软件设计综合实验

Software Design Projects

Plane War 游戏设计文档

13331352 张子轩

13331305 薛　昭

13331216 邵嘉怡

13331006 蔡　艳

13331133 李展文

13331240 陶久成

目录

1 引言

1.1 编写目的

本文档的编写目的在于让读者对Plane War这款游戏的设计有全局性、总体方面的了解，根据需求提炼出具体的设计方案。

1.2 阅读对象

此文档将适合以下人员阅读：

- 本项目组成员
- 对本游戏感兴趣的开发人员

2 游戏介绍

这是一款射击类游戏，整体环境主要还是围绕太空为主，高保真的音效，为玩家呈现一场不一样射击体验。玩家在游戏中要做的就是驾驶着最新战机，在敌机身前发动攻击。在击毁敌机的同时获得分数，击毁的敌机越多，则相对的获得分数就越高。在击毁敌机是，会随机出现漂浮道具，玩家控制飞机捡取漂浮道具进而强化飞机。玩家进行游戏的时候需要注意不能被敌机及敌机子弹碰到，否则玩家控制角色死亡，同时游戏结束，记录玩家获取的积分。

3 技术选型

3.1 实现语言：C++

C++支持过程化程序设计、面向对象程序设计等多种程序设计风格，同时可以跟C语言进行很好地结合。通过C++引入的面向对象也使得进行交互的开发更为简单。

C++避免了平台限定或没有普遍用途的特性，同时C++还设计成无需复杂的程序设计环境。

3.2 开发框架：Cocos2D-X

Cocos2d-X是一个基于MIT协议的开源框架，用于构建游戏、应用程序和其他图形界面交互应用。相比于其他的游戏开发引擎，Cocos2D功能更加多样，更便于上手，强大而且灵活；提供成熟的框架和多种工具；开源、免费，社区支持强大。

Cocos2d的流程控制非常方便，可以管理不同场景间的流程；精灵和动作的管理也非常方便。

4 架构设计

由项目的需求分析可以清晰的对本游戏的具体功能实现进行设计，如下图是本游戏的总体架构设计：

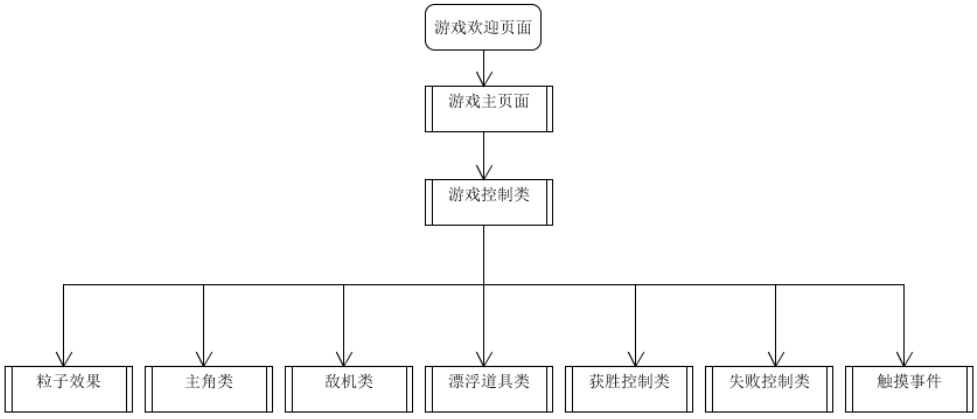

图 4-1Plane War总架构图

通过对游戏的需求进行分析和细致的归纳，可以认为游戏的主要内容是由游戏进行时和游戏失败两个主要部分所构成。游戏进行时包含了用户对主角的操作以及对主角信息（获得的分数及获取的道具）的管理，用户游戏结束的操作选择，游戏进行为本系统的设计核心。

