浙 江 大 學

实验报告

课程名称：智能终端软件开发技术

APP 名称：星空飞机大战

专业：信息安全

成员 1 姓名：许多 学号：3160103958

成员 2 姓名：吕慕凡 学号：3160102463

电子邮件地址：3160103958@zju.edu.cn

手机：17342019969

任课老师：张 寅

目录

一．应用简介

1.1 背景

飞机大战是电脑游戏发展史中早期较为经典的游戏之一，无论是在电脑端、移动端还是游戏机上都能见到类似相关的射击类游戏。这款游戏曾经在那些终端上都是通过键盘或者手柄操作，而在智能触摸屏手机问世以后，触摸控制飞机让这款游戏的体验更好。几年前微信游戏中的全民飞机大战也曾经风靡一时，让我们感受到这款游戏如果在游戏简单的原理之上加以包装和对关卡和剧情进行合理设计，尽管这是非常老的游戏但还是非常能够吸引现在广大手机用户。甚至如果剧情合理或有新意还能吸引玩家充值游戏。

1.2 与同类 APP 的对比

触控：相比游戏机中和电脑终端中的使用键盘控制飞行的飞机大战游戏，我们选择利用现在智能手机的触控特点，让飞机的操控更加简单，交互性更强。

道具：我们给飞机设置了很多掉落道具，如果飞机触碰到这些道具，那么它将拥有一定时间的 BUFF 加成，提高游戏的趣味性。

背景：使用了年轻人喜爱的星空背景图片，带来更多星际元素，使得这个经典游戏不老套。

音效：初始界面使用游戏第五人格的登录界面音效《第五人格》，带来一种神秘色彩；战斗界面使用第五人格游戏的战斗准备音乐《盛宴再临》，增加战斗热情的同时还带来流行元素。

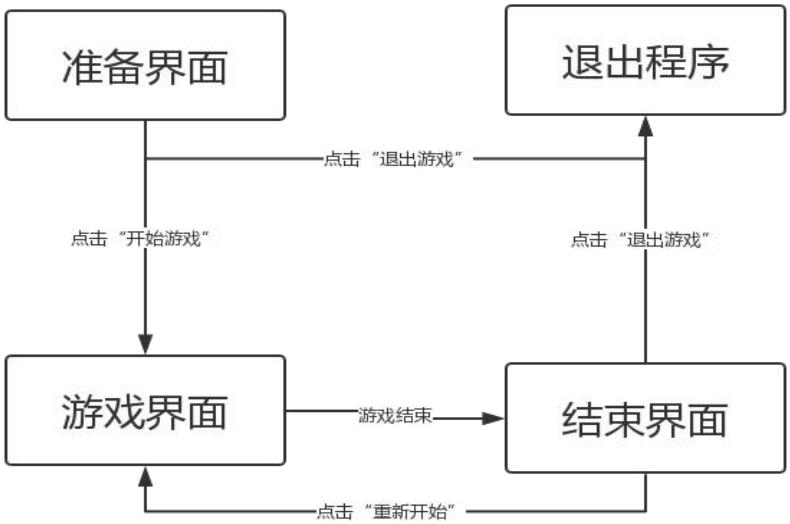
1.3 运行环境

1 号测试手机版本：Sumsung SM-G9009D (Android 6.0.1, API 23）
2 号测试手机版本：OPPO R9m (Android 5.1, API 22)

