|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2012级物联网1班** | | | **姓名** | **陈点** |
| **实验题目** | **飞行射击游戏** | | | | | |
| **实验时间** | **2014/11/09-2014/11/13** | | **实验地点** | **主教0410** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 ■综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  利用Java GUI、Java Graphics和多线程技术，编写实现飞行射击游戏——雷电。 | | | | | | |
| 二、实验项目内容  1、根据项目任务书，形成完整的需求分析。  2、根据需求，完成详细的设计。  3、基于Java编程实现。 | | | | | | |
| 三、实验过程或算法（源程序）  以下将列举关键部分源程序及做出简要解释说明，完整可执行代码随压缩包一同提交，请使用ADT在Android 4.2.2及以上打开。  【GConstant.java】列举工程中使用的部分常量，以注释对其作出说明  **package** cn.cd.constant;  **import** java.util.HashMap;  **import** android.annotation.SuppressLint;  **import** android.content.Context;  **import** android.media.AudioManager;  **import** android.media.SoundPool;  **import** android.os.Handler;  **import** cn.liwang.monster.R;  @SuppressLint("UseSparseArrays")  **public** **class** GConstant {  **public** **static** SoundPool *sp*; //得到一个声音池引用  **public** **static** HashMap<Integer,Integer> *spMap*; //得到一个map的引用  **public** **static** Context *context*;  **public** **static** **final** String *GAME\_DESCRIBE*=  "一天早上，圣枪游侠在轰鸣声中醒来，" +  "出门看，一架陨落的灰机进入了视线。\n" +  "原来是一群飞龙追赶着人类，误入了召唤师峡谷，" +  "他们挥舞着巨大的翅膀，口吐火球，瞬间让野区映成了一片火海。\n" +  "（这不是来送经验的么）卢锡安表示要向这些不速之客讨回公道。\n" +  "注明： 游戏背景图来自mmm\_m，地图滚动方式部分代码摘自CSDN，人物图源来自游戏英雄联盟，交流学习用不作商业用途 By okcd00";  **public** **static** **final** **int** *GAME\_THREAD\_DELAY*=4000; //从游戏启动动画界面到游戏主菜单的延时时间  //这个数组保存怪兽的飞行动画的帧的资源id  **public** **static** **final** **int**[] *monsterAnimation* = **new** **int**[] {R.drawable.*monster\_03*,R.drawable.*monster\_06*};  //怪兽死亡后的动画  **public** **static** **final** **int**[] *monsterDeadID*={R.drawable.*monster\_03*,R.drawable.*monster\_03\_01*,  R.drawable.*monster\_03\_02*,R.drawable.*monster\_03\_03*,R.drawable.*monster\_03\_03*,R.drawable.*monster\_03\_05*};  **public** **static** **final** **int** *MY\_WARPLANE\_ID*=R.drawable.*peal*;  **public** **static** **int** *gameState*=1001; //游戏状态值， 1001表示游戏正在运行  **public** **static** Thread *gameThread*; //游戏进程  **public** **static** **final** **int** *monsterBullet* = R.drawable.*monster\_fire*; //怪兽子弹图片ID  //最大的怪兽数量  **public** **static** **int** *numMonsters* =20;  //怪兽增加的速度，即刷几次屏怪兽出现速度加快  **public** **static** **final** **long** *speedAddMonster* = 1000;  //刷几次屏出生一只怪兽  **public** **static** **int** *drawCount* =120;  //最大子弹数量  **public** **static** **int** *numBullets* = 20;  //刷几次屏怪兽射出一颗子弹  **public** **static** **int** *drawMonsterBullet* =30;  //刷几次屏射出一颗子弹  **public** **static** **int** *bulletCount* = 4;  //子弹飞行速度  **public** **static** **final** **int** *speed\_slow* = 3, *speed\_middle* = 10, *speed\_fast* = 14;  **public** **static** **float** *extraDamage*=0.01f;  //游戏背景资源Id  **public** **static** **final** **int**[] *GAME\_BG\_IMAGE\_ID*={R.drawable.*img\_bg\_level\_1*,R.drawable.*img\_bg\_level\_1*};  //游戏音乐Id  **public** **static** **final** **int** *GAME\_BACKGROUNG\_MUSIC\_ID*=R.raw.*combatribesboss1*;    //游戏音乐初始音量  **public** **static** **final** **int** *R\_VOLUM*=50;  **public** **static** **final** **int** *L\_VOLUM*=50;  //设置游戏背景音乐循环标志  **public** **static** **final** **boolean** *LOOP\_BACKGROUNG\_MUSIC*=**true**;      //判断游戏是否结束  **public** **static** **boolean** *gameNoOver*=**true**;  //结束游戏回到主菜单的标志  **public** **static** **final** **int** *endGame*=1010;  **public** **static** Handler *handler*;  //飞机爆炸效果图Id  **public** **static** **final** **int**[] *PLANE\_EXPLODE\_IMAGE\_ID*={R.drawable.*bomb\_enemy\_0*,R.drawable.*bomb\_enemy\_1*,  R.drawable.*bomb\_enemy\_2*,R.drawable.*bomb\_enemy\_3*,R.drawable.*bomb\_enemy\_4*,R.drawable.