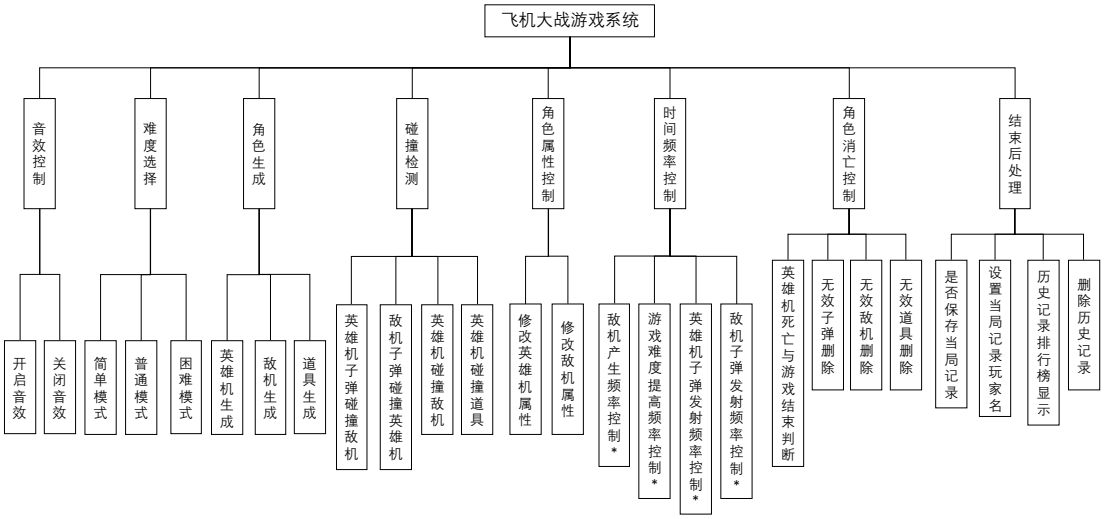
1 实验环境

本次实验所用到的操作系统为微软公司于 2021 年发布的 Windows 11，主要开发工具为 JetBrains 公司旗下一款著名的 Java 语言集成开发环境 IntelliJ IDEA，版本号为 IntelliJ IDEA 2021.3.3 (Ultimate Edition)。

2 实验过程

2.1 系统功能分析

可根据本次实验设计出的飞机大战游戏系统，绘制出系统的功能层次图，如下图所示（亮点功能用*标注）。



不难看出，所设计出的飞机大战游戏系统主要具备音效控制、难度选择、角色生成、碰撞检测、角色属性控制、时间频率控制、角色消亡控制、结束后处理等 8 大类功能。

（1） 音效控制

在游戏的开始面板可以对是否开启音效进行选择。如果选择关闭音效，游戏全程都不会播放任何声音；如果选择开启音效，游戏全程都会循环播放游戏背景音乐，直至英雄机坠毁、游戏结束时停止播放。

除此之外，游戏中所涉及到的音效还有：英雄机子弹击中敌机音效、炸弹道具爆炸音效、加血道具音效、游戏结束音效。当 Boss 敌机出场时，也会循环播放其背景音乐，直至游戏结束或 Boss 敌机坠毁时停止播放。

（2） 难度选择

在游戏的开始面板还可以对游戏难度进行选择。该系统一共设置了 3 种游戏难度，分别是简单模式、普通模式和困难模式。当点击某一游戏难度对应的按钮

时，该模式下难度的游戏开始。

可以通过**模板模式**实现不同难度游戏的设定。它们之间的区别主要如下表格所示。（注：表格中因素的 1 次改变即为 1 次游戏难度提升，若未说明某因素变化的最大程度，如“最低不低于”、“最高不高于”等，则说明该因素将持续改变直至游戏结束）

	简单模式	普通模式	困难模式
是否会生成 Boss 敌机	否	是	是
每次召唤是否提高 Boss 敌机血量		否	每次召唤 Boss 敌机血量提高 240
游戏难度是否随时间增加	否	是	是
游戏难度随时间增加的周期		每 15 秒提升一次游戏难度	每 10 秒提升一次游戏难度
敌机生成周期	保持 600 毫秒不变	初始为 560 毫秒，每次降低 40 毫秒，最低不低于 240 毫秒	初始为 520 毫秒，每次降低 40 毫秒，最低不低于 160 毫秒
英雄机子弹发射周期	保持 120 毫秒不变	初始为 240 毫秒，每次提高 20 毫秒，最高不高于 600 毫秒	初始为 360 毫秒，每次提高 40 毫秒，最高不高于 600 毫秒
敌机子弹发射周期	保持 600 毫秒不变	初始为 480 毫秒，每次降低 40 毫秒，最低不低于 120 毫秒	初始为 360 毫秒，每次降低 40 毫秒,最低不低于 40 毫秒
普通敌机与精英敌机生成概率比	保持 1 不变	初始为 0.5，每次乘以 0.95	初始为 0.25,每次乘以 0.85
同一时刻敌机数量最大值	保持 5 不变	初始为 6，每次增加 1，最多不超过 15	初始为 7，每次增加 1，最多不超过 20
Boss 敌机生成的分数阈值		初始为 4000,每次降低 200 分，最低不低于 2000 分	初始为 4000,每次降低 200 分，最低不低于 1000 分
普通敌机血量	保持 30 不变	初始为 30,每次增加 3	初始为 30,每次增加 5
精英敌机血量	保持 60 不变	初始为 60,每次增加 5	初始为 60,每次增加 10
Boss 敌机血量		初始为 240，每次增加 10	初始为 240，每次增加 15
精英敌机每次子弹发射数目	保持 1 不变	初始为 1，每次增加 1，最多不超过 5	初始为 1，每次增加 1，最多不超过 6
Boss 敌机每次子		初始为 3，每次增	初始为 3，每次增

弹发射数目		加 1，最多不超过 15	加 1，最多不超过 20
精英敌机子弹伤害	保持 20 不变	初始为 20,每次增加 1，最多不超过 30	初始为 20,每次增加 1，最多不超过 40
Boss 敌机子弹伤害		初始为 20,每次增加 2，最多不超过 50	初始为 20,每次增加 2，最多不超过 60

（3） 角色生成

游戏中主要有 3 种角色需要生成：英雄机、敌机和道具。

英雄机采用**单例模式**创建，在游戏一开始生成，每局中有且仅有一个。所有敌机都采用**工厂方法模式**创建。其中普通敌机和精英敌机每隔一段时间生成，它们的产生概率之比可以根据游戏难度需要进行调整。英雄机消灭敌机时，会有一定的分数奖励。每当游戏得分超过一定阈值时，会生成一架 Boss 敌机。普通敌机不能发射子弹；精英敌机可以发射子弹，采用直射方式；Boss 敌机也可以发射子弹，采用散射方式。不同战机射击策略的设定，通过**策略模式**实现。