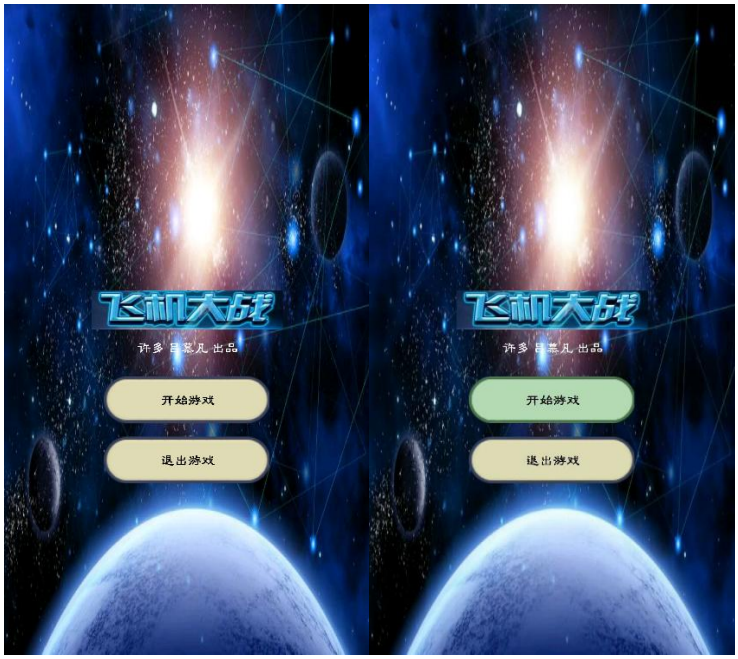
二．总体架构及功能设计

2.1 总体架构

整体分为三个界面：准备界面、游戏界面、结束界面



2.2 准备界面功能设计



准备界面主要功能：

1、展示游戏主界面
2、显示游戏名称
3、显示开发人员
4、两个按钮选择进入游戏和退出此程序

功能设计：

1、主界面设计：采用一张星空风格背景作为整个界面背景
2、游戏名称设计：采用艺术字截图而成的图片，放置在界面中央部分

3、开发人员显示：在游戏名称下方以白色文本形式显示开发人员信息

4、按钮设计：采用外部绘制的图形导为图片插入此界面的方法，在此图片上以文本形式显示按钮功能

2.3 游戏界面功能设计

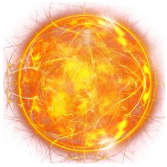
游戏界面主要功能：

- 打飞机游戏主体：战机会自动发射子弹，用手指控制战机改变战机方向
- 玩家飞机形态：三种形态，攻击方式跟子弹威力均不相同
- 敌方战机：四种战机，攻击方式、子弹威力和生命值均不同
- 辅助道具：导弹、子弹、生命道具等，可以增强战力
- 游戏特效：子弹效果、战机爆炸、导弹引爆、无敌光辉等
- 等级记录：等级随着分数的增加而提升，同时敌军也会越来越强力
- 分数记录：击落不同的敌机能获取到相应的分数，分数代表着战绩
- 暂停功能：点击以后暂停游戏，再次点击继续游戏

游戏玩法简介：

✧ 飞机机体及子弹介绍

➢ 我方战机：三种形态分别对应三种类型的子弹，从左至右威力依次升级



（导弹爆炸效果）

（无敌状态效果）

➢ 敌方战机——小型机

- 敌方战机——中型机

- 敌方战机——大型机：会发射有初始定位功能的子弹

- 敌方战机——boss 机：有四种颜色形态，右侧上下两种形态是狂化状态

 （普通子弹）  （可形成连续子弹流）  （在中部横向移动）

（两种从下往上发射的子弹，可形成阵列发射）

子弹的发射数量和杀伤力将逐步升级，难度逐渐提升。与 boss 的状态和被杀死的次数有关。

✧ 掉落物品介绍

生命值+1

导弹+1

获得 15 秒紫色子弹（二级子弹）

获得 15 秒红色子弹（三级子弹）

✧ 动画效果概览

2.4 结束界面功能设计

结束界面主要功能：

1、 展示结束界面
2、 显示“game over”
3、 显示当局游戏总分
4、 两个按钮选择重新开始游戏和退出此程序

功能设计：

1、 界面设计采取与准备界面相同风格的设计
2、 游戏结束标识：采用 PNG 图片的形式，放置在界面中央偏上部分

三．关键数据结构/算法

我们在设计时，为了更好地对工程中的关键变量及函数进行封装，定义了几个基类和它的派生类，将同一功能的类放置在相同目录下，便于管理和后续设计中增添新的功能。

相关类的结构如下所示：

- **GameObject　游戏对象基类**
  - GameGoods　——继承 object 基类的物品类
    - LifeGoods　物品—生命值
    - MissileGoods　物品—导弹
    - PurpleBulletGoods　物品—我方二级子弹
    - RedBulletGoods　物品—我方三级子弹
  - Bullet　——继承 object 基类的子弹类

    我方子弹
    - MyBlueBullet　　我方一级子弹类
    - MyPurpleBullet　我方二级子弹类
    - MyRedBullet　　我方三级子弹类

    敌方子弹
    - EnemyBullet　——继承 Bullet 类

      敌方-大型飞机子弹类
      - BigPlaneBullet

      敌方-boss 飞机子弹类
      - BossDefaultBullet　　boss 默认子弹类
      - BossFlameBullet　　boss 子弹流类
      - BossRHellfireBullet　boss 地狱火（红）子弹类
      - BossYHellfireBullet　boss 地狱火（黄）子弹类
      - BossSunBullet　　boss 闪光弹类
  - Plane　飞机机体类——继承 GameObject 基类
    - MyPlane　我方飞机类
    - EnemyPlane　敌方飞机类
      - SmallPlane　敌方—小型机
      - MiddlePlane　敌方—中型机
      - BigPlane　敌方—大型机
      - BossPlane　敌方—boss 机
- Factory　create 函数的集合
  - GameObjectFactory　封装所有物品的创建函数
- Constant 常量类
  - ConstantUtil　　常量宏定义
  - DebugConstant　调整相关参数定义
  - GameConstant　　游戏相关参数定义
- Sounds　声音类
  - GameSoundPool
- View 视图类

- BaseView　视图基类
  - ReadyView　准备界面视图
  - MainView　主游戏界面视图
  - EndView　结束界面视图
- MainActivity　UI 界面