*bomb\_enemy\_5*  };  **public** **static** **float** *planePower* = 5; //飞机子弹攻击力  **public** **static** **float** *monsterPower* = 20; //怪兽子弹攻击力  **public** **static** **int** *killCount* = 0; //杀敌数    //处理将战绩显示到文本框的handler  **public** **static** Handler *mainHandler*;      //设置是否是正常模式 还是随机模式 true表示正常模式，false表示随机模式  **public** **static** **boolean** *normalOrRandom*=**true**;  //背景音乐开与关  **public** **static** **boolean** *onOffFlag*=**true**;    **public** **static** **final** **int**[] *dead\_monsters* = **new** **int**[]{ //怪兽死亡的位图  };  //怪兽死亡时的音效  **public** **static** **final** **int** *DEAD\_MONSTER\_SOUND* = 0;  **public** **static** **void** initSoundPool(){ //初始化声音池  *sp*=**new** SoundPool(  5, //maxStreams参数，该参数为设置同时能够播放多少音效  AudioManager.*STREAM\_MUSIC*, //streamType参数，该参数设置音频类型，在游戏中通常设置为：STREAM\_MUSIC  0 //srcQuality参数，该参数设置音频文件的质量，目前还没有效果，设置为0为默认值。  );  *spMap*=**new** HashMap<Integer,Integer>();  *spMap*.put(1, *sp*.load(*context*, R.raw.*bullet*, 1));  *spMap*.put(2, *sp*.load(*context*, R.raw.*enemy4\_out*, 1));  *spMap*.put(3, *sp*.load(*context*, R.raw.*game\_over*, 1));  }  @SuppressWarnings("static-access")  **public** **static** **void** playSound(**int** sound,**int** number){ //播放声音,参数sound是播放音效的id，参数number是播放音效的次数  AudioManager am=(AudioManager)*context*.getSystemService(*context*.*AUDIO\_SERVICE*);//实例化AudioManager对象  **float** audioMaxVolumn=am.getStreamMaxVolume(AudioManager.*STREAM\_MUSIC*); //返回当前AudioManager对象的最大音量值  **float** audioCurrentVolumn=am.getStreamVolume(AudioManager.*STREAM\_MUSIC*);//返回当前AudioManager对象的音量值  **float** volumnRatio=audioCurrentVolumn/audioMaxVolumn;  *sp*.play(  *spMap*.get(sound), //播放的音乐id  volumnRatio, //左声道音量  volumnRatio, //右声道音量  1, //优先级，0为最低  number, //循环次数，0为不循环，-1为永远循环  1 //回放速度 ，该值在0.5-2.0之间，1为正常速度  );  }  }  【Bullet.java】飞机大战中子弹的飞行是相当重要的一个环节，reliveBullet部分借鉴了CSDN上 “v月下鸿鹄” 的思路  package cn**.**cd**.**gamerole**;**  **import** cn**.**cd**.**constant**.**GConstant**;**  **import** android**.**content**.**Context**;**  **import** android**.**graphics**.**Bitmap**;**  **import** android**.**graphics**.**BitmapFactory**;**  **import** android**.**graphics**.**Canvas**;**  **import** android**.**graphics**.**Paint**;**  **import** android**.**graphics**.**Rect**;**  /\*\*  \* 子弹类，每一列子弹都需要成为一个链表。  \* 子弹撞击到怪兽或飞出屏幕时，自动失效。  \* 子弹在射击的过程中是用线性表保存的  \* @author okcd00 部分摘自“v月下鸿鹄”  \*/  public class Bullet **{**    //子弹的位置  public float bulletX**=**0**,** bulletY**=**0**;**  //子弹的尺寸  public float width**,** height**;**  //子弹是否需要被绘制  public boolean isFire**=false;**  //下一颗子弹  public Bullet nextBullet **=** **null;**  //子弹的位图  public Bitmap bulletMap**;**  //子弹位图的位置矩形  public Rect rectBullet**;**  //子弹的两个方向的飞行速度  public float speedX **=** 0f**,** speedY **=** 0**-**GConstant**.**speed\_middle**;**    public Bullet**(**Context context**,** int resId**){**  //初始化子弹的位图  bulletMap **=** BitmapFactory**.**decodeResource**(**context**.**getResources**(),** resId**);**  //初始化子弹的位置矩形  rectBullet **=** **new** Rect**();**  //后续指针为空  nextBullet **=** **null;**  //设置子弹尺寸  width **=** bulletMap**.**getWidth**();**  height **=** bulletMap**.**getHeight**();**  **}**    public void flyBullet**(**Canvas canvas**,** Paint paint**)** **{**  //更新子弹的横纵坐标  bulletY**+=**speedY**;**  bulletX**+=**speedX**;**  //更新子弹的位置矩形  rectBullet**.**set**((**int**)**bulletX**,** **(**int**)**bulletY**,** **(**int**)(**bulletX**+**width**),** **(**int**)(**bulletY**+**height**));**  //绘制子弹  canvas**.**drawBitmap**(**bulletMap**,** bulletX**,** bulletY**,** paint**);**  **}**    public void reliveBullet**(**float x**,** float y**,** float differ**)** **{**  bulletX **=** x**+**differ**;**  bulletY **=** y**;**  isFire **=** **true;**  **}**  public void reliveBullet**(**float x**,** float y**,** float sx**,** float sy**)** **{**  bulletX **=** x**;**  bulletY **=** y**;**  isFire **=** **true;**  speedX **=** sx**;**  speedY **=** sy**;**  **}**  **}**  【GameBackground.java\_part】滚动地图实现部分  public GameBackground**(**Context context**,**int**[]** maps**){**  map1**=**BitmapFactory**.**decodeResource**(**context**.**getResources**(),** maps**[**0**]);**  map2**=**BitmapFactory**.**decodeResource**(**context**.**getResources**(),** maps**[**1**]);**  rect0**=new** Rect**(**0**,**0**,**map1**.**getWidth**(),**map1**.