软件设计文档

项目名称:Hit-airplane

我们最后小组讨论的选择方案为方案1

软件的技术选型理由：

|  |  |  |  |
| --- | --- | --- | --- |
| 项目 名称 | 方案1 | 方案2 | 方案3 |
| 技术类型 | C++ | Java | Html |
| 编程强度 | 较强 | 强 | 中 |
| 实现周期 | 两到三周 | 三到四周 | 两到三周 |
| 移动开发 | 支持 | 支持 | 不易支持 |
| 后期维护 | 一般 | 较难 | 难 |
| 界面优化 | 一般 | 一般 | 较好 |
| 流畅程度 | 较好 | 一般 | 较差 |

架构设计：

逻辑架构：

WelcomeScene 表示加载场景，把一些图片、声音资源和一些动画提前做好预先加载到缓存里；

GameScene 表示游戏主场景，最重要功能的是进行碰撞检测，游戏大部分时间在这里度过；

GameOverScene表示游戏结束场景，游戏结束后显示历史最高分、当次游戏获得分数等；

Enemy类主要是产生不同类型的敌机，包括敌机的血量、打爆该敌机得到的分数等；

EnemyLayer类提供一些函数负责处理敌机产生后的飞行动画、爆炸动画、敌机的回收等，它有一个array数组来存储产生的敌机。其实两个类的提供的功能区分得不是很明显，也可以集合到一个类处理。

MutiBulletLayer类负责产生子弹、回收子弹等关于子弹的相关处理，它有一个array数组来存储子弹；

PlaneLayer类负责处理一些飞机的功能，包括移动飞机，飞机爆炸的回调函数等；因为子弹的发射位置由飞机的位置决定，所以MutiBulletLayer是PlaneLayer的一个成员；

接口设计：基于cocos2dx游戏引擎的设计思想，系统内部的模块分为：处理视图的Scene，Layer，Sprite；处理动作逻辑的Action。模块之间通过函数调用的方式进行通信。

技术架构：遵循cocos2dx游戏引擎最佳实践，基于引用计数的方式，创建对象时将对象加入自动释放池，实现内存的精简管理。具体编码通过宏#define CREATE\_FUNC(\_\_TYPE\_\_)实现。

非功能性设计：

性能：使用图片缓存策略，将帧动画提前加入缓存：SpriteFrameCache

可维护：高内聚低耦合的编码思想，将逻辑拆分至不同的类，提高可维护性；

可移植性：可移植性完全交由cocos2dx游戏引擎负责，实现跨平台。

模块划分：

首先，定义一下各个标准库函数，建立一个结构体数组，定义各个功能所需的函数等，然后设置主函数，程序采用模块化设计，主函数程序的入口，各模块独立，可分块调试，均由主函数控制调用。

软件设计技术：

|  |  |  |
| --- | --- | --- |
| 具体设计技术 | 源代码位置 | 对应模块和代码 |
| Structure programming | 独立模块结构，减少模块的相互联系使模块可作为插件或积木使用，降低程序的复杂性，提高可靠性。 | 例如PlaneLayer类负责处理一些飞机的功能，包括移动飞机，飞机爆炸的回调函数等；因为子弹的发射位置由飞机的位置决定，所以MutiBulletLayer是PlaneLayer的一个成员； |
| Object-Oriented Programming | 所有.h文件中的class都是面向对象的 | class AppDelegate（）  class BulletLayer（）  class ControlLayer（）  class Enemy（）  class EnemyLayer（）  class GameLayer（）  class GameScene（） |
| Aspect-Oriented Programming | 面向切面编程概念 |  |
| Services-Oriented Architecture | 面向服务架构 |  |
| Design Pattern | 单例模式singleton：导演类（director类）：AppDelegate.cpp | bool AppDelegate::applicationDidFinishLaunching() |
| 观察者模式：所有callback函数：ControlLayer.cpp | void ControlLayer::menuPauseCallback(Ref\* pSender)（） |