3.1 游戏对象相关类设计

GameObject 类是游戏中所有对象的基类，其中声明定义了一个对象需要的所有参数和需要使用到的函数的接口。

```
protected int currentFrame;    // 帧数
protected int speed;           // 移动速度
protected float object_x;       // 物品的 x 坐标
protected float object_y;       // 物品的 y 坐标
protected float object_width;    // 物体宽度
protected float object_height;    // 物体高度
protected float screen_width;    // 屏幕宽度
protected float screen_height;  // 屏幕高度
protected boolean isAlive;       // 存活状态
protected Paint paint;           // 画笔
protected Resources resources;  // 资源文件
public GameObject(Resources resources) {  // 加载资源，新建画笔
    this.resources = resources;
    this.paint = new Paint();
}
public void setScreenWH(float screen_width, float screen_height) {
    this.screen_width = screen_width;
    this.screen_height = screen_height;
}  // 设置屏幕尺寸
public void initial(int arg0, float arg1, float arg2) {}  // 初始化
protected abstract void initBitmap();  // 载入图片资源
public abstract void drawSelf(Canvas canvas);  // 界面设计
public abstract void release();  // 释放资源
public boolean isCollide(GameObject obj) {  // 判断对象之间是否碰撞
    return true;
}
public void logic() {}  // 逻辑设计
// 用于传参的函数
public int getSpeed() {
    return speed;
}
public void setSpeed(int speed) {
    this.speed = speed;
```

```
}
public float getObject_x() {
    return object_x;
}
public void setObject_x(float object_x) {
    this.object_x = object_x;
}
public float getObject_y() {
    return object_y;
}
public void setObject_y(float object_y) {
    this.object_y = object_y;
}
public float getObject_width() {
    return object_width;
}
public void setObject_width(float object_width) {
    this.object_width = object_width;
}
public float getObject_height() {
    return object_height;
}
public void setObject_height(float object_height) {
    this.object_height = object_height;
}
public boolean isAlive() {
    return isAlive;
}
public void setAlive(boolean isAlive) {
    this.isAlive = isAlive;
}
```

在此基础上，对其他具有代表性功能的对象进行设计，定义新的参数和函数接口，并重写基类函数。

3.1.1 Goods 掉落物品设计

GameGoods 类继承了 GameObject 基类。在其中定义了新的成员变量，并重写了基类的部分函数，这些函数可以在 GameGoods 类的子类中调用，是所有掉落物品都需要实现的功能。

```
protected Bitmap bmp;  //图片资源对象
private int direction;  //移动方向
```

```
public GameGoods(Resources resources) {
    super(resources);
    this.speed = 10;
    Random ran = new Random();
    direction = ran.nextInt(2) + 3;
    initBitmap();
}

@Override
public void initial(int arg0, float arg1, float arg2) {
    isAlive = true;
    object_x = screen_width / 2 - object_width / 2;
    object_y = -object_height * (arg0 * 2 + 1);
}

@Override
protected void initBitmap() {}
//绘制物品
@Override
public void drawSelf(Canvas canvas) {
    if (isAlive) {
        canvas.save();
        canvas.clipRect(object_x, object_y, object_x + object_width,
object_y + object_height);
        canvas.drawBitmap(bmp, object_x, object_y, paint);
        canvas.restore();
        logic();
    }
}
//释放图片资源
@Override
public void release() {
    if (!bmp.isRecycled()) {
        bmp.recycle();
    }
}
//物品移动的逻辑
@Override
public void logic() {
    Random ran = new Random();
    // 左上方移动时的逻辑
    if (direction == ConstantUtil.DIR_LEFT_UP) {
        object_x -= ran.nextInt(3) + speed;
```

```
        object_y -= ran.nextInt(3) + speed;
        if (object_x <= 0 || object_y <= 0) {
            if (object_x <= 0)
                object_x = 0;
            else
                object_y = 0;
            int dir = 0;
            do {
                dir = ran.nextInt(4) + 1;
            }
            while (dir == direction);
            direction = dir;
            this.speed = 10 + ran.nextInt(5);
        }
    }
    // 右上方移动时的逻辑
    else if (direction == ConstantUtil.DIR_RIGHT_UP) {
        object_x += ran.nextInt(3) + speed;
        object_y -= ran.nextInt(3) + speed;
        if (object_x >= screen_width - object_width || object_y <= 0) {
            if (object_x >= screen_width - object_width)
                object_x = screen_width - object_width;
            else
                object_y = 0;
            int dir = 0;
            do {
                dir = ran.nextInt(4) + 1;
            }
            while (dir == direction);
            direction = dir;
            this.speed = 10 + ran.nextInt(5);
        }
    }
    // 左下方移动时的逻辑
    else if (direction == ConstantUtil.DIR_LEFT_DOWN) {
        object_x -= ran.nextInt(3) + speed;
        object_y += ran.nextInt(3) + speed;
        if (object_x <= 0 || object_y >= screen_height - object_height)
{
            if (object_x <= 0)
                object_x = 0;
            else
                object_y = screen_height - object_height;
            int dir = 0;
```

```
            do {
                dir = ran.nextInt(4) + 1;
            }
            while (dir == direction);
            direction = dir;
            this.speed = 10 + ran.nextInt(5);
        }
    }
    // 右下方移动时的逻辑
    else if (direction == ConstantUtil.DIR_RIGHT_DOWN) {
        object_x += ran.nextInt(3) + speed;
        object_y += ran.nextInt(3) + speed;
        if (object_x >= screen_width - object_width || object_y >=
screen_height - object_height) {
            if (object_x >= screen_width - object_width)
                object_x = screen_width - object_width;
            else
                object_y = screen_height - object_height;
            int dir = 0;
            do {
                dir = ran.nextInt(4) + 1;
            }
            while (dir == direction);
            direction = dir;
            this.speed = 10 + ran.nextInt(5);
        }
    }
}
//判断移动时是否与其他物品碰撞，掉落的物品需要判断的碰撞对象仅为我方飞机主体
@Override
public boolean isCollide(GameObject obj) {
    if (object_x <= obj.getObject_x()
            && object_x + object_width <= obj.getObject_x()) {
        return false;
    }
    else if (obj.getObject_x() <= object_x
            && obj.getObject_x() + obj.getObject_width() <= object_x) {
        return false;
    }
    else if (object_y <= obj.getObject_y()
            && object_y + object_height <= obj.getObject_y()) {
        return false;
    }
    else if (obj.getObject_y() <= object_y
```

```
            && obj.getObject_y() + obj.getObject_height() <= object_y) {
        return false;
    }
    isAlive = false;
    return true;
}
```