**getHeight**());**  rect1**=new** Rect**(**bgX**,**bg1Y**,**screemX**,**screemY**);**  rect2**=new** Rect**(**bgX**,**bg2Y**,**screemX**,**0**);**  **}**  public void drawBackground**(**Canvas canvas**,**Paint paint**){**  rect1**=new** Rect**(**bgX**,**bg1Y**,**screemX**,**screemY**+**bg1Y**);**  rect2**=new** Rect**(**bgX**,**bg2Y**,**screemX**,**screemY**+**bg2Y**);**  canvas**.**drawBitmap**(**map1**,**rect0**,**rect1**,null);**  canvas**.**drawBitmap**(**map2**,**rect0**,**rect2**,null);**  scrollBackground**(**screemY**);**  **}**  private void scrollBackground**(**int screemY**){**  bg1Y**+=**1**;**  bg2Y**+=**1**;**  **if(**bg1Y**>=**screemY**){**  bg1Y**=-**screemY**;**  **}**  **if(**bg2Y**>=**screemY**){**  bg2Y**=-**screemY**;**  **}**  **}**  【Monster.java】怪兽类  **package** cn.cd.gamerole;  **import** java.util.Random;  **import** android.content.Context;  **import** android.graphics.Bitmap;  **import** android.graphics.BitmapFactory;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.Rect;  **import** cn.cd.constant.GConstant;  **import** cn.cd.gameanim.Animation;  **import** cn.liwang.monster.R;  /\*\*  \* 泛怪兽类，作为游戏中玩家要消灭的对象，这个类的不同属性的对象会让游戏中有不同的怪兽出现。  \* 怪兽的初始化（初始化）都在此类的构造函数当中，怪兽的飞行路线以及生命状态的变化都由这个类当中的方法执行。  \* 怪兽有两种生命周期，分别是是：①出生->负伤->被击杀->重生。②出生->飞出屏幕->重新回到战场。  \* 飞出屏幕的怪兽生命值自动变为-1234，而生命值是作为怪兽是否存活的标杆属性。 本类只进行单个怪兽类的定义，对怪兽类数组的管理在GameMainView中进行。  \* 怪兽也可以发射子弹，由调用方类控制怪兽何时发射子弹，每只怪兽只可在屏幕上保留一颗子弹。  \*  \* **@author** okcd00  \*\*/  **public** **class** Monster {  // 调用方的上下文  **public** Context context;  // 怪兽的位置坐标，以及初始横坐标  **public** **float** monsterX = 0f, monsterY = 0f;  // 怪兽上次死亡的位置  **public** **float** deadX = 0f, deadY = 0f;  // 怪兽的尺寸  **public** **float** width, height;  // 怪兽的最大生命值和现有生命值，现有生命值也用来判断怪兽是否存活  **public** **int** blood = 0;  **private** **final** **int** maxblood;  // 怪兽位图  **public** Animation monsterAnim = **null**;  **public** Animation monsterDeadAnim=**null**;  // 当怪兽活着的时候，这个值为true，使得动画持续，怪兽生命周期完成后值为false，动画暂停。  **public** **boolean** isLoop = **true**;  // 这只怪兽的下一只怪兽  **public** Monster nextMonster = **null**;  // 设定怪兽横向移动的移动幅度  **public** **float** xMove = 0;  // 怪兽的单次偏移量  **public** **float** perMove = 0;  // 怪兽已经偏移的量  **public** **float** hasMove = 0;  // 怪兽的位置矩形，碰撞检测时使用  **public** Rect rectMonster = **null**;  // 怪兽所发射的子弹  **public** Bullet monsterBullet;  // 玩家的飞机  **public** MyPlane plane;  // 怪兽刷屏计数器  **public** **int** drawMonsterCount = 0;  // 屏幕尺寸  **public** **float** screemX, screemY;  // 用于显示怪兽血条矩形框的矩形  **public** Rect rectMaxBlood;  // 用于显示怪兽血量的矩形  **public** Rect rectBlood;  **public** Monster(Context context, **int**[] resId,**int**[] deadID, **boolean** isLoop, **float** screemX,  **float** screemY, **int** resMonsterBulletId, MyPlane plane) {  // 初始化屏幕尺寸  **this**.screemX = screemX;  **this**.screemY = screemY;  isLoop = **true**;  monsterAnim = **new** Animation(context, resId, isLoop);  monsterDeadAnim=**new** Animation(context,deadID,**false**);  // 设定怪兽的尺寸，这里要求怪兽的动画帧图尺寸都一样  Bitmap bmp = BitmapFactory.*decodeResource*(context.getResources(),  R.drawable.*monster\_03*);  // Bitmap bmp2 = BitmapFactory.decodeResource(context.getResources(),  // R.drawable.monster\_06);  // float w = bmp2.getWidth(),h=bmp2.getHeight();  width = bmp.getWidth();  height = bmp.getHeight();  // 用随机数设置怪兽的初始X坐标值  Random r = **new** Random();  monsterX = r.nextInt((**int**) (screemX - width));  // 设置怪兽的初始Y坐标  monsterY = 0 - height;  // 设定怪兽的生命值,最大生命值  maxblood = blood = 100;  // 初始化怪兽的位置矩形  rectMonster = **new** Rect();  // 初始化怪兽所发射的子弹，并设置属性  monsterBullet = **new** Bullet(context, resMonsterBulletId);  initMonsterBullet();  // 初始化玩家的飞机  **this**.plane = plane;  // 初始化context  **this**.context = context;  // 初始化怪兽血条方格矩形和怪兽血量矩形  rectMaxBlood = **new** Rect();  rectBlood = **new** Rect();  // 初始化怪兽死亡音效  }  /\*\*  \* 初始化怪兽子弹状态  \*/  **public** **void** initMonsterBullet() {  monsterBullet.isFire = **false**;  monsterBullet.bulletX = monsterX + (width / 2);  monsterBullet.bulletY = monsterY + (height / 4); // 让子弹处于怪兽的嘴边  }  **public** **void** monsterFly(Canvas canvas, Paint paint) {  // 常规飞行  monsterY += 2;  **if** (blood > 0 && blood <= maxblood / 2  && Math.