当精英敌机或 Boss 敌机被消灭时，都会以一定概率掉落道具。道具总共有 3 种：加血道具、火力道具、炸弹道具。所有道具都是在与英雄机碰撞时才生效。加血道具生效时，能让英雄机恢复一定血量。火力道具生效时，能让英雄机的发射方式由直射改为散射，并增加英雄机每次发射的子弹数，但经过一段时间后，英雄机的发射方式和射击子弹数就会恢复到游戏初始水平。炸弹道具生效时，会清除掉屏幕上所有除 Boss 敌机以外的敌机和敌机子弹，并将对应分数奖励计入总分、生成对应道具。炸弹道具的这一功能，主要通过**观察者模式**实现。

（4） 碰撞检测

游戏中主要进行 4 种类型的碰撞检测：英雄机子弹碰撞敌机、敌机子弹碰撞英雄机、英雄机碰撞敌机、英雄机碰撞道具。

检测到英雄机子弹碰撞敌机时，扣除敌机一定生命值；同理，当敌机子弹碰撞英雄机时，扣除英雄机一定生命值。无论是英雄机还是敌机，生命值一旦≤0，都会坠毁。此外，如果英雄机与敌机直接相撞，两者都会坠毁，游戏结束。检测到英雄机与道具碰撞时，道具生效。

（5） 角色属性控制

游戏角色属性的控制主要包括对英雄机属性的控制和对敌机属性的控制。

火力道具的实现需要对英雄机发射方式、发射子弹数进行改变，加血道具的实现需要对英雄机血量进行改变。此外，不同游戏难度下，英雄机子弹的发射频率也需要进行改变。

敌机的血量、子弹发射数目、子弹伤害、子弹发射频率等属性，也需要随着游戏难度的改变而改变。

此外，英雄机速度通过鼠标控制，普通敌机仅有向下速度（y 速度），精英敌机同时具有向下速度和横向速度（x、y 速度），Boss 敌机仅具有横向速度（x 速度），道具在敌机坠毁的位置生成，仅具有向下速度（y 速度）。

（6） 时间频率控制

敌机产生频率决定多久生成一次敌机，英雄机子弹产生频率决定英雄机多久进行一次发射，敌机子弹产生频率决定敌机多久进行一次发射。可以通过改变它

们的大小来达到改变游戏难度的目的。

此外，游戏难度提高的频率本身也是可以改变的，普通模式相比困难模式，游戏难度提高的频率也会更低。

（7） 角色消亡控制

当英雄机生命值≤0 或与敌机相撞时，判定英雄机坠毁，游戏结束。同理，敌机生命值≤0 时或与英雄机相撞时，判定敌机坠毁。

道具与英雄机碰撞、英雄机子弹与敌机碰撞、敌机子弹与英雄机碰撞，都会导致相应的道具或子弹直接失效。除此之外，敌机、子弹、道具一旦飞出游戏画面的边界，也会被判定为失效。

在游戏过程中，系统会不断对当前所有角色进行检查，删除掉坠毁或失效的战机、子弹和道具。

（8） 结束后处理

当英雄机坠毁时，游戏立即结束，从游戏面板跳转至排行榜面板，进行一定的后处理。

首先会询问用户是否需要保存本局游戏记录，并要求用户输入玩家名。如果用户选择“取消”，则本局游戏不会被记录；反之，若用户选择“确定”，则会根据用户的输入保存本局记录（如果用户输入的字符串为空，则玩家名会被默认设置为“User”）。

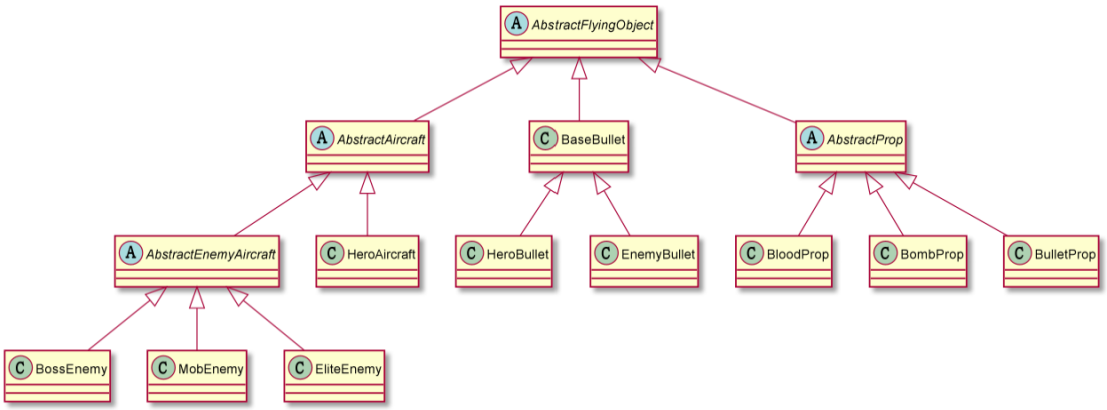
随后，向用户显示该游戏难度下所有已保存的历史记录。这些游戏记录按照总得分的降序排列，包含游戏时间、玩家名等信息。玩家可以选择删除其中的任意一条历史记录。这一功能主要通过**数据访问对象模式**实现。

完成这一切后，用户点击右上角的关闭按钮即可退出本系统。

2.2 类的继承关系分析

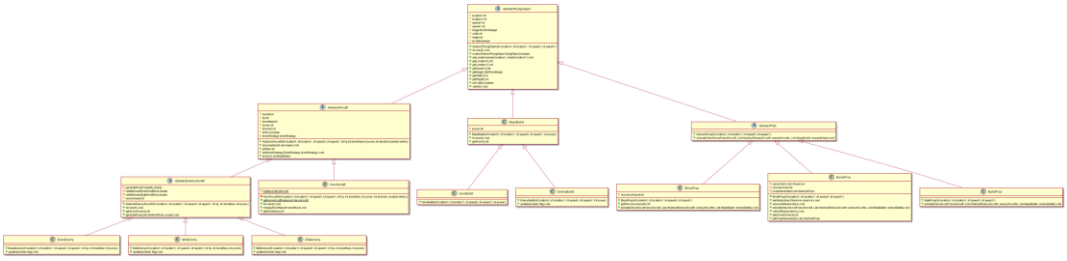
（1）绘制呈现所有飞机类、道具类和子弹类继承关系的 UML 类图

可以根据所设计出来的飞机大战游戏系统，绘制出呈现所有飞机类、道具类和子弹类继承关系的 UML 类图，如下图所示。（来自 uml 目录下的 Inheritence-simplified.puml 文件）