同属于物品类，GameGoods 的子类继承父类的函数，重写 initBitmap()函数加载属于自己的图片资源。

3.1.2 bullet 子弹设计

游戏中的子弹设计的基本功能由 bullet 类声明和定义。Bullet 继承了 GameObject 类的成员，重写基类函数，并定义了独属于自己的成员函数和成员变量。

```
protected int harm;  //子弹的伤害值
public Bullet(Resources resources) {
    super(resources);
    initBitmap();
}
@Override
protected void initBitmap() {}  //加载图片资源
@Override
public void drawSelf(Canvas canvas) {}   //绘制子弹
@Override
public void release() {}  //释放资源
@Override
public boolean isCollide(GameObject obj) {  //判断碰撞事件
    // 位于物体左边
    if (object_x <= obj.getObject_x()
            && object_x + object_width <= obj.getObject_x()) {
        return false;
    }
    // 位于物体右边
    else if (obj.getObject_x() <= object_x
            && obj.getObject_x() + obj.getObject_width() <= object_x) {
        return false;
    }
    // 位于物体上边
    else if (object_y <= obj.getObject_y()
            && object_y + object_height <= obj.getObject_y()) {
        return false;
    }
    // 位于物体下边
```

```
    else if (obj.getObject_y() <= object_y
            && obj.getObject_y() + obj.getObject_height() <= object_y) {
        if (obj instanceof SmallPlane) {
            if (object_y - speed < obj.getObject_y()) {
                isAlive = false;
                return true;
            }
        } else
            return false;
    }
    isAlive = false;
    return true;
}
public int getHarm() { return harm;}  //获取伤害值
public void setHarm(int harm) { this.harm = harm;}  //设置伤害值
```

（1） 我方子弹

我方飞机有三种状态的子弹，分别为蓝色子弹（一级）、紫色子弹（二级）、红色子弹（三级），伤害值依次递增。这三种子弹的形态和发射方式均有不同，需要重写它的父类 Bullet 的成员函数，并另外重写从 GameObject 继承的属于自己的逻辑函数。

（2） 敌方子弹

由于敌方子弹种类增多，大型飞机和 boss 飞机均会发射子弹，且 boss 飞机有五种形态的子弹。为了方便统一管理，定义了 EnemyBullet 类，对敌方子弹相同特性的部分进行统一定义。

EnemyBullet 类：

```
public EnemyBullet(Resources resources) {
   super(resources);
   this.harm = 1;
}
// 初始化数据
@Override
public void initial(int arg0, float arg1, float arg2) {}
// 初始化图片资源
@Override
public void initBitmap() {}
// 绘图方法
@Override
public void drawSelf(Canvas canvas) {  }
// 释放资源的方法
@Override
public void release() {}
// 子弹运动逻辑，默认向下
@Override
```

```
public void logic() {
   if (object_y >= 0) {
      object_y -= speed;
   } else {
      isAlive = false;
   }
}
// 碰撞检测
@Override
public boolean isCollide(GameObject obj) {
   // 矩形 1 位于矩形 2 的左侧
   if (object_x <= obj.getObject_x()
         && object_x + object_width <= obj.getObject_x()) {
      return false;
   }
   // 矩形 1 位于矩形 2 的右侧
   else if (obj.getObject_x() <= object_x
         && obj.getObject_x() + obj.getObject_width() <= object_x) {
      return false;
   }
   // 矩形 1 位于矩形 2 的上方
   else if (object_y <= obj.getObject_y()
         && object_y + object_height <= obj.getObject_y()) {
      return false;
   }
   // 矩形 1 位于矩形 2 的下方
   else if (obj.getObject_y() <= object_y
         && obj.getObject_y() + obj.getObject_height() <= object_y) {
      return false;
   }
   isAlive = false;
   return true;
}
```

大型飞机子弹类和 boss 飞机的子弹类根据自身的特性重写父类的成员函数，其中由于 boss 飞机默认子弹是双发的，因此需要重写判断碰撞事件的函数，其他子弹类均可使用 EnemyBullet 类的 `isCollide(GameObject obj)`函数。