*abs*(hasMove) < Math.*abs*(xMove)) {  // 如果怪兽的血量低于一半，则偏移飞行  monsterX += perMove;  hasMove += perMove;  }  // 更新怪兽位置矩形  rectMonster.set((**int**) (monsterX + width \* 0.25), (**int**) monsterY,  (**int**) (monsterX + width \* 0.8),  (**int**) ((monsterY + height) - height \* 0.3));  // 绘制矩形  monsterAnim.DrawAnimation(canvas, paint, monsterX, monsterY);  // canvas.drawRect(rectMonster, paint);  drawMonsterCount++;  // 判断怪兽是否可以产生子弹了  **if** (drawMonsterCount >= GConstant.*drawMonsterBullet*  && !monsterBullet.isFire) {  drawMonsterCount = 0;  // 计算子弹应该在两个方向上的路程  **float** dx = plane.planeX + plane.width / 2 - monsterBullet.bulletX, dy = plane.planeY  + plane.height / 2 - monsterBullet.bulletY;  // 设置怪兽子弹的横纵飞行速度  **float** speedX, speedY, a = dy / dx;  **if** (dx > 0) {  speedX = (**float**) Math.*sqrt*(GConstant.*speed\_slow*  \* GConstant.*speed\_slow* / (1 + a \* a));  } **else** {  speedX = 0 - (**float**) Math.*sqrt*(GConstant.*speed\_slow*  \* GConstant.*speed\_slow* / (1 + a \* a));  }  **if** (dy > 0) {  speedY = Math.*abs*(speedX \* a);  } **else** {  speedY = (0 - Math.*abs*(speedX)) \* Math.*abs*(a);  }  // 重置子弹属性，让其准备好射出  monsterBullet.reliveBullet(monsterX + (width / 2), monsterY  + (height / 4), // 让子弹处于怪兽的嘴边  speedX, speedY);  monsterBullet.isFire = **true**;  }  // 在这里设置怪兽的子弹  **if** (monsterBullet.isFire) { // 当子弹有效时  **if** (monsterBullet.bulletY > 0 - monsterBullet.height  && monsterBullet.bulletY <= screemY  && monsterBullet.bulletX > 0 - monsterBullet.width  && monsterBullet.bulletX < screemX) {  // 如果子弹没有射出屏幕，继续飞行  monsterBullet.flyBullet(canvas, paint);  } **else** { // 如果子弹射出屏幕，失效  initMonsterBullet();  }  // 碰撞检测  **if** (plane.rectPlane.intersect(monsterBullet.rectBullet)  || plane.rectPlane.contains(monsterBullet.rectBullet)) { // 如果怪兽子弹击中飞机  plane.blood -= GConstant.*monsterPower*;  monsterBullet.isFire = **false**;  }  }  // 怪兽与飞机的碰撞检测  **if** (plane.rectPlane.intersect(rectMonster)  || plane.rectPlane.contains(rectMonster)) { // 如果怪兽和飞机相撞  plane.blood = -1; // 飞机直接死亡  }  drawMonsterMessage(canvas);  }  **public** **void** drawMonsterMessage(Canvas canvas) {  Paint p = **new** Paint();  **float** lenBlood = (**float**) (width \* 0.55 \* blood / maxblood); // 血量长度  // 设置血条方框矩形和现有血量矩形  rectMaxBlood.set((**int**) (monsterX + 0.2 \* width),  (**int**) (monsterY + height), (**int**) (monsterX + 0.75 \* width),  (**int**) (monsterY + height + 5));  rectBlood.set((**int**) (monsterX + 0.2 \* width),  (**int**) (monsterY + height),  (**int**) (monsterX + 0.2 \* width + lenBlood), (**int**) (monsterY  + height + 5));  // 设置画笔，并绘制血条方格和血量  p.setColor(Color.*RED*);  p.setStyle(Style.*FILL*);  canvas.drawRect(rectBlood, p);  p.setColor(Color.*BLACK*);  p.setStyle(Style.*STROKE*);  canvas.drawRect(rectMaxBlood, p);  }  **public** **void** toDeadMonster(Canvas canvas, Paint paint, **int** soundID,**long** drawCount) {  // 保存怪兽死亡时的坐标  monsterDeadAnim.DrawAnimation(canvas, paint, monsterX, monsterY);  // 怪兽的死亡音效  **if**(GConstant.*onOffFlag*)**if**(drawCount%2==0)GConstant.*playSound*(2,1);  }  **public** **void** recoverMonster() {  // 随机生成X坐标  Random r = **new** Random();  monsterX = r.nextInt((**int**) (screemX - width));  **if** (**new** Random().nextBoolean()) { // 有一半的几率使飞机可以横向飞行  xMove = screemX / 2 - monsterX;  // 如果横向偏移量过小，则适当增大  **if** (Math.*abs*(xMove) < Math.