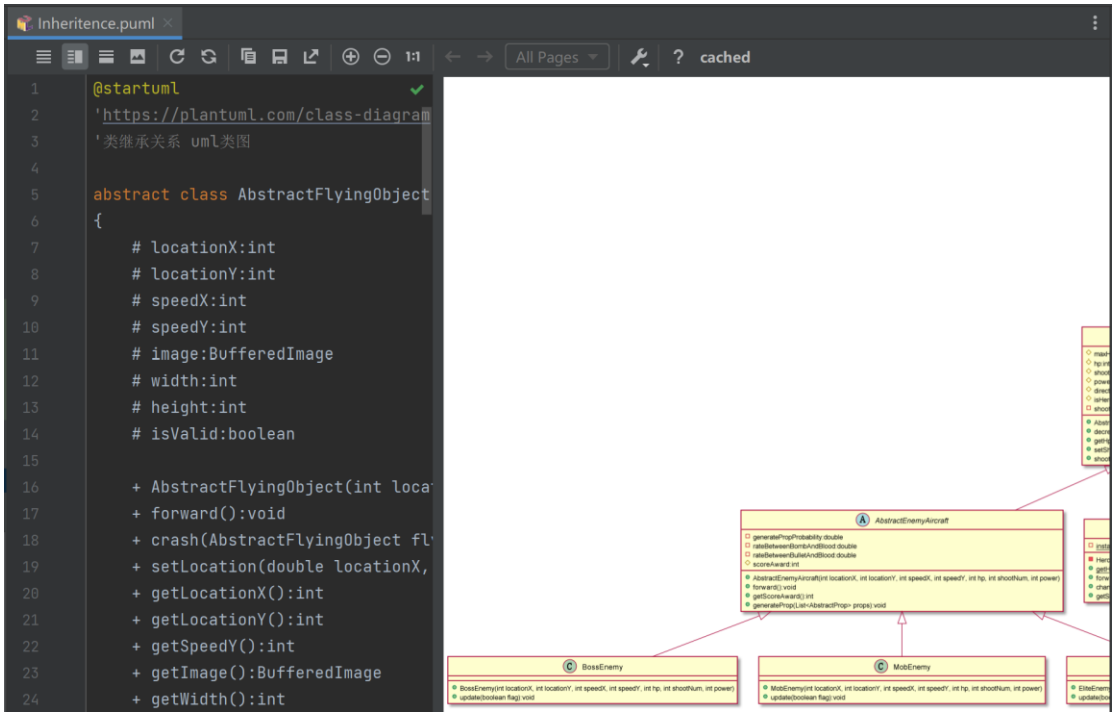


该类图仅包含类名信息和类之间的继承关系，不包含任何类中的属性或方法信息。这是因为这些飞机类、道具类和子弹类中的属性、方法较多，如果采用包含完整属性、方法的 uml 类图，来呈现所有飞机类、道具类、子弹类之间的继承关系，实验报告中的图片会比较模糊，如下图所示。（来自 uml 目录下的

Inheritence.puml 文件）



因此，若要查看清晰且包含完整属性、方法的 UML 类图，可直接查看 uml 目录下的 Inheritence.puml 文件，如下图所示。



（2）继承关系分析

抽象类 AbstractFlyingObject（抽象飞行物）是所有飞机、道具、子弹的公共父类，抽象类 AbstractAircraft、类 BaseBullet、抽象类 AbstractProp 都直接继承自它。

抽象类 AbstractAircraft（抽象飞机）是所有飞机的公共父类，抽象类 AbstractEnemyAircraft、英雄机类 HeroAircraft 都直接继承自它。

抽象类 AbstractEnemyAircraft（抽象敌机）是所有敌机的公共父类，普通敌机类 MobEnemy、精英敌机类 EliteEnemy、Boss 敌机类 BossEnemy 都直接继承自它。

子弹基类 BaseBullet 是所有子弹的公共父类，英雄机子弹类 HeroBullet、敌机子弹类 EnemyBullet 都直接继承自它。

抽象类 AbstractProp（抽象道具）是所有道具的公共父类，加血道具类 BloodProp、炸弹道具类 BombProp、火力道具类 BulletProp 都直接继承自它。

2.3 设计模式应用

2.3.1 单例模式

1.应用场景分析

（1）飞机大战游戏中需要用到单例模式的场景

在飞机大战游戏中，只有一种英雄机，且每局游戏有且仅有一个英雄机实例（由玩家通过鼠标控制移动）。当英雄机被摧毁时，游戏结束，因此在每局游戏中英雄机只能被实例化一次而不能被多次实例化。在游戏结束前的任意时刻，都必须保证英雄机类只有一个实例。

因此，在飞机大战游戏中，单例模式的应用场景为英雄机实例的创建与使用。我们需要保证英雄机类只有一个实例，且能提供一个访问该实例的全局节点。

（2）设计中遇到的实际问题（未使用单例模式重构代码前）

第一，并未使用单例模式对英雄机实例的创建进行限制，通过 new 运算符创建英雄机实例无法保证英雄机实例的唯一性。由于 HeroAircraft 类的构造方法用 public 修饰，是公有方法，所以在 Game 类中可以通过其构造方法创建多个英雄机的实例，违反了每局游戏有且仅有一个英雄机实例这一设计初衷。

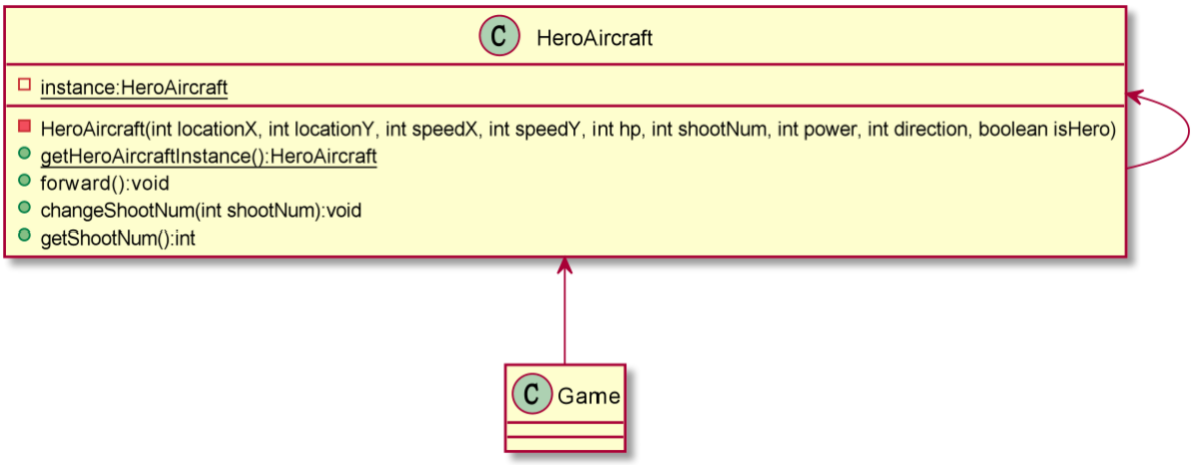
第二，并未对 Game 类隐藏英雄机的创建细节，违反单一职责原则。由于英雄机是在 Game 类当中创建的，生成英雄机所必须的参数（如 locationX、locationY、speedX、speedY、hp 等）都直接由 Game 类传递给 HeroAircraft 类，而不是由 HeroAircraft 类自行决定。Game 类同时承担英雄机的创建、使用职责，这违反了单一职责原则。