基于这些考虑，本游戏将对游戏进行时的事件作为一个重点的功能模块进行详细设计。

5 模块划分

5.1 游戏欢迎页面

5.1.1 简要说明

游戏欢迎页面主要是预加载游戏进行需要的图片、音乐等资源进行预加载，以及游戏操作的介绍。

5.1.2 事件流

1) 基本事件流

(a) 本模块开始于玩家进入游戏欢迎页面。

(b) 系统显示游戏指南，并自动加载背景及音乐；

(c) 玩家点击屏幕上的“开始游戏”按钮；

A1：玩家选择关闭游戏

2) 备选事件流

A1：玩家选择关闭游戏

✧ 游戏停止运行。

5.1.3 特殊需求

无

5.1.4 前置条件

玩家运行游戏客户端并进入欢迎页面。

5.1.5 后置条件

玩家进入游戏主页面。

5.1.6 游戏欢迎页面活动图

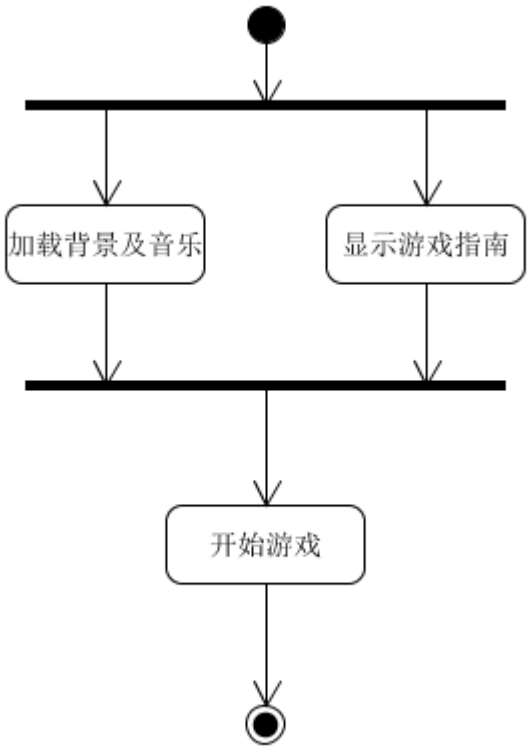

图 5-1游戏欢迎页面活动图

5.2 游戏主页面

5.2.1 简要说明

游戏主页面主要是进行交互，接受玩家控制从而操作飞机做出动作，进行各种条件的判断从而控制各种场景的流程。

5.2.2 事件流

1) 基本事件流

(a) 系统加载玩家数据；

(b) 系统生成敌机；

(c) 玩家控制飞机运动、攻击敌机；

(d) 判断玩家、敌机是否被攻击；

(e) 判断游戏是否结束；

2) 备选事件流

无

5.2.3 特殊需求

无

5.2.4 前置条件

玩家进入游戏主页面

5.2.5 后置条件

进入游戏结束页面。

5.2.6 游戏主页面活动图

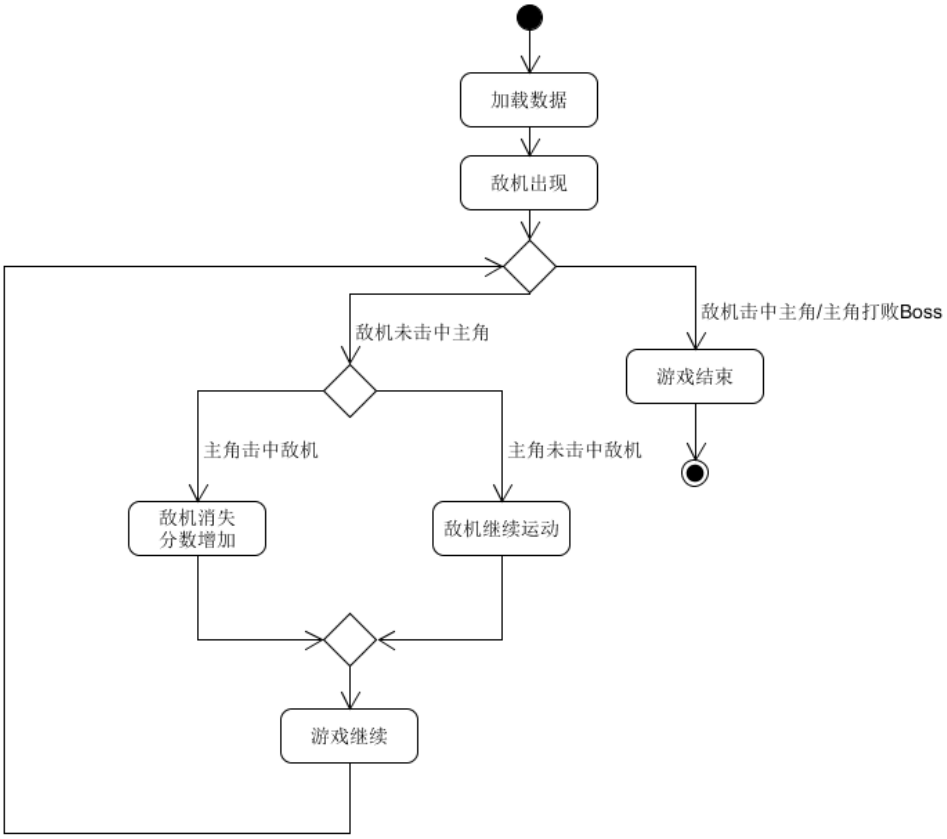

图 5-2游戏主页面活动图

5.2.7 漂浮道具活动图

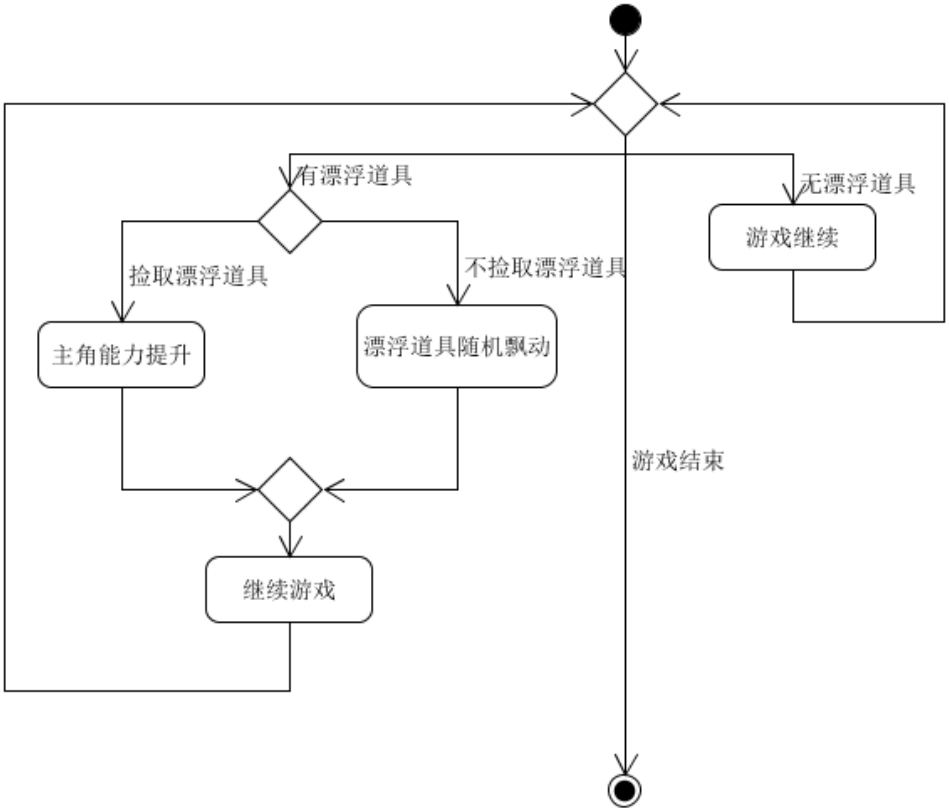

图 5-3漂浮道具活动图

5.3 游戏结束界面

5.3.1 简要说明

游戏结果界面主要是给出玩家的游戏结果，包括游戏结束、通关状态等，供玩家选择是否退出游戏。

5.3.2 事件流

1) 基本事件流

(a) 加载结束页面；

(b) 玩家选择退出游戏

A1：玩家选择重新游戏

(c) 退出游戏

2) 备选事件流

A1：玩家选择重新游戏

✧ 进入游戏欢迎页面

5.3.3 特殊需求

无

5.3.4 前置条件

玩家在游戏中，并且达成游戏结束条件。

5.3.5 后置条件

退出游戏或重新进入游戏欢迎页面。

5.3.6 游戏主页面活动图

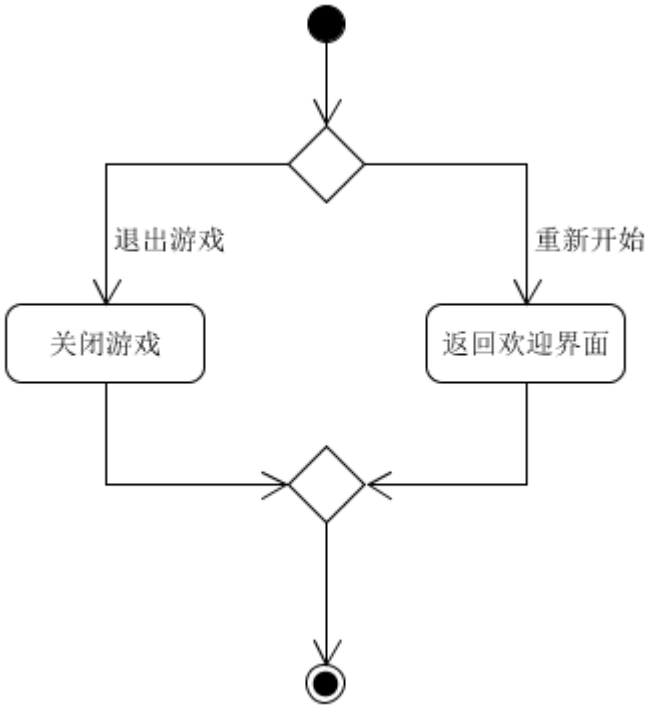