*abs*(screemX / 6))  xMove \*= 2.8;  // 设置每次刷屏时的偏移量，使得20次刷屏后，飞机完成横向偏移  perMove = xMove / 20;  } **else** { // 飞机无横向移动  xMove = 0;  }  monsterY = -100;  blood = maxblood;  hasMove = 0;  initMonsterBullet();  }  }  【MyPlane.java】我方飞机类  **package** cn.cd.gamerole;  **import** android.content.Context;  **import** android.graphics.Bitmap;  **import** android.graphics.BitmapFactory;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Rect;  **public** **class** MyPlane {  //屏幕的尺寸  **public** **float** screemX, screemY;  // 飞机的位置，初始化位置固定  **public** **float** planeX, planeY;  // 飞机的尺寸  **public** **float** width, height;  // 飞机是否需要被绘制  **public** **boolean** isFire = **false**;  // 飞机的位图  **public** Bitmap planeMap;  // 飞机位图的位置矩形  **public** Rect rectPlane;  //飞机的最大生命值，固定为100  **public** **final** **float** maxblood = 500;  //飞机的实时生命值  **public** **float** blood;    **public** MyPlane(Context context, **int** resId, **float** screemX, **float** screemY) {  //获取屏幕尺寸  **this**.screemX = screemX;  **this**.screemY = screemY;  //得到位图  planeMap = BitmapFactory.*decodeResource*(context.getResources(), resId);  //得到飞机矩形  rectPlane = **new** Rect((**int**)planeX +30, (**int**)planeY+15, (**int**)(planeX+width)-30, (**int**)(planeY+height)-10);  //设置飞机尺寸  width = planeMap.getWidth();  height = planeMap.getHeight();  //初始化飞机当前生命值  blood = maxblood;  //初始化飞机坐标  planeX = screemX/2-width/2;  planeY = screemY;  }    **public** **void** planeFly(**float** x, **float** y, **float** a, **float** b) {  **float** chaX=Math.*abs*((x - a)), chaY=Math.*abs*((y - b));  **if** (x - a < 0 && y - b < 0) {  planeX += chaX;  planeY += chaY;  } **else** **if** (x - a > 0 && y - b > 0) {  planeX -= chaX;  planeY -= chaY;  } **else** **if** (x - a > 0 && y - b < 0) {  planeX -= chaX;  planeY += chaY;  } **else** **if** (x - a < 0 && y - b > 0) {  planeX += chaX;  planeY -= chaY;  } **else** **if** (x - a == 0 && y - b > 0) {  planeY -= chaY;  } **else** **if** (x - a == 0 && y - b < 0) {  planeY += chaY;  } **else** **if** (x - a > 0 && y - b == 0) {  planeX -= chaX;  } **else** **if** (x - a < 0 && y - b == 0) {  planeX += chaX;  }  //控制飞机不飞出屏幕  **if** (planeX < 0) planeX = 0;  **else** **if** (planeX > screemX - width) planeX = screemX - width;  **else** **if** (planeY < 0) planeY = 0;  **else** **if** (planeY > screemY - height) planeY = screemY - height;    }    **public** **boolean** planeDraw(Canvas canvas, Paint paint){  canvas.drawBitmap(planeMap, planeX, planeY, paint);  //设置飞机的位置矩形  rectPlane.set((**int**)planeX +30, (**int**)planeY+15, (**int**)(planeX+width)-30, (**int**)(planeY+height)-10);  Paint p = **new** Paint();  p.setColor(Color.*RED*);  // canvas.drawRect(rectPlane, paint);  **if** (blood >= 0)**return** **true**;  **else** **return** **false**;  }  }  【GameMainView.java】安卓应用主要绘制部分  **package** cn.cd.gameview;  **import** java.util.Random;  **import** android.content.Context;  **import** android.graphics.Bitmap;  **import** android.graphics.BitmapFactory;  **import** android.graphics.Canvas;  **import** android.graphics.Color;  **import** android.graphics.Paint;  **import** android.graphics.Paint.Style;  **import** android.graphics.Rect;  **import** android.view.MotionEvent;  **import** android.view.SurfaceHolder;  **import** android.view.SurfaceView;  **import** cn.cd.constant.GConstant;  **import** cn.cd.gameanim.Animation;  **import** cn.cd.gamerole.Bullet;  **import** cn.cd.gamerole.GameBackground;  **import** cn.cd.gamerole.Monster;  **import** cn.cd.gamerole.MyPlane;  **import** cn.liwang.monster.R;  **public** **class** GameMainView **extends** SurfaceView **implements**  SurfaceHolder.Callback, Runnable {  **private** SurfaceHolder holder;  **private** **float** x = 40, y = 40; // 手指按下的坐标  **private** **boolean** flag = **true**;  **private** Canvas canvas;  **private** Paint paint;  **private** **float** screemX, screemY;  **private** **float** a, b; // 微量移动后手指的坐标  **final** **static** **int** *PLAN\_STEP* = 10;  **public** **float** chaX = 0;  **public** **float** chaY = 0;  **private** Monster aliveMonster = **null**; // 活着的怪兽的头指针，该队列有对头对象  **private** Monster headDeadMonster = **null**; // 死怪兽队列（带头节点）的头指针,活怪兽死了的时候向这里添加  **private** Monster tailDeadMonster = **null**; // 死怪兽队列的尾指针，死怪兽复活时用这个移出  **private** **int** drawCount = 300; // 怪兽用的刷屏计数器  **public** **int** drawMonsterCount = 300; // 刷几次屏出生一只怪兽  **public** **long** drawAddMonsterCount = 0;  **public** **int** drawInitPlane = 0; // 前几次刷屏  **private** MyPlane plane; // 玩家控制的飞机  **private** Bullet headUnuseBullet; //没有射出的子弹  **private** Bullet headFireBullet; //已射出的子弹  **private** **int** drawBulletCount; // 子弹用的刷屏计数器  **private** **boolean** isBulletLeft = **true**; // 控制下一颗子弹是否从左边射出  **private** Rect rectMsg = **new** Rect(), // 飞机的血条方格矩形  rectBlood = **new** Rect(); // 飞机的血条矩形  **private** **int** lenBlood; // 飞机血条的长度  **private** GameBackground gbground; // 背景对象  **private** Bitmap[] deadPlane; // 飞机死亡后的爆炸位图  **public** GameMainView(Context context) {  **super**(context);  // **TODO** Auto-generated constructor stub  holder = getHolder();  holder.addCallback(**this**);  paint = **new** Paint();  // 设置画笔颜色为白色  paint.setColor(Color.*BLACK*);  // 设置文本大小  paint.setTextSize(20);  // 设置焦点  setFocusable(**true**);  }  @Override  **public** **void** surfaceChanged(SurfaceHolder arg0, **int** arg1, **int** arg2, **int** arg3) {  // **TODO** Auto-generated method stub  }  @Override  **public** **void** surfaceCreated(SurfaceHolder arg0) {  // **TODO** Auto-generated method stub  screemX = **this**.getWidth();  screemY = **this**.getHeight();  GConstant.*initSoundPool*();  **if**(!GConstant.*normalOrRandom*)initBaseData();  **else** GConstant.*killCount*=0;  initScrollBackground();  // 初始化游戏信息  rectMsg.set((**int**) screemX - 105, 5, (**int**) screemX - 5, 20);  // 初始化飞机  plane = **new** MyPlane(getContext(), R.drawable.*wsparticle\_test\_001*, screemX, screemY);  initBullet();  initMonsters();  deadPlane = **new** Bitmap[GConstant.*PLANE\_EXPLODE\_IMAGE\_ID*.length];  **for** (**int** i = 0; i < deadPlane.length; i++) {  deadPlane[i] = BitmapFactory.*decodeResource*(getResources(),  GConstant.*PLANE\_EXPLODE\_IMAGE\_ID*[i]);  }  GConstant.*gameThread* = **new** Thread(**this**);  GConstant.*gameThread*.start();  }  @Override  **public** **void** surfaceDestroyed(SurfaceHolder arg0) {  // **TODO** Auto-generated method stub  flag = **false**;  }  @Override  **public** **boolean** onTouchEvent(MotionEvent event) {  // **TODO** Auto-generated method stub  **switch** (event.getAction()) {  **case** MotionEvent.*ACTION\_DOWN*:  x = event.getX();  y = event.getY();  **break**;  **case** MotionEvent.*ACTION\_UP*:  **break**;  **case** MotionEvent.*ACTION\_MOVE*:  **if**(drawInitPlane>120) {  a = event.getX();  b = event.getY();  plane.planeFly(x, y, a, b);  x = a;  y = b;  **break**;  }  }  **return** **true**;  }  **public** **void** myDraw() {  **switch** (GConstant.*gameState*) {  **case** 1001:  canvas.drawColor(Color.*GREEN*);  gbground.drawBackground(canvas, paint);  **if** (GConstant.*gameNoOver*) {  GConstant.*gameNoOver* = plane.planeDraw(canvas, paint);  **if** (!GConstant.*gameNoOver*) {  Animation pEAnim = **new** Animation(getContext(), deadPlane,  **false**);  pEAnim.DrawAnimation(canvas, paint, plane.planeX,  plane.planeY);  }  } **else** {  flag = **false**;  **try** {  Thread.*sleep*(2000);  } **catch** (Exception e) {  }  GConstant.*handler*.sendEmptyMessage(GConstant.*endGame*);  }  firstPlaneFly(); // 飞机的开场飞行  drawMessage();  bulletFire();  ctrlMonster(); // 绘制并控制怪兽  **break**;  **case** 0:  **try** {  Thread.*sleep*(Integer.*MAX\_VALUE*);  } **catch** (Exception e) {  }  **break**;  }  }  **public** **void** run() {  **while** (flag) {  **long** chatime = 0;  **long** start = System.*currentTimeMillis*();  **synchronized** (holder) {  canvas = holder.lockCanvas();  myDraw();  holder.unlockCanvasAndPost(canvas);  }  **long** end = System.*currentTimeMillis*();  chatime = end - start;  **try** {  **if** (chatime < 15) {  Thread.*sleep*(15 - chatime);  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  drawCount++;  drawBulletCount++;  drawAddMonsterCount++;  }  }    **private** **void** initBaseData(){  Random r=**new** Random();  GConstant.*bulletCount*=r.nextInt(15)+3;  GConstant.*planePower*=r.nextInt(15)+10;  GConstant.