因此，需要运用单例模式对代码进行重构解决上述问题。

（3）使用该模式的优势

能够有效达到系统设计的预期，保证在飞机大战游戏中，只有一种英雄机，且每局游戏有且仅有一个英雄机实例，且提供了一个访问该实例的全局节点。此外，对 Game 类隐藏英雄机的创建细节，降低了代码耦合度，符合单一职责原则。

2. 设计模式结构图



HeroAircraft 为英雄机类，其中包含一个 HeroAircraft 类型的静态私有变量 instance，是对自身类型的引用（自我引用），属于一般关联关系，因此存在一个指向自身的箭头。

英雄机类的构造方法被设置为私有，这使得在其它类中不能直接通过调用英雄机类的构造方法（即通过 new 运算符）生成英雄机实例，以防止被多次调用从而创建多个英雄机实例。这样，由类自身负责保存它的唯一实例，同时保证没有其他实例可以被实例化，并提供一个访问该实例的静态公有方法。

静态公有方法 getHeroAircraftInsntance 负责返回这个全局唯一的英雄机实例（如果首次调用该方法，则创建这个唯一的英雄机实例），同时也保证了英雄机的实现细节（如 locationX、locationY 等参数）由 HeroAircraft 类自己决定，对调用者隐藏创建过程。另外，通过增加 synchronized 关键字到该方法中，可迫使每个线程进入到这个方法前，要先等候别的线程离开该方法，从而使得不会有两个线程同时进入这个方法。

Game 类实现了游戏的主逻辑，其中包含有英雄机类实例，存在对英雄机的引用，属于一般关联关系，因此需要绘制一个从 Game 类到英雄机类的箭头。

2.3.2 工厂模式

1. 应用场景分析

（1）飞机大战游戏中需要用到工厂模式的场景

在飞机大战游戏中，有 3 种类型的敌机需要生成（Boss 敌机、精英敌机、普通敌机），有 3 种类型的道具需要生成（加血道具、火力道具、炸弹道具）。敌机可以发射子弹（普通敌机不能发射子弹）对英雄机造成伤害，碰撞英雄机可直接毁灭英雄机、游戏结束。敌机被英雄机子弹击中时生命值下降，当生命值≤0 时坠毁。精英敌机、Boss 敌机坠毁时有一定概率产生道具。道具以一定速度向下移动，与英雄机碰撞或移动至界面底部时消失。当道具与英雄机碰撞时，触发一定效果（恢复英雄机血量、增强英雄机火力、清除敌机与子弹）。

由上述分析不难看出，三种敌机之间、三种道具之间都存在相似的功能和作用，不同之处在于部分属性或方法的具体实现。

因此，可运用工厂模式完成 3 种类型敌机与道具的创建，在父类中提供一个创建对象的方法，允许子类决定实例化对象的类型。

（2）设计中遇到的实际问题（使用工厂模式重构前）

第一，并未对 Game 类隐藏 3 种敌机、3 种道具的创建细节，违反单一职责原则。由于 3 种敌机、3 种道具都是在 Game 类中通过 new 运算符创建的，生成它们所必需的参数（如 locationX、locationY、speedX、speedY 等）都直接由 Game 类传递给相应类，通过 new 运算符完成创建，而非由这些类自行决定。Game 类同时承担创建与使用职责，这违反了单一职责原则。

第二，不利于扩展新类型的敌机和道具，违反开闭原则。如果需要新增其他类型的敌机或道具，则不得不对 Game 类中的代码进行修改，使 Game 类关联更多类，不符合开闭原则（对修改关闭，对扩展开放）。

第三，创建过程面向具体实现而非抽象接口，违反依赖倒转原则。如果需要为敌机新增其他属性（如防御力），则在 Game 类中所有使用 new 运算符创建敌机的代码都需要被修改，新增其它输入（如防御力）。这使得创建过程面向具体实现而非抽象接口，违反依赖倒置原则，抽象敌机类作为所有敌机的公共父类并未发挥真正作用。

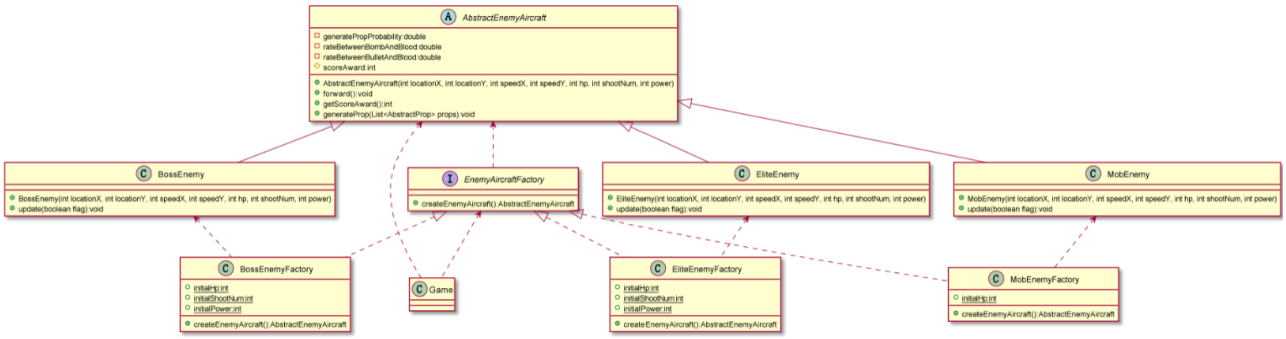
因此，需要运用工厂模式对代码进行重构解决上述问题。

（3）使用该模式的优势

能够对 Game 类隐藏敌机与道具的创建细节，在敌机工厂或道具工厂中决定生成它们的必要参数，而不是在 Game 类中调控制造细节，降低代码耦合度，符合单一职责原则。此外，有利于扩展新类型的敌机和道具，直接增加相应工厂类即可，减少对 Game 类的修改，符合开闭原则（对修改关闭，对扩展开放）。不仅如此，还能让创建过程依赖于抽象接口而非具体实现，发挥抽象敌机类的巨大作用，符合依赖倒置原则。