3.1.3 plane 飞机机体设计

Plane 飞机机体类主要包含了我方飞机和地方飞机，其中敌方飞机有四种型号，由一个统一的 EnemyPlane 类进行管理。

（1） 我方飞机

MyPlane 类中定义的成员变量和成员函数（包含重写的从 GameObject 基类继承的函数和自身新增的成员函数）：

```
private static final boolean Random = false;
private float middle_x;
private float middle_y;
private long startTime; // 开始时间
private long endTime; // 结束时间
private boolean isChangeBullet; // 更换子弹类型
private Bitmap mPlane;
private Bitmap mPlaneExplosion;
private List<Bullet> bullets; // 子弹列表
private MainView mainView;
private GameObjectFactory factory;
private boolean isInvincible; // 是否无敌
private boolean isDamaged; // 是否受损
private int bulletType;//当前子弹类型
private boolean isMissileBoom; //导弹是否被引爆
public MyPlane(Resources resources) { //加载飞机
    super(resources);
    initBitmap();
    this.speed = GameConstant.MYPLANE_SPEED;
    isInvincible = false;
    isChangeBullet = false;
    isDamaged = false;
    isMissileBoom = false;

    factory = new GameObjectFactory();
    bullets = new ArrayList<Bullet>();
    changeBullet(ConstantUtil.MYBULLET);
    bulletType = ConstantUtil.MYBULLET;
}
public void setMainView(MainView mainView) {  //主窗口绑定
    this.mainView = mainView;
}
@Override  //设置屏幕尺寸
public void setScreenWH(float screen_width, float screen_height) {
    super.setScreenWH(screen_width, screen_height);
    object_x = screen_width / 2 - object_width / 2;
    object_y = screen_height - object_height;
    middle_x = object_x + object_width / 2;
    middle_y = object_y + object_height / 2;
}
//加载图片资源
```

```
@Override
public void initBitmap() {
    mPlane = BitmapFactory.decodeResource(resources,
R.drawable.myplane);
    mPlaneExplosion = BitmapFactory.decodeResource(resources,
            R.drawable.myplaneexplosion);

    object_width = mPlane.getWidth() / 3;
    object_height = mPlane.getHeight();
}
//绘图
@Override
public void drawSelf(Canvas canvas) {
    if (isDamaged) {
        drawExplosion(canvas);
    } else {
        drawPlane(canvas);
    }
}
//绘制机体
private void drawPlane(Canvas canvas) {
    int x = (int) (currentFrame * object_width);
    canvas.save();
    canvas.clipRect(object_x, object_y, object_x + object_width,
            object_y + object_height);
    canvas.drawBitmap(mPlane, object_x - x, object_y, paint);
    canvas.restore();
    if (isInvincible) {
        currentFrame++;
        if (currentFrame >= 3) {
            currentFrame = 0;
        }
    } else if (isAlive) {
        if (bulletType == ConstantUtil.MYBULLET) {
            currentFrame = 0;
        } else if (bulletType == ConstantUtil.MYBULLET1) {
            currentFrame = 1;
        } else if (bulletType == ConstantUtil.MYBULLET2) {
            currentFrame = 2;
        }
    }
}
//绘制爆炸机体
private void drawExplosion(Canvas canvas) {
```

```
    int x = (int) (currentFrame * object_width);
    canvas.save();
    canvas.clipRect(object_x, object_y, object_x + object_width,
            object_y + object_height);
    canvas.drawBitmap(mPlaneExplosion, object_x - x, object_y, paint);
    canvas.restore();
    if (bulletType == ConstantUtil.MYBULLET) {
        currentFrame++;
        if (currentFrame >= 2) {
            currentFrame = 0;
        }
    } else if (bulletType == ConstantUtil.MYBULLET1) {
        currentFrame++;
        if (currentFrame >= 4) {
            currentFrame = 2;
        }
    } else if (bulletType == ConstantUtil.MYBULLET2) {
        currentFrame++;
        if (currentFrame >= 6) {
            currentFrame = 4;
        }
    }
}
//释放内存
@Override
public void release() {
    for (Bullet obj : bullets) {
        obj.release();
    }
    if (!mPlane.isRecycled()) {
        mPlane.recycle();
    }
    if (!mPlaneExplosion.isRecycled()) {
        mPlaneExplosion.recycle();
    }
}
//射击动作的逻辑
@Override
public void shoot(Canvas canvas, List<EnemyPlane> planes) {
    for (Bullet bullet : bullets) {
        if (bullet.isAlive()) {
            // 绘制子弹
            bullet.drawSelf(canvas);
            // 检测子弹是否击中敌机
```

```
            checkAttacked(planes, bullet);
        }
    }
}
//判断子弹是否击中敌机
private void checkAttacked(List<EnemyPlane> planes, Bullet bullet) {
    for (EnemyPlane enemyPlane : planes) {
        boolean isCollide = enemyPlane.isCanCollide() &&
bullet.isCollide(enemyPlane);
        if (isCollide) {
            attackedEnemyPlane(bullet, enemyPlane);
            break;
        }
    }
}
//子弹击中敌机的处理
private void attackedEnemyPlane(Bullet bullet, EnemyPlane plane) {
    // 记录敌机的受损状态
    plane.attacked(bullet.getHarm());
    if (plane.isExplosion()) {
        // 根据击毁的不同敌机增加相应的分数(同时播放爆炸音乐)
        mainView.addGameScore(plane.getScore());
        if (plane instanceof SmallPlane) {
            mainView.playSound(2);
        } else if (plane instanceof MiddlePlane) {
            mainView.playSound(3);
        } else if (plane instanceof BigPlane) {
            mainView.playSound(4);
        } else {
            mainView.playSound(5);
        }
    }
}
初始化子弹
@Override
public void initBullet() {
    for (Bullet obj : bullets) {
        if (!obj.isAlive()) {
            obj.initial(0, middle_x, middle_y);
            break;
        }
    }
}
//更换子弹类型
```

```
@Override
public void changeBullet(int type) {
    bulletType = type;
    bullets.clear();
    if (isChangeBullet) {
        if (type == ConstantUtil.MYBULLET1) {
            for (int i = 0; i < 6; i++) {
                MyPurpleBullet bullet1 = (MyPurpleBullet) factory
                        .createMyPurpleBullet(resources);
                bullets.add(bullet1);
            }
        } else if (type == ConstantUtil.MYBULLET2) {
            for (int i = 0; i < 4; i++) {
                MyRedBullet bullet2 = (MyRedBullet) factory
                        .createMyRedBullet(resources);
                bullets.add(bullet2);
            }
        }
    } else {
        for (int i = 0; i < 10; i++) {
            MyBlueBullet bullet = (MyBlueBullet)
factory.createMyBlueBullet(resources);
            bullets.add(bullet);
        }
    }
}
//判断特殊子弹是否失效
public void isBulletOverTime() {
    if (isChangeBullet) {
        endTime = System.currentTimeMillis();
        if (endTime - startTime > GameConstant.MYSPECIALBULLET_DURATION)
{
            isChangeBullet = false;
            startTime = 0;
            endTime = 0;
            changeBullet(ConstantUtil.MYBULLET);
        }
    }
}
//设置飞机无敌状态时间
public void setInvincibleTime(long time) {
    if (DebugConstant.INVINCIBLE) {
        isInvincible = true;
        SystemClock.sleep(time);
```

```
            isInvincible = false;
        }
    }
    //判断是否无敌状态中
    public boolean isInvincible() {
        return isInvincible;
    }
    //设置导弹状态
    public void setMissileState(boolean isBoom) {
        isMissileBoom = isBoom;
    }
    //检测导弹状态
    public boolean getMissileState() {
        return isMissileBoom;
    }
    //设置机体是否受损
    public void setDamaged(boolean arg) {
        isDamaged = arg;
    }
    //得到机身状态
    public boolean getDamaged() {
        return isDamaged;
    }
    //设置起始时间
    public void setStartTime(long startTime) {
        this.startTime = startTime;
    }
    //判断子弹类型是否改变
    @Override
    public boolean isChangeBullet() {
        return isChangeBullet;
    }
    @Override
    public void setChangeBullet(boolean isChangeBullet) {
        this.isChangeBullet = isChangeBullet;
    }
    //其他参量传递
    @Override
    public float getMiddle_x() {
        return middle_x;
    }
    @Override
    public void setMiddle_x(float middle_x) {
        this.middle_x = middle_x;
```

```
    this.object_x = middle_x - object_width / 2;
}
@Override
public float getMiddle_y() {
    return middle_y;
}
@Override
public void setMiddle_y(float middle_y) {
    this.middle_y = middle_y;
    this.object_y = middle_y - object_height / 2;
}
```

