); // 重绘，调用paint()方法

}

}, intervel, intervel);

}

/\*\*

\* 随机生成飞行物

\*

\* @return 飞行物对象

\*/

public static FlyingObject nextOne() {

Random random = new Random();

int type = random.nextInt(20); // [0,19)

if (type == 0) {

return new Bee();

} else {

return new Airplane();

}

}

int flyEnteredIndex = 0; // 飞行物入场计数

/\*\* 飞行物入场 \*/

public void enterAction() {

flyEnteredIndex++;

if (flyEnteredIndex % 40 == 0) { // 400毫秒--10\*40

FlyingObject obj = nextOne(); // 随机生成一个飞行物

flyings = Arrays.copyOf(flyings, flyings.length + 1);

flyings[flyings.length - 1] = obj;

}

}

public void stepAction() {

/\*\* 飞行物走一步 \*/

for (int i = 0; i < flyings.length; i++) {

FlyingObject f = flyings[i];

f.step();

}

/\*\* 子弹走一步 \*/

for (int i = 0; i < bullets.length; i++) {

Bullet b = bullets[i];

b.step();

}

hero.step();

}

int shootIndex = 0; // 射击计数

/\*\* 射击 \*/

public void shootAction() {

shootIndex++;

if (shootIndex % 30 == 0) { // 100毫秒发一颗

Bullet[] bs = hero.shoot(); // 英雄打出子弹

bullets = Arrays.copyOf(bullets, bullets.length + bs.length); // 扩容

System.arraycopy(bs, 0, bullets, bullets.length - bs.length,

bs.length); // 追加数组

}

}

/\*\* 子弹与飞行物碰撞检测 \*/

public void bangAction() {

for (int i = 0; i < bullets.length; i++) { // 遍历所有子弹

Bullet b = bullets[i];

bang(b);

}

}

/\*\* 子弹和飞行物之间的碰撞检查 \*/

public void bang(Bullet bullet) {

int index = -1; // 击中的飞行物索引

for (int i = 0; i < flyings.length; i++) {

FlyingObject obj = flyings[i];

if (obj.shootBy(bullet)) { // 判断是否击中

index = i; // 记录被击中的飞行物的索引

break;

}

}

if (index != -1) { // 有击中的飞行物

FlyingObject one = flyings[index]; // 记录被击中的飞行物

FlyingObject temp = flyings[index]; // 被击中的飞行物与最后一个飞行物交换

flyings[index] = flyings[flyings.length - 1];

flyings[flyings.length - 1] = temp;

flyings = Arrays.copyOf(flyings, flyings.length - 1); // 删除最后一个飞行物(即被击中的)

// 检查one的类型 如果是敌人, 就算分

if (one instanceof Enemy) { // 检查类型，是敌人，则加分

Enemy e = (Enemy) one; // 强制类型转换

score += e.getScore(); // 加分

}

if (one instanceof Award) { // 若为奖励，设置奖励

Award a = (Award) one;

int type = a.getType(); // 获取奖励类型

switch (type) {

case Award.DOUBLE\_FIRE:

hero.addDoubleFire(); // 设置双倍火力

break;

case Award.LIFE:

hero.addLife(); // 设置加命

break;

}

}

}

}

/\*\* 删除越界飞行物及子弹 \*/

public void outOfBoundsAction() {

int index = 0;

FlyingObject[] flyingLives = new FlyingObject[flyings.length]; // 活着的飞行物

for (int i = 0; i < flyings.length; i++) {

FlyingObject f = flyings[i];

if (!f.outOfBounds()) {

flyingLives[index++] = f; // 不越界的留着

}

}

flyings = Arrays.copyOf(flyingLives, index); // 将不越界的飞行物都留着

index = 0; // 重置为0

Bullet[] bulletLives = new Bullet[bullets.length];

for (int i = 0; i < bullets.length; i++) {

Bullet b = bullets[i];

if (!b.outOfBounds()) {

bulletLives[index++] = b;

}

}

bullets = Arrays.copyOf(bulletLives, index); // 将不越界的子弹留着

}

/\*\* 检查游戏结束 \*/

public void checkGameOverAction() {

if (isGameOver()) {

state = GAME\_OVER; // 改变状态

}

}

/\*\* 检查游戏是否结束 \*/

public boolean isGameOver() {

for (int i = 0; i < flyings.length; i++) {

int index = -1;

FlyingObject obj = flyings[i];

if (hero.hit(obj)) { // 检查英雄机与飞行物是否碰撞

hero.subtractLife();

hero.setDoubleFire(0);

index = i;

}

if (index != -1) {

FlyingObject t = flyings[index];

flyings[index] = flyings[flyings.length - 1];

flyings[flyings.length - 1] = t;

flyings = Arrays.copyOf(flyings, flyings.length - 1);

}

}

return hero.getLife() <= 0;

}

}