2. 设计模式结构图

（1）工厂模式创建敌机



AbstractEnemyAircraft 为抽象敌机类。BossEnemy、EliteEnemy、MobEnemy 分别为 Boss 敌机类、精英敌机类、普通敌机类，均继承自抽象敌机类。

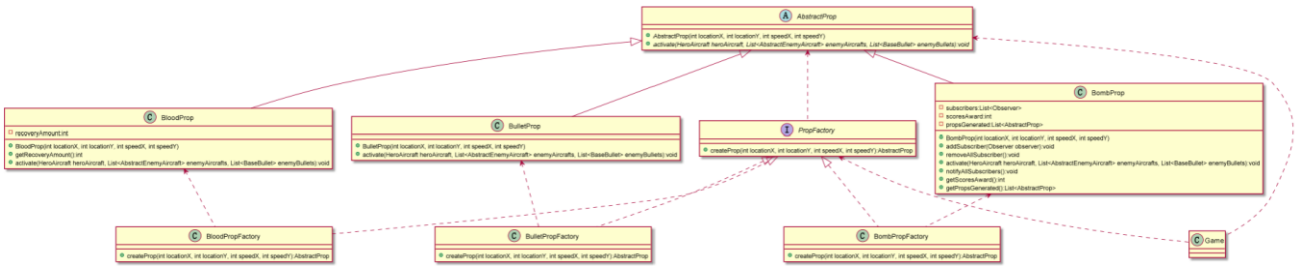
EnemyAircraftFactory 为敌机工厂接口。BossEnemyFactory、EliteEnemyFactory、MobEnemyFactory 分别为 Boss 敌机工厂类、精英敌机工厂类、普通敌机工厂类，均实现了敌机工厂接口。

除前面所述的泛化（继承）关系、实现关系外，该图还体现了依赖关系。在敌机工厂类中创建具体敌机时，需要调用敌机类的构造方法，故三种敌机工厂类功能的实现分别需要对应三种敌机类的协助，因此工厂与敌机之间存在依赖关系。此外，Game 为游戏主逻辑类，其功能的实现需要抽象敌机类、敌机工厂接口的协助，因此 Game 类与它们之间也存在依赖关系。

对部分关键属性或方法的简单说明如以下表格所示。

	名称	含义
属性	initialHp	所设定的初始血量（可随时间变化）
属性	initialShootNum	所设定的初始子弹射击数（可随时间变化）
属性	initialPower	所设定的初始子弹伤害（可随时间变化）
方法	createEnemyAircraft	生产某一类型的敌机
方法	getScoreAward	返回该类型敌机被消灭时的分数奖励值

（2）工厂模式创建道具



AbstractProp 为抽象道具类。BloodProp、BulletProp、BombProp 分别为加血道具类、火力道具类、炸弹道具类，均继承自抽象道具类。

PropFactory 为道具工厂接口。BloodPropFactory、BulletPropFactory、BombPropFactory 分别为加血道具工厂类、火力道具工厂类、炸弹道具工厂类，均实现了道具工厂接口。

除前面所述的泛化（继承）关系、实现关系外，该图还体现了依赖关系。在道具工厂类中创建具体道具时，需要调用道具类的构造方法，故三种道具工厂类功能的实现分别需要对应三种道具类的协助，因此工厂与道具之间存在依赖关系。此外，Game 为游戏主逻辑类，其功能的实现需要抽象道具类、道具工厂接口的协助，因此 Game 类与它们之间也存在依赖关系。

对部分关键属性或方法的简单说明如以下表格所示。

	名称	含义
属性	recoveryAmount	血量恢复值（加血道具专有）
方法	getRecoveryAmount	获得血量恢复值（加血道具专有）
方法	createProp	生产某一类型的道具
方法	activate	道具生效

2.3.3 策略模式

1.应用场景分析

（1）使用到策略模式的场景

在飞机大战游戏中，英雄机的射击方式为直射，但碰到火力道具时改为散射；普通敌机不能发射子弹，无需考虑普通敌机的射击方式；精英敌机的射击方式保持为直射；Boss 敌机的射击方式保持为散射。直射方式和散射方式的区别在于，直射出来的子弹只有纵向速度，但散射出来的子弹同时具有纵向速度和横向速度。

从上述分析中不难看出，英雄机、精英敌机、Boss 敌机这 3 种战机都需要发射出子弹，具有功能相近、行为类似的射击方法。但这 3 种战机的射击方法并不是完全相同的，存在着部分差异，如英雄机刚创建时为直射、精英敌机保持为直射、Boss 敌机保持为散射；即便是同 1 种战机，如英雄机，随着时间的推移，当它与火力道具碰撞时，射击方式也要由直射改为散射。

因此，对于这种有多种算法相似的场景，适合采用策略模式进行维护。策略模式允许射击策略随着对象的改变而改变，可以定义一系列算法，把每一个算法封装起来，使它们可相互替换，从而使得算法能够独立于使用它的客户而变化。

（2）设计中遇到的实际问题（修改代码前存在的问题）

代码的可复用性低，存在着较多重复，不利于后续修改与维护。

修改前的代码在英雄机类、精英敌机类、Boss 敌机类等战机类中都定义了射击方法 shoot()，这些射击方法在实现的时候，大部分代码都是相同的，仅在部分细节上存在差异（如直射和散射方式的主要区别就在于子弹的横向速度是否为 0）。如果需要增加采用直射射击方法的新类型敌机，则几乎是将原有的直射方法代码重新复制了一份，代码的可复用性低，不利于后续修改与维护。