（2） 敌方飞机

由于敌方飞机有四种不同的种类，为了便于管理。我们定义了一个新的 EnemyPlane，用于描述所有敌方飞机共同拥有的功能。

EnemyPlane 类的成员变量和成员函数：

```
protected int score; //分数
protected int blood;  //当前血量
protected int bloodVolume; //总血量
protected boolean isExplosion; //爆炸状态
protected boolean isVisible;  //可见状态
public int speedTime; //游戏速度的倍数

public EnemyPlane(Resources resources) {
    super(resources);
    initBitmap(); // 初始化图片
}
// 初始化
@Override
public void initial(int arg0, float arg1, float arg2) {
    speedTime = arg0;
}
// 初始化图片
@Override
public void initBitmap() {}
//绘制
@Override
public void drawSelf(Canvas canvas) { /* 由子类实现 */ }
//释放资源
@Override
public void release() {}
//逻辑设计
@Override
```

```java
public void logic() {
    if (object_y < screen_height) {
        object_y += speed;
    } else {
        isAlive = false;
    }
    if (object_y + object_height > 0) {
        isVisible = true;
    } else {
        isVisible = false;
    }
}
//被攻击时的逻辑
public void attacked(int harm) {
    blood -= harm;
    if (blood <= 0) {
        isExplosion = true;
    }
}
//判断是否发生碰撞
@Override
public boolean isCollide(GameObject obj) {
    // 位于物体左边
    if (object_x <= obj.getObject_x()
            && object_x + object_width <= obj.getObject_x()) {
        return false;
    }
    // 位于物体右边
    else if (obj.getObject_x() <= object_x
            && obj.getObject_x() + obj.getObject_width() <= object_x) {
        return false;
    }
    // 位于物体上方
    else if (object_y <= obj.getObject_y()
            && object_y + object_height <= obj.getObject_y()) {
        return false;
    }
    // 位于物体下方
    else if (obj.getObject_y() <= object_y
            && obj.getObject_y() + obj.getObject_height() <= object_y) {
        return false;
    }
    // 若不满足上述条件,则判断为相碰撞
    return true;
```

```
}
//是否能碰撞判断
public boolean isCanCollide() {
    return isAlive && !isExplosion && isVisible;
}
//其他参数传递
public int getScore() { return score;}
public void setScore(int score) { this.score = score;}
public int getBlood() { return blood;}
public void setBlood(int blood) { this.blood = blood;}
public int getBloodVolume() { return bloodVolume;}
public void setBloodVolume(int bloodVolume) { this.bloodVolume =
bloodVolume;}
public boolean isExplosion() { return isExplosion;}
```