图 5-4游戏结束界面活动图

6 游戏实现及其技术

6.1 游戏总体实现

游戏的模块结构是对游戏的进行一个总体划分，要真正的实现游戏，还需要进一步的设计用户的功能。

游戏的功能大致有：游戏欢迎页面、游戏主页面、游戏控制类、主角类、敌机类、漂浮道具类、获胜控制类、失败控制类和触摸事件，针对每一个功能都实现了不同的作用。

6.2 面向对象的开发方法

总体来说游戏中使用最突出的技术就是面向对象的编程方法，我们把敌机、玩家、子弹等精灵抽象成类，针对不同的对象，我们设计了不同的属性和操作。下面的6.7和6.8两部分主要使用了这种技术。

6.3 继承 Cosos2D-X 的结构关系

Cocos2D程序由一系列场景组成，每一个场景又由不同的层组成，层上有不同的精灵，这些精灵有不同的属性和动作，导演负责场景的切换以保证整个流程的推进。程序的执行过程就是操纵每个层上的精灵或者菜单选项，使得整个程序在不同的场景中切换。

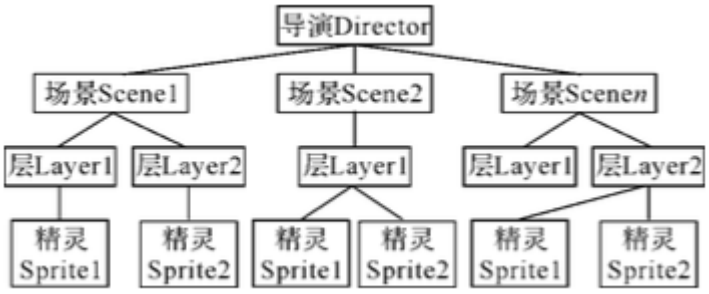

图 6-1Cocos2D基本元素结构

对应于我们的游戏，我们设计了三个场景，分别是欢迎、游戏和结束。

欢迎场景和结束场景主要各对应一个层，层上放置着label和button，以实现跳转功能。对应代码如下：

```
Scene* Win::createScene()
{
    // 'scene' is an autorelease object
    auto scene = Scene::create();

    // 'layer' is an autorelease object
    auto layer = Win::create();

    // add layer as a child to scene
    scene->addChild(layer);

    // return the scene
    return scene;
}
```

图 6-2胜利页面的场景

游戏场景有一个层，层上放置着包括玩家、敌机、子弹、陨石等等各种精灵。这些元素均由我们的导演控制。

```
Scene* game::createScene()
{
    auto scene = Scene::createWithPhysics();
    //scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