因此，应当使用策略模式对上述代码进行重构。

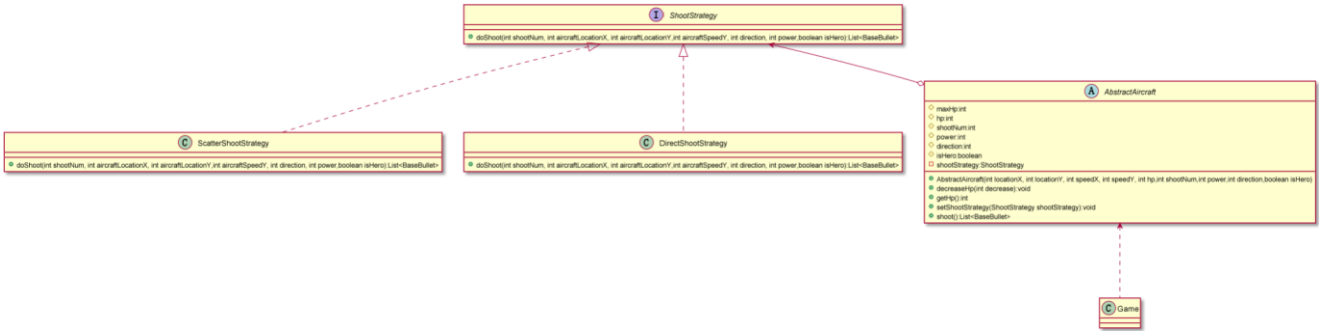
（3）使用策略模式的优势

策略模式定义一系列的算法，把每一个算法封装起来、放入独立的类中，使它们可相互替换，从而使得算法能够独立于使用它的客户而变化。策略模式封装了算法，方便新的算法插入到已有系统中，以及老算法从系统中“退休”，实现方法替换。

在有多种算法相似的情况下，使用多个 if-else 语句将使得代码变得复杂和难以维护，而策略模式允许策略随着对象改变而改变。

此外，策略模式的优点还有：能让不同算法间的切换更加自由、方便；避免多重条件判断；而且扩展性良好，增加新类型的策略比较方便；遵守大部分设计原则，高内聚、低耦合；提供了一种替代继承的方法，而且既保持了继承的优点（代码重用），还比继承更灵活（算法独立，可以任意扩展）。

2. 设计模式结构图



ShootStrategy 为射击策略接口。

ScatterShootStrategy 和 DirectShootStrategy 分别为散射射击策略类和直射射击策略类，均实现了射击策略接口。

抽象飞机类中含有一个私有 ShootStrategy 类型的成员变量，其射击功能的实现依赖于射击策略接口，是整体与部分的关系，因此两者间存在聚合关系。

Game 为游戏主逻辑类，其功能的实现需要抽象飞机类的协助，因此 Game 类和抽象飞机类之间存在依赖关系。

对部分关键属性或方法的简单说明如以下表格所示。

	名称	含义
属性	shootStrategy	当前飞机的射击策略
方法	doShoot	射击策略接口中的射击算法
方法	setShootStrategy	设置当前飞机的射击策略
方法	shoot	当前飞机的射击方法

2.3.4 数据访问对象模式

1. 应用场景分析

（1）使用到数据访问对象模式的场景

在飞机大战游戏中，每局游戏都会记录英雄机的得分（英雄机每消灭一架敌机都会获得一定分数奖励）。游戏结束后，需要保存英雄机当局游戏的得分，并显示在游戏的总分排行榜上，内容包括名次、玩家名、得分和记录时间。此外，玩家还可以选择删除某一历史记录。

因此，在飞机大战中，需要对玩家多次游戏的数据（排名、得分、玩家名、时间）进行保存和显示，并根据玩家需要删除某一历史数据。可以通过数据对象访问模式解决该问题。

（2）设计中遇到的实际问题

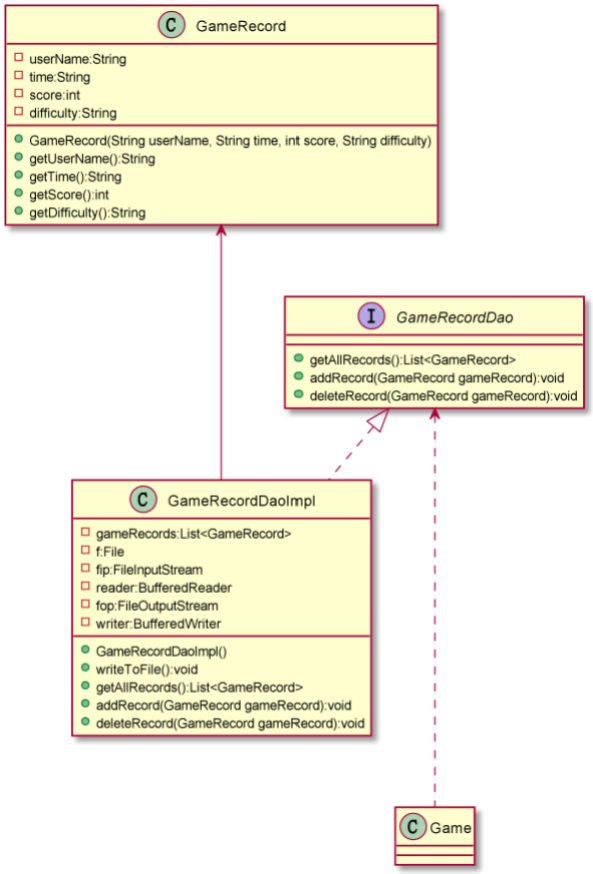
在游戏结束后，需要对玩家的游戏数据进行保存，并对历史游戏记录进行排序、显示或删除。在这个过程当中，需要频繁地读写文件、操作数据。如果不把这些访问数据的代码单独隔离出来，将会导致业务逻辑代码与数据访问代码混杂在一起，使得代码具有较高的耦合度，不利于后续修改与维护。这还同时会导致业务逻辑代码需要额外承担数据访问与修改的任务，违反单一职责原则。

（3）使用数据访问对象模式的优势

可以把低级的数据访问 API 和操作从高级的业务中分离出来。能提供一个访问数据的接口，使得访问过程面向抽象而不是具体实现，符合依赖倒置原则。

此外，由于新增了 DAO 层，能隔离数据层、降低代码耦合度，不会影响到服务或者实体对象与数据库的交互，发生错误时会在该层进行错误抛出。

2. 设计模式结构图



GameRecord 为游戏记录类，DAO 接口 GameRecordDao 中定义了主要的数据访问方法。类 GameRecordDaoImpl 则实现抽象接口 GameRecordDao，由于它含有一个 GameRecord 列表，属于对 GameRecord 的引用，因此它和类 GameRecord 之间存在一般关联关系。

Game 类为游戏主逻辑类，需要游戏数据进行访问和修改，这一功能的实现需要 DAO 接口 GameRecordDao 的协助，因此两者之间存在依赖关系。

对部分关键属性或方法的简单说明如以下表格所示。

	名称	含义
属性	userName	玩家名
属性	time	游戏时间
属性	score	本局总分

属性	difficulty	本局难度
方法	writeToFile	将 DAO 中保存的数据写入文件
方法	getAllRecords	返回一个包含所有游戏记录的列表
方法	addRecord/deleteRecord	添加或删除某一游戏记录