在此基础上，设计了其他四种不同形态的飞机类。小型机和中型机除了重写父类的成员函数以外，不需要添加其他自身的函数。但是大型机和 boss 飞机的设计就相对比较复杂，还需要增加射击子弹的相关函数。而 boss 机还需要区分不同损毁状态时的射击事件。

3.2 view 视图设计

初始界面排版逻辑

```
public void drawSelf() {
    try {
        canvas = sfh.lockCanvas();
        canvas.drawColor(Color.BLACK);
        canvas.save();
        canvas.scale(scalex, scaley, 0, 0);
        canvas.drawBitmap(background, 0, 0, paint);
        canvas.restore();
        canvas.drawBitmap(text, text_x, text_y, paint);
        if (isBtChange) {
            canvas.drawBitmap(button2, button_x, button_y, paint);
        } else {
            canvas.drawBitmap(button, button_x, button_y, paint);
        }
        if (isBtChange2) {
            canvas.drawBitmap(button2, button_x, button_y2, paint);
        } else {
            canvas.drawBitmap(button, button_x, button_y2, paint);
        }
```

```
        //开始游戏的按钮
        paint.setFakeBoldText(true);
        paint.setColor(Color.BLACK);
        canvas.drawText(startGame, screen_width / 2 - strwid / 2, button_y
                + button.getHeight() / 2 + strhei / 2, paint);
        //退出游戏的按钮
        canvas.drawText(exitGame, screen_width / 2 - strwid / 2, button_y2
                + button.getHeight() / 2 + strhei / 2, paint);

        //版本信息的内容
        paint.setColor(Color.rgb(255, 255, 255));
        canvas.drawText(version, screen_width / 2 - version_width / 2,
                button_y / 2 + text_y / 2 + text.getHeight() / 2, paint);

        canvas.restore();
    } catch (Exception err) {
        err.printStackTrace();
    } finally {
        if (canvas != null)
            sfh.unlockCanvasAndPost(canvas);
    }
}
```

游戏界面背景滚动逻辑

```
  public void viewLogic() {
    if (bg_y > bg_y2) {
       bg_y += 10;
       bg_y2 = bg_y - background.getHeight();
    } else {
       bg_y2 += 10;
       bg_y = bg_y2 - background.getHeight();
    }
    if (bg_y >= background.getHeight()) {
       bg_y = bg_y2 - background.getHeight();
    } else if (bg_y2 >= background.getHeight()) {
       bg_y2 = bg_y - background.getHeight();
    }
 }
```

结束界面与开始界面相似，这里就不过多赘述了。

3.3 sound 音效及背景音乐

音效播放相关结构

```
public GameSoundPool(MainActivity mainActivity) {
        this.mainActivity = mainActivity;
        map = new HashMap<Integer, Integer>();
        soundPool = new SoundPool(8, AudioManager.STREAM_MUSIC, 0);
        //初始化对象，第一个参数是允许有多少个声音流同时播放,第 2 个参数是声音
类型,第三个参数是声音的品质
    }
    public void initGameSound() {
        map.put(1, soundPool.load(mainActivity, R.raw.shoot, 1));
        map.put(2, soundPool.load(mainActivity, R.raw.explosion, 1));
        map.put(3, soundPool.load(mainActivity, R.raw.explosion2, 1));
        map.put(4, soundPool.load(mainActivity, R.raw.explosion3, 1));
        map.put(5, soundPool.load(mainActivity, R.raw.bigexplosion, 1));
        map.put(6, soundPool.load(mainActivity, R.raw.get_goods, 1));
        map.put(7, soundPool.load(mainActivity, R.raw.button, 1));
    }
    public void playSound(int sound, int loop) {
        AudioManager am = (AudioManager)
mainActivity.getSystemService(Context.AUDIO_SERVICE);
        float stramVolumeCurrent =
am.getStreamVolume(AudioManager.STREAM_MUSIC);
        float stramMaxVolumeCurrent =
am.getStreamVolume(AudioManager.STREAM_MUSIC);
        soundPool.play(map.get(sound), 0.3f, 0.3f, 1, loop, 1f);
    }
```