    scene->getPhysicsWorld()->setGravity(Vec2::ZERO);
    auto layer = game::create(scene->getPhysicsWorld());
    scene->addChild(layer);
    // return the scene
    return scene;
}
```

图 6-3游戏主页面的场景和层

```
    //xz
    auto Aenemy = myEnemy->createNewEnemy(Score);
    this->addChild(Aenemy,1);
}
```

图 6-4游戏主页面的层和精灵

6.4 游戏欢迎页面实现及相关技术

用户运行可执行文件后，首先看到的就是游戏的欢迎页面。

欢迎页是为了对资源，即图片资源和音乐资源进行预加载，图片资源和音乐资源相对较大，进入游戏后在加载比较慢，对游戏体验会造成一定影响，所以在欢迎页面进行预加载，进入游戏后可以享受较好的体验，不会因为加载资源而造成游戏卡的问题。

在代码中对应的部分如下图：

```
Sprite *bg = Sprite::create("background.jpg");
float scale_x = visibleSize.width / bg->getContentSize().width;
float scale_y = visibleSize.height / bg->getContentSize().height;
bg->setScaleX(scale_x);
bg->setScaleY(scale_y);
bg->setPosition(visibleSize.width / 2, visibleSize.height / 2);
this->addChild(bg, 0);

auto begin_label = Label::createWithSystemFont("开始游戏", "Microsoft Yahei", 30);
auto beginItem = MenuItemLabel::create(begin_label, CC_CALLBACK_1(HelloWorld::beign_game, this));
beginItem->setPosition(Vec2(visibleSize.width / 2, visibleSize.height / 2 + beginItem->getContentSize().height * 2));
```