2.3.5 观察者模式

1.应用场景分析

（1）飞机大战游戏中使用到观察者模式的场景

在飞机大战游戏中，英雄机与炸弹道具碰撞后，炸弹道具生效。炸弹道具生效时会清除掉界面上除 Boss 敌机以外的所有敌机和敌机子弹，并将所消灭敌机的分数计入游戏总得分。同时，由于精英敌机坠毁时有一定概率掉落道具，故还需要按照这个概率生成相应的道具。

因此，可以使用观察者模式这一行为设计模式实现上述功能，定义一种订阅机制，可以在对象事件发生时通知多个“观察”该对象的其他对象（即观察者）。在本游戏中，可以将需要消灭的敌机和敌机子弹认为是观察者（或订阅者），将炸弹道具作为被观察的对象。当新事件发生（英雄机碰撞炸弹道具）时，通知所有订阅者调用 vanish()函数，清除相应的敌机或敌机子弹，并再生成新的道具、累加得分。

（2）设计中遇到的实际问题

如果不采用观察者模式，可以通过遍历当前敌机与敌机子弹、逐个调用销毁函数的方式实现炸弹道具的功能。但时，这样做会导致敌机类、敌机子弹类的代码与炸弹道具生效函数的代码耦合在一起，不利于后续修改与维护。与此同时，这种做法还违反了开闭原则，如果需要修改设计为 Boss 敌机也会受到炸弹道具的影响，则不得不重新修改炸弹道具的 activate 方法。

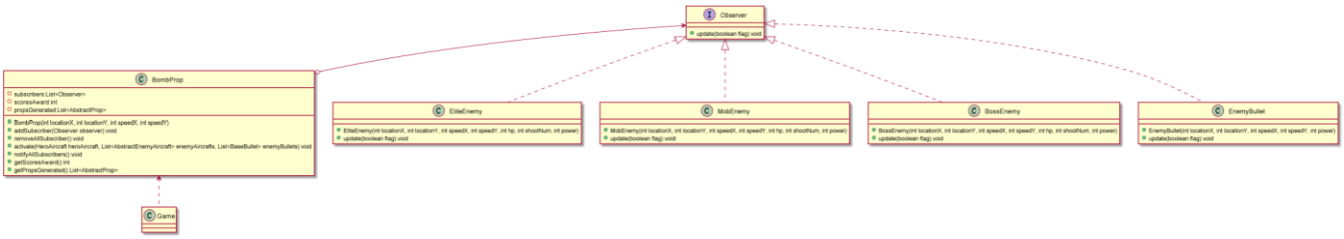
因此，可以使用观察者模式解决上述问题，定义一种订阅机制，当新事件发生（炸弹道具生效）时，被观察者通知观察者处理更新（调用销毁函数）。

（3）使用观察者模式的优势

使用观察者模式而不是仅仅通过简单遍历的方式，实现炸弹道具的功能，能有效降低代码耦合度，便于后续修改与维护。观察者模式在观察者和被观察者之间建立一个抽象的耦合。被观察者角色知道的只是一个具体观察者列表，每一个具体观察者都符合一个抽象观察者的接口。被观察者并不认识任何一个具体的观察者，它只知道它们有一个共同的接口。由于被观察者和观察者没有紧密地耦合在一起，因此它们可以属于不同的抽象化层次。

除此之外，观察者模式还支持广播通讯。被观察者会向所有登记过的观察者发出通知，提醒它们处理更新（本游戏中为调用销毁方法）。

2. 设计模式结构图（如不清晰可查看 uml 目录下 ObserverPattern.puml 文件）



订阅者接口为 Observer，精英敌机类 EliteEnemy、普通敌机类 MobEnemy、Boss 敌机类 BossEnemy、敌机子弹类 EnemyBullet 均实现了该接口。此外，炸弹道具类中含有 Observer 列表，属于对 Observer 的引用，因此它和 Observer 之间存在一般关联关系。

Game 类中实现了游戏的主逻辑，其中使用到了炸弹道具类 BombProp，因此两者之间存在依赖关系。

对部分关键属性或方法的简单说明如以下表格所示。

	名称	含义
属性	subscribers	订阅者列表
方法	addSubscriber	添加订阅者
方法	removeAllSubscriber	移除所有订阅者
方法	activate	道具生效
方法	notifyAllSubscribers	通知所有订阅者
方法	update	观察者对更新进行处理的函数
方法	getScoresAward	获得炸弹所消灭敌机的分数奖励
方法	getPropsGenerated	获得炸弹所消灭敌机掉落的道具

2.3.6 模板模式

1. 应用场景分析

（1）飞机大战游戏中使用到模板模式的场景

在飞机大战游戏中，需要设置三种不同难度的游戏，即简单模式、普通模式、困难模式。用户在开始界面选择以后，会出现对应的游戏地图，游戏难度也会相应调整。

从上述分析中不难看出，三种难度下的游戏具有相似的功能和特点，仅在部分细节或角色的属性上存在差异。因此，可以用模板模式这一行为型模式解决上述问题，在抽象类中定义一个算法的框架，允许子类在不修改结构的情况下重写算法的特定步骤。也就是说，模板方法在抽象类中实现，某些特定步骤在子类中实现。