背景音乐播放相关结构

```
   // 背景音乐
   mMediaPlayer = MediaPlayer.create(mainActivity, R.raw.game2);
   mMediaPlayer.setLooping(true);
   if (!mMediaPlayer.isPlaying()) {
      mMediaPlayer.start();
   }
   //此页面任务结束后
    mMediaPlayer.stop();
```

四．开发困难及解决方案

4.1 界面切换的问题

这个问题在我们的期中作业中就反复的遇到过，虽然这次比上次有经验，但是还是想记录下来。具体问题就是在按钮点击选择了以后，要进行界面跳转的时候，程序会出现闪退现象。

解决方法：这个问题是线程的问题，在各个界面的代码中要注意是否有创建新线程的问题，如果没有则需要在相应 view 界面加上“`thread = new Thread(this);`”

4.2 一直做不到背景音乐随界面切换的问题

在设计时，一开始我想采用一个背景音乐贯穿整个游戏的做法，但是后面感觉这样子过于单调，而且准备界面安安静静的风格和后面炮火纷飞的激烈风格差异很大，我认为战斗的主页面和准备界面应该使用两个风格不同的音乐，但是很难过的是对于音乐切换的地方我加了很多代码都一直没有做到播放出第二个音乐。

解决方法：把背景音乐 1 和背景音乐 2 从 GameSoundPool 中拿出来，不用那个方法来播放，采用 MediaPlayer 来播放。在准备界面中 ReadyView 初始化中直接开始播放背景音乐 1，但是不要忘记在 onTouchEvent 里设置触摸到按钮“开始游戏”的时候的 mMediaPlayer.stop()。然后在 MainView 的初始化里也直接设置背景音乐 2 的播放。

4.3 复杂结构与类的管理

在设计时，我们发现飞机大战游戏的逻辑比较简单和统一，但是相关结构和类的封装比较复杂，如何更加清晰地设计整体的框架，让我们的设计能够体现不一样的变化是我们的重要课题。一开始我们采用的是一个模型对应一个对象的方法，但是随着设计的模型数量增多，我们发现整个工程中涉及到的各种对象模型都混杂在了一起，显得比较混乱，也不方便在视图中进行调用，我们的工程也陷入了瓶颈。

解决方法：为了解决这个难题，后来我们对前面定义的各种对象进行了分类整理，并在同一类别的对象中提取公共部分，整理构建了一个父类，进行分类化的管理。这样一来，在视图中的模块封装性就增强了。也方便我们在后续的工作中调整参量，进行功能测试。

五．分工

姓名	工作
许多	界面制作、物体素材制作、飞机受伤及爆炸效果制作、音乐音效设置、报告撰写、视频后期制作

吕慕凡	游戏整体参数设置、机体、子弹及掉落物品运动逻辑等、报告撰写、视频录制

六．总结

这次的实验，我们尽自己最大的努力，把一款简单的飞机大战游戏尽量的丰富化。我们在这个程序的完成过程中学到了很多东西。

首先是从期中的那次作业中学到的东西在这次的作业里也派上了用场。一个是有关界面切换的问题，在期中的作业中我们就遇到过因为界面切换没有注意到线程关系而卡在一个地方很久的情况，而这次的 APP 制作过程中我们明显感觉到自己有了上一次的经验以后在这个地方的进步，虽然也在界面切换的地方卡了一小下，但是很快就排查出了原因，在合适的地方加上线程以后界面切换很快就不成问题了。

另一个从期中收获过来用到期末的地方是触屏的控制。在期中作业里，我们实体标注的方法是通过触屏选中文字再进行操作，在那一次的操作中我学到了很多跟触屏这类交互有关的东西，而这一次的游戏里，我们选择把飞机的每一个动作还有我们的每一个选择都放在触屏交互中完成，相比那种上下左右按键控制的飞机游戏多了灵活性。

而相比期中的作业，我们的期末选题为一个游戏还学到了很多期中没有学到的东西。老师在上课的时候就提到，我们学技术的同学在做项目的时候最好和学艺术的同学组队来避免做出非常难看的应用，在这次的 APP 制作中我们也非常明显的感受到音乐和图画对一个应用带来的影响。

在定题的时候我们也想过是否要定一个类似于文字处理、排版、工具或者将网页做成手机客户端的那种跟期中很像的题目，但是最后我们还是想选择一个趣味性更大的题目来做，而我们最终也的确收获了很多更有趣的知识。例如在这个程序中，我们基本上是要使用外来图片进行操作，背景音效的添加、背景图片的设置还有各种各样的图片让我感受到了游戏比文本编辑多出来的吸引力。而且不仅是在 Android studio 里设置图片的各类活动让我收获了很多，为了让飞机有多种同一形态不同颜色的设定，能做出来爆炸动画和受伤闪烁动画，以及处理所有图片让它们没有底色，我在 Photoshop 上也学到了很多图片的操作，还初步体验了一下用渐变的图片在连续显示来制作小动画的感觉，可谓在审美和艺术层面也收获良多。