*monsterPower*=r.nextInt(20)+15;  GConstant.*numMonsters*=r.nextInt(13)+12;  GConstant.*numBullets*=r.nextInt(10)+10;  GConstant.*killCount*=0;  GConstant.*drawCount*=r.nextInt(60)+60;  GConstant.*drawMonsterBullet*=r.nextInt(10)+20;  GConstant.*extraDamage*=r.nextInt(4)\*0.01f+0.01f;  }  **public** **void** firstPlaneFly() {  drawInitPlane++;  **if** (drawInitPlane <= 70) {  plane.planeY -= screemY/130;  } **else** **if** (drawInitPlane > 70 && drawInitPlane < 120) {  plane.planeY += screemY/110;  }  }    **private** **void** initScrollBackground() {  gbground = **new** GameBackground(getContext(), GConstant.*GAME\_BG\_IMAGE\_ID*);  gbground.bg2Y = -(**int**) screemY;  gbground.screemX = (**int**) screemX;  gbground.screemY = (**int**) screemY;  }  **public** **void** drawMessage() {  // 绘制生命值信息  Paint p = paint;  canvas.drawText((String) getResources().getText(R.string.*plane\_blood*),  (**int**) screemX - 140, 20, p);  canvas.drawText((String) getResources().getText(R.string.*kill\_num*),  (**int**) screemX - 140, 50, p);  // 绘制当前生命值（血条）  lenBlood = (**int**) (100 \* plane.blood / plane.maxblood);  rectBlood.set((**int**) screemX - 105, 5, (**int**) screemX - 105 + lenBlood,  20);  p.setStyle(Style.*FILL*);  p.setColor(Color.*RED*);  canvas.drawRect(rectBlood, paint);  // 绘制血条的方格  p.setStyle(Style.*STROKE*);  p.setColor(Color.*BLACK*);  canvas.drawRect(rectMsg, paint);  // 绘制玩家的杀敌数  canvas.drawText("" + GConstant.*killCount*, (**int**) screemX - 95, 50, p);  }  **public** **void** initBullet() {  headFireBullet = **new** Bullet(getContext(), R.drawable.*bullet*);  headUnuseBullet = **new** Bullet(getContext(), R.drawable.*bullet*);  Bullet workB = headUnuseBullet;  **for** (**int** i = 0; i < GConstant.*numBullets*; i++) {  workB.nextBullet = **new** Bullet(getContext(), R.drawable.*bullet*);  workB = workB.nextBullet;  workB.isFire = **false**;  }  }  **public** **void** bulletFire() {  Bullet workB = headUnuseBullet.nextBullet;  **if** (drawBulletCount >= GConstant.*bulletCount* && workB != **null**) {  // 如果刷屏计时器达到规定值，则射出一颗子弹  // 重置子弹的刷屏计时器  drawBulletCount = 0;  // 从无效子弹表中删掉一颗子弹  headUnuseBullet.nextBullet = workB.nextBullet;  // 把刚刚删掉的子弹插入到有效子弹表中  workB.nextBullet = headFireBullet.nextBullet;  headFireBullet.nextBullet = workB;  // 设置子弹从左或者右射出  **float** differ = plane.width / 6;  **if** (!isBulletLeft)  differ \*= 5;  // 执行子弹重新装填工作，装填后的子弹将在下一个循环射出  workB.reliveBullet(plane.planeX-13, plane.planeY, differ);  // 更改下次子弹的左右  isBulletLeft = !isBulletLeft;  **if**(GConstant.*onOffFlag*)GConstant.*playSound*(1,1);  }  // 更新已射出的子弹的位置  workB = headFireBullet;  **while** (workB.nextBullet != **null**) {  **if** (workB.nextBullet.isFire) { // 如果这颗子弹有效  **if** (workB.nextBullet.bulletY > 0 - workB.nextBullet.height) {  // 如果子弹没有射出屏幕，继续飞行  workB.nextBullet.flyBullet(canvas, paint);  } **else** { // 如果子弹射出屏幕，失效  workB.nextBullet.isFire = **false**;  }  } **else** { // 如果子弹被设置为无效  // 将子弹从有效队列中移出  Bullet outBullet = workB.nextBullet;  workB.nextBullet = outBullet.nextBullet;  // 将子弹移进无效队列中  outBullet.nextBullet = headUnuseBullet.nextBullet;  headUnuseBullet.nextBullet = outBullet;  **if** (workB.nextBullet == **null**)  **break**; // 预防溢出  }  workB = workB.nextBullet;  }  }  **public** **void** initMonsters() {  // 初始化死怪兽队列，死怪兽队列有头结点  headDeadMonster = **new** Monster(getContext(), GConstant.*monsterAnimation*,GConstant.*monsterDeadID*,  **true**, screemX, screemY, GConstant.*monsterBullet*, plane);  // 定义这个函数中的工作指针  Monster workM = headDeadMonster;  // 初始化若干个怪兽  **for** (**int** i = 0; i < GConstant.*numMonsters*; i++) {  workM.nextMonster = **new** Monster(getContext(),  GConstant.*monsterAnimation*,GConstant.*monsterDeadID*, **true**, screemX, screemY,  GConstant.*monsterBullet*, plane);  workM = workM.nextMonster;  }  tailDeadMonster = workM;  // 初始化活怪兽队列的头结点  aliveMonster = **new** Monster(getContext(), GConstant.*monsterAnimation*,GConstant.*monsterDeadID*,  **true**, screemX, screemY, GConstant.