哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2022 春季

课程名称：面向对象的软件构造导论

实验名称：飞机大战游戏系统的设计与实现

实验性质：设计型

实验学时：16 地点：T2506

学生班级：13 班

学生学号：200111325

学生姓名：杨行

评阅教师：

报告成绩：

实验与创新实践教育中心制

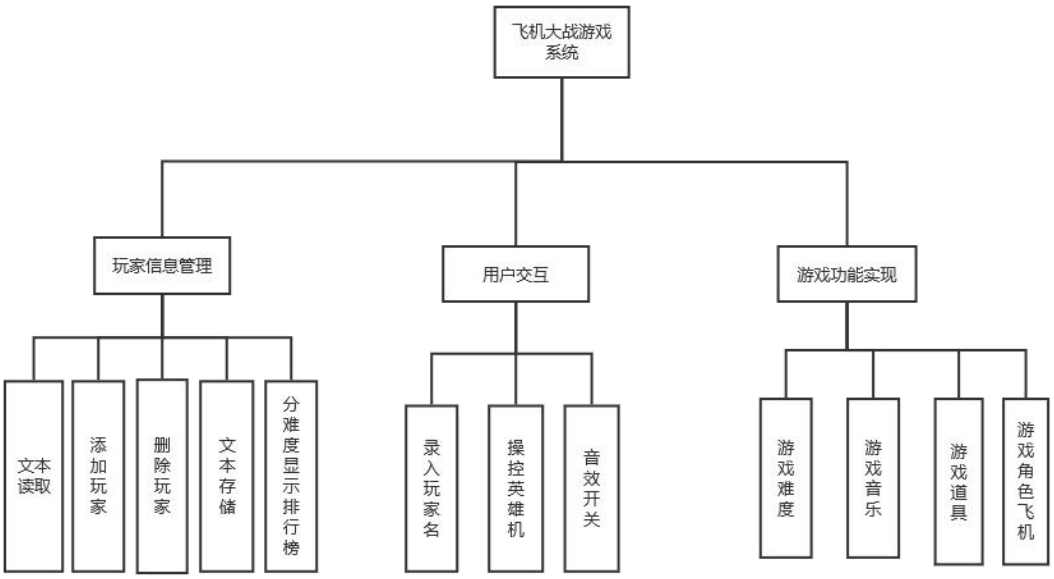
2022 年 4 月

1 实验环境

操作系统：win10，
主要开发工具：IntelliJ IDEA 2021.3.2

2 实验过程

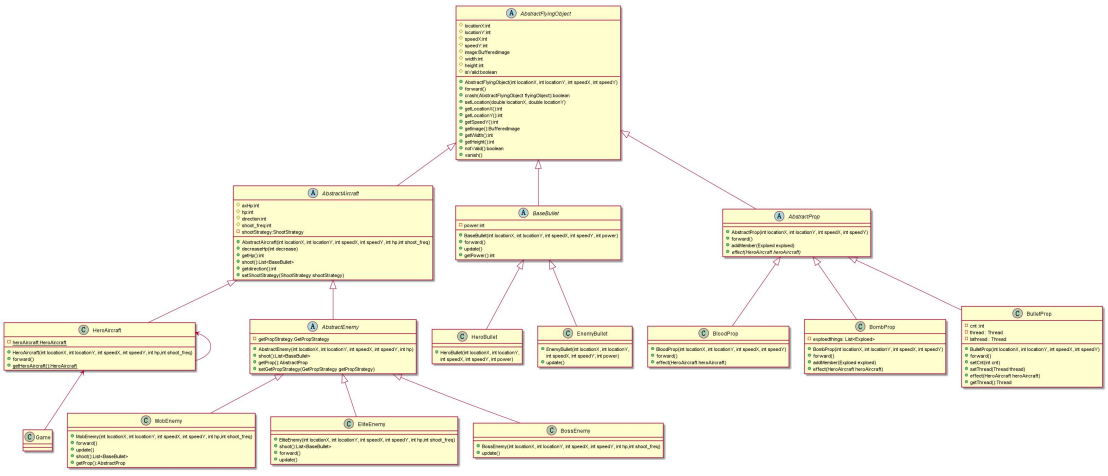
2.1系统功能分析



系统主要实现了以下几个功能：

1. 用户交互功能，实现用户对英雄机的控制，对音效开关的选择以及游戏结束后录入玩家信息。
2. 玩家信息管理功能，实现每次游戏结束添加本次玩家信息的功能，以及将玩家信息根据游戏难度存储于不同文本，并可以从文本中读取玩家信息依据不同难度通过玩家分数将分数排行榜显示出来。
3. 游戏功能，实现游戏难度分级，不同难度，游戏呈现效果不同；实现音乐功能，关闭时游戏静音，开启时实现音效的播放与切换；实现游戏道具功能，不同的道具对英雄机有不同的增益效果；实现游戏角色飞机的出现，射击，物品掉落等功能。