（2）设计中遇到的实际问题

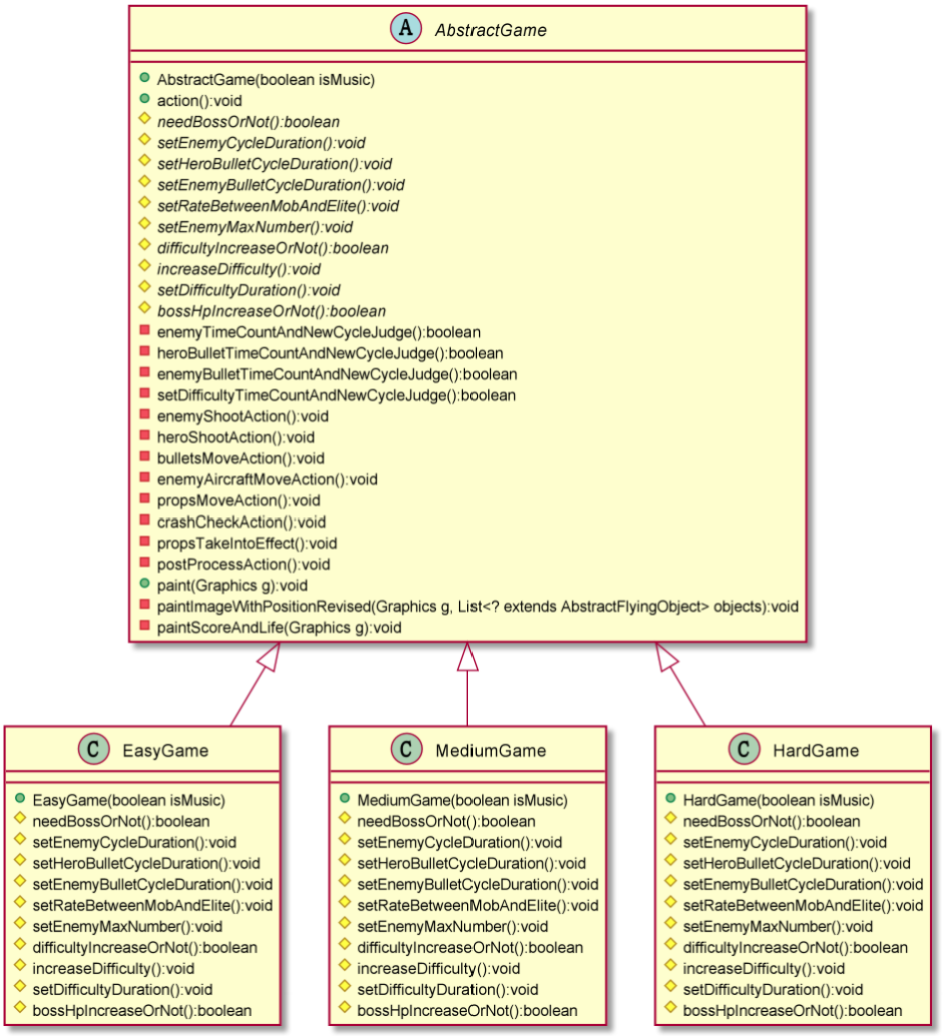
如果不使用模板模式，同样能仅通过类的继承实现三种不同难度的游戏。但是，这样容易导致大量具有相似功能的代码产生重复，降低了代码的可复用

性，不利于后续修改与维护，不符合开闭原则。如果需要新增其它难度的游戏，则需要大量使用已有代码，可扩展性差。

（3）使用模板模式的优势

模板模式定义了一个操作中的算法骨架，差异性的实现交由子类完成。通过模板模式，能将相同处理逻辑的代码放到抽象父类当中，把不变行为搬运到超类，去除子类中的重复代码，提高代码的可复用性。同时，将不同的代码放入不同的子类中，通过子类的扩展增加新的行为，让子类实现父类中的某些细节，提高代码的可扩展性，符合开闭原则。

2. 设计模式结构图



AbstractGame 为抽象游戏类，其中定义了所有不同难度游戏运行的一个框架，并包含有不能让子类重写的模板方法。简单难度游戏类 EasyGame、普通难度游戏类 MediumGame、困难难度游戏类 HardGame 均继承自这个抽象游戏类，并重写了其中的部分方法，以实现不同难度的游戏。

对部分关键方法的简单说明如以下表格所示。

	名称	含义
方法	needBossOrNot	是否需要生产 Boss 敌机
方法	setEnemyCycleDuration	设置敌机生成的周期
方法	setRateBetweenMobAndElite	设置普通、精英敌机的生成概率比
方法	setDifficultyDuration	设置经过多久提升一次游戏难度
方法	bossHpIncreaseOrNot	每次召唤是否需要提高 Boss 敌机血量

3 收获和反思

通过本学期面向对象的软件构造导论实验，我体验了一次软件开发的完整流程，并进一步巩固了所学的 Java 编程语言基础知识，如继承、Swing、多线程等，同时也熟悉了目前常见的面向对象设计模式，如单例模式、工厂模式、策略模式、观察者模式等。通过完成这六次实验，我的动手实践能力、代码调试能力、版本管理能力均得到了较大提高，进一步深化对理论课所学知识的理解，同时也为后续课程学习奠定坚实的基础。

我在实验过程中遇到的一个印象较为深刻的问题，是在实验五利用多线程实现音乐播放时，出现了卡顿的现象。后来经过检查发现，子弹击中敌机音效、Boss 敌机登场音效的播放，我都始终使用的是它们各自的同一个线程来实现。也就是说，我没有在上述两种音效播放结束后，重新创建新的线程，而是继续使用原来的线程来尝试播放，调用其 start 方法。然而，在 Java 中每个线程只能执行一次，线程一旦终止以后就进入死亡状态，不能被重新启动，所以我不能再使用原有线程来播放新一轮的音乐，强行调用 start 方法只会导致程序卡顿。修改代码后，每次子弹击中敌机或 Boss 敌机登场时，我都使用 new 运算符重新创建一个线程进行播放，成功解决上述问题。

另外，我还在实验六观察者模式的实现过程中遇到问题。当时我发现英雄机与炸弹碰撞时会出现卡顿现象。检查后发现，原来是因为我在使用 for 循环遍历订阅者清单的同时删除订阅者，从而导致出错。直接改为调用 List 的 clear 方法清空其中元素即可解决该问题。

调试能力时衡量一个人编程水平的重要环节。通过六次实验，我寻找程序问题的能力得到了进一步提高。在实验三中，我们使用了 JUnit 对代码进行单元测试，掌握代码调试的基本技能，同时安装阿里编码规约插件，养成规范的代码习惯。当程序运行出现与预期不符的现象时，我也能灵活运用控制台打印的方法，弄清楚程序可以正确执行到哪个位置、到底是运行到哪一步出的问题，成功定位到问题所在。此外，我从实验刚开始的时候，就听从老师建议，使用 Git 对代码进行版本控制管理，当需要对现有代码进行修改时，也能轻松找回历史版本。

实验结束后实验报告的撰写，有利于我们养成严谨认真的科学态度，同时也便于我们日后回顾、反思之前的实验。每次 UML 类图的绘制，虽然耗费时间较长，但却在无形之中加深了我对所学设计模式的理解，让我重新审视所学的每一种设计模式的本质是什么、其核心代码到底应该怎么写，同时也对各个类、接口的作用以及它们之间的关系有了更清晰的认识。

至于建议，我认为如果能在六次实验过程中增加更多的需求变动，或许能让

我们更加充分认识到所学设计模式的优势之处。实验中飞机大战游戏系统有相当一部分功能都是通过各种设计模式实现的，但事实上这些功能不采用设计模式也能够实现。如果能够增加更多的需求变动，我们也许能更加明白为什么使用设计模式实现上述功能会更好，从而更加充分感受到这些设计模式所带来的可复用性、可扩展性、低耦合性。

总的来说，本学期《面向对象的软件构造导论》课程实验还是让我收获较大。