图 6-5游戏欢迎页面代码（部分）

6.5 主场景的实现及相关技术

游戏的主场景主要是为之后所有的精灵生成一个层，所有精

灵均在此层上进行动作。

在代码中对应的部分如下图：

```
Scene* game::createScene() { ... }

game* game::create(PhysicsWorld* world) { ... }

bool game::init(PhysicsWorld* world)
{
```

图 6-6游戏主页面代码（部分）

6.6 主角类的实现及相关技术

游戏的主角类主要是生成主角精灵，接收来自管理类的控制信息从来实现主角的动作。

在代码中对应的部分如下图：

```
class PlaneLayer : public cocos2d::Ref
{
public:
    static PlaneLayer* getInstance();
    void createPlane(Speed* &speed_);
    Sprite* getPlane();
    float getAngle();
    int getPower();
    void getHurt();
    void getCure();

    void movePlane(float touchX,float touchY,Speed* &speed);

    void speedup_begin();
    void speedup_end();

    void invin_begin();
```

图 6-7主角类代码（部分）

其中，主角类的实现我们主要使用了面向对象的思想，把主角精灵单独抽象出来。

6.7 敌机类、道具类的实现及相关技术

游戏的此类主要是生成敌机、子弹、漂浮道具精灵，在场景的随机位置生成随机的敌机，并控制敌机向主角攻击。此外子弹类还进行碰撞检测，判断子弹是否攻击到敌机，从而实现积分的增加。

在代码中对应的部分如下图：

```
class Enemy:public cocos2d::Ref
{
public:
    static Enemy* getInstance();
    Sprite* createNewEnemy(int score);
    Sprite* createBoss();
    void enmey_Moving(float planex,float planey);
    void deleteEnemy(Sprite* e);
    int getEnemySize();
    Sprite* getAEnemy(int i);
private:
    Enemy();
    Vector<Sprite*> enemys;
    static Enemy* enemy;
};
```

图 6-8敌机类代码（部分）

```
USING_NS_CC;
class FloatItem : public cocos2d::Ref
{
public:
    //获取单例工厂
    static FloatItem* getInstance();
    //生成一个漂浮道具，并存储到容器中管理
    Sprite* createItem(int type);
    //让容器中的道具随机移动
    void moveItem();
    //移除道具
    void removeItem();
    //移除特定道具
    void remove_contactItem(Sprite* sp);
    //获取道具数量
    int getSize();
    //判断道具是否被拾到 0 没有 1,2,3,对应type的1,2,3,
    int is_getbuff(float planeX, float planeY,float R);
```

图 6-9漂浮道具类代码（部分）

其中，敌机类和子弹类的实现我们主要使用了面向对象的思想，把敌机、子弹、漂浮道具精灵抽象出来作为一个类，使用时只需实例化即可。

6.8 胜利、失败控制类的实现及相关技术

这两个类主要是用以判断实例和失败的条件，在满足条件是跳转至胜利和失败的页面。

主要对应的代码如下：

```
bg->setPosition(visibleSize.width / 2, visibleSize.height / 2);
this->addChild(bg, 0);

auto label = Label::createWithSystemFont("游戏获胜", "Microsoft Yahei", 30);

// position the label on the center of the screen
label->setPosition(Vec2(visibleSize.width / 2, visibleSize.height / 2 + label->getContentSize().height));

// add the label as a child to this layer
this->addChild(label, 1);

auto label_restart = Label::createWithSystemFont("重新开始", "Microsoft Yahei", 30);
auto restartItem = MenuItemLabel::create(label_restart, CC_CALLBACK_1(Win::restart, this));
restartItem->setPosition(Vec2(visibleSize.width / 2, visibleSize.height / 2));
```

图 6-10胜利控制类代码（部分）

```
auto label = Label::createWithSystemFont("游戏失败", "Microsoft Yahei", 30);
label->setPosition(Vec2(visibleSize.width / 2, visibleSize.height / 2 + label->getContentSize().height));
this->addChild(label, 1);

auto label_restart = Label::createWithSystemFont("重新开始", "Microsoft Yahei", 30);
auto restartItem = MenuItemLabel::create(label_restart, CC_CALLBACK_1(Fail::restart, this));
restartItem->setPosition(Vec2(visibleSize.width / 2, visibleSize.height / 2));

auto menu = Menu::create(restartItem, NULL);
menu->setPosition(Vec2::ZERO);
this->addChild(menu, 1);

auto exit_label = Label::createWithSystemFont("退出游戏", "Microsoft Yahei", 30);
```

图 6-11失败控制类代码（部分）