*monsterBullet*, plane);  }    // \* 通过碰撞检测，以及怪兽定位判断怪兽是否被子弹击中或者飞出屏幕 即控制怪兽的死亡。    **public** **void** isMonsterDead() {  // 初始化怪兽工作指针  Monster workM = aliveMonster;  **while** (workM.nextMonster != **null**) {  workM = workM.nextMonster;  // 判断怪兽是否在屏幕内  **if** (workM.monsterX < screemX && workM.monsterY < screemY) { // 如果怪兽还在屏幕内，则进行碰撞检测  // 初始化子弹工作指针  Bullet workB = headFireBullet;  **while** (workB.nextBullet != **null**) {  workB = workB.nextBullet;  **if** (workM.rectMonster.intersect(workB.rectBullet)  || workM.rectMonster.contains(workB.rectBullet)) { // 当子弹击中怪兽时  workM.blood -= GConstant.*planePower*;  workB.isFire = **false**;  }  }  } **else** { // 如果怪兽已飞出屏幕  workM.blood = -1234;  }  }  }  **public** **void** ctrlMonster() {  // 判断是否调整怪兽出现速度  **if** (drawAddMonsterCount >= GConstant.*speedAddMonster*) {  drawMonsterCount /= 1.5;  drawAddMonsterCount = 0;  }  **if** (drawCount >= drawMonsterCount  && headDeadMonster.nextMonster != **null**  && tailDeadMonster != **null**) {  // 当刷屏线程已经执行了30次时，让最早的一只死了的怪兽复活  drawCount = 0;  // 将那只最早死亡的怪兽放到活怪兽队列中，让其成为队头  tailDeadMonster.nextMonster = aliveMonster.nextMonster;  aliveMonster.nextMonster = tailDeadMonster;  // 重置将要复活的怪兽的属性  tailDeadMonster.recoverMonster();  // 将复活的怪兽从死怪兽队列中移出  Monster workM = headDeadMonster;  **while** (workM.nextMonster != tailDeadMonster) {  workM = workM.nextMonster;  }  workM.nextMonster = **null**;  tailDeadMonster = workM;  }  // 绘制所有的活着的怪兽，并让活怪兽队列中的死怪兽进入死怪兽队列  Monster drawM = aliveMonster;  **while** (drawM.nextMonster != **null**) {  // 判断怪兽是否死亡  **if** (drawM.nextMonster.blood <= 0) {  // 判断怪兽是不是被玩家打死的  **if** (drawM.nextMonster.blood != -1234) { // 如果是  GConstant.*killCount*++; // 杀敌数+1  // 每杀死一只怪兽，飞机子弹攻击力+0.01  GConstant.*planePower* += GConstant.*extraDamage*;  // 执行怪兽死亡效果方法。  // drawM.nextMonster.toDeadMonster(canvas, paint, GConstant.DEAD\_MONSTER\_SOUND, drawAddMonsterCount);  }  // 如果怪兽已经死亡，则移至死亡怪兽队列  deadMonster(drawM);  } **else** { // 如果怪兽还活着，则继续绘制  drawM.nextMonster.monsterFly(canvas, paint);  }  **if** (drawM.nextMonster != **null**)  drawM = drawM.nextMonster;  **else**  **break**;  }  isMonsterDead();  }  **public** **void** deadMonster(Monster lastM) {  Monster workM = lastM.nextMonster;  // 移出活怪兽队列  lastM.nextMonster = workM.nextMonster;  // 放入死怪兽队列  workM.nextMonster = headDeadMonster.nextMonster;  headDeadMonster.nextMonster = workM;  **if** (headDeadMonster == tailDeadMonster)  tailDeadMonster = workM;  }  **public** **void** onPause() {  GConstant.*gameState* = 0;  }  **public** **void** onResume() {  flag = **true**;  GConstant.*gameState* = 1001;  //GConstant.gameThread.interrupt();  //DELETE THIS　ＳＥＮＴＥＮＳＥ  }  **public** **void** onDestory() {  flag = **false**;  GConstant.*gameState* = 0;  GConstant.*gameThread*.interrupt();  }  } | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  以下将以图文的方式对实验结果与调试过程进行说明：    首先，由于大家全都是电脑端的JAVA应用，我之前有做过一些安卓开发，故尝试做APP    对所有的类进行一下简要的说明：  ·同PC端的应用一样，Bullet子弹类，Monster怪兽类，MyPlane主角类是飞机大战的主要部分，提供了游戏中的移动、攻击等功能  ·GConstant为常量类，用于存放各类常用常量  ·Animation、GameBackground、Anim为游戏界面的图像图形类  ·EnterGame\ExitGame\GameMainMenu\PauseGame\SetGame\HomeKeyBroadReceive一系列均为游戏常用类，包括进入、退出、主菜单、暂停、继续、Home键效果  ·ShareResult为分享类，安卓游戏开发常见类  ·GameMusic声音类，使用声音池进行音效和BGM的实现    手机中此APP的图标[Attack\_on\_Lucian]与启动页、欢迎页截图 （那个PNG显示黑底很奇怪……ADT的模拟器中是没有黑底的，在Samsung Note上运行就会有黑底）    左上角的设置，可以设置是否播放音乐以及游戏模式（更改的是敌人出现的频率与增长速度参数），右下角的开始战斗点击后会变成右图所示的圆圈旋转代表loading    游戏界面，可以明显地看出敌人的子弹是朝向英雄方向发射的，右上角的血条代表生命    结束时会出现根据击杀数而变化的结束语，并打印出各种参数，更新历史成绩（没有做文件储存，所以游戏进程关闭前会正常更新，被关闭后会重置）。    在游戏中点击返回按钮会弹出暂停页面，点击继续会继续游戏，也可重开或者返回菜单  F:\【资料】重大教学资料\JAVA\Monster\res\drawable-mdpi\bomb_enemy_0.png F:\【资料】重大教学资料\JAVA\Monster\res\drawable-mdpi\bomb_enemy_1.png F:\【资料】重大教学资料\JAVA\Monster\res\drawable-mdpi\bomb_enemy_2.png F:\【资料】重大教学资料\JAVA\Monster\res\drawable-mdpi\bomb_enemy_3.png  特别地，当直接撞击敌人会直接判负，并产生爆炸效果  注：安卓apk以及源码压缩包随本实验报告一同压缩提交。本次提交延时是因为前往北京参加ACM-ICPC亚洲区域赛北京赛区现场赛（结果为铜牌），故返程后继续完善至今方才提交，给老师和助教添麻烦了实在不好意思。 学生/陈点 20125209 物联网1班 | | | | | | |