2.2 类的继承关系分析



如图，AbstractFlyingObeject 下有三个子类 AbstractAircraft、AbstractProp 和 BaseBullet。AbstractAircraft 下有两个子类，分别是英雄机类 HeroAircraft 和所有敌机的父类 AbstractEnemy。AbstractEnemy 有三个子类分别代表不同敌机的普通敌机类 MobEnemy,精英敌机类 EliteEnemy,Boss 敌机类 BossEnemy。AbstractProp 有三个子类，代表不同的道具，加血道具类 BloodProp，火力加强道具 BulletProp,爆炸道具 BombProp。BaseBullet 有两个子类 HeroBullet 和 EnemyBullet 代表英雄机子弹和敌机子弹。同时 AbstractEnemy 和 BaseBullet 还实现了观察者模式中所完成的接口 Exploed。

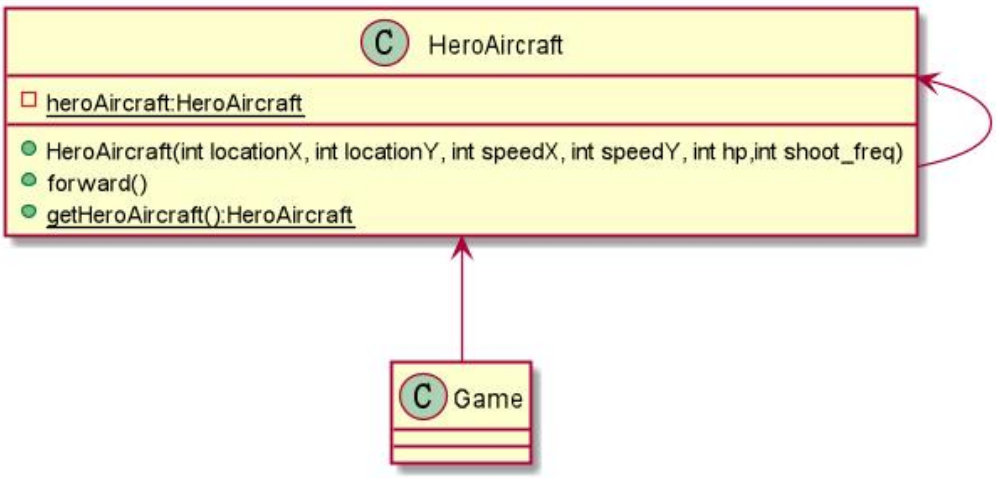
2.3设计模式应用

2.3.1 单例模式

1.应用场景分析

应用单例模式的场景是英雄机的构建，由于在飞机大战游戏中只有一种英雄机，且每局游戏只有一架英雄机，要保证英雄机的类必须只有一个实例存在，所以考虑使用单例模式。起初英雄机构造为公共方法，且英雄机的构建和调用都在 Game 文件中，违背了单一职责原则的同时增加了在一次游戏进程中创建多个英雄机的可能，可能造成内存浪费，功能异常，安全性降低等问题。采用单例模式可以有效增强安全性，避免资源浪费。

2.设计模式结构图



解决方案：采用单例模式的思想,运用双重检查锁定的方法，重构代码。首先将英雄机类 HeroAircraft 的构造方法声明为私有，同时在类内部创建一个私有成员 heroAircraft 用来保存唯一实例, 外部若想调用只能通过 getHeroAircraft 方法获得该实例。进而提高安全性。

描述：图中主要有两个类，Game 类和 HeroAircraft 类。

HeroAircraft 类的构造方法为私有，Game 类不能直接调用 HeroAircraft 类来创建实例。HeroAircraft 类中有一个私有成员变量 heroAircraft 用来保存该唯一实例。HeroAircraft 类的 getHeroAircraft 方法可以返回该类的唯一实例。并且为了确保线程安全，不出现多个线程同时第一次调用 getHeroAircraft 方法的情况，该方法加入了线程检查(即双重检查锁定)。HeroAircraft 为构建函数定义了英雄机 X 轴位置，Y 轴位置，X 轴速度，Y 轴速度，血量以及射击频率等基本属性。getHeroAircraft 方法以单例模式的思想达到返回该类的唯一实例的目的。Game 通过调用 getHeroAircraft 达到在画布上加入英雄机的作用，达到单一职责。

2.3.2 工厂模式

1.应用场景分析

敌机与道具的构造需要应用到此模式。游戏中有 3 种类型敌机：普通敌机、精英敌机、 Boss 敌机。普通敌机和精英敌机以一定频率在界面随机位置出现并向屏幕下方移动。Boss 敌机悬浮于界面上方，直至被消灭。 游戏中还有 3 种类型道具：火力道具、炸弹道具、加血道具。敌机坠毁后，以较低概率随机掉落某种道具。英雄机通过碰撞道具后，道具自动触发生效。根据游戏的需求看出 3 种敌机和 3 种道具可以分别归为一类，联想到应用工厂模式。

实验初期敌机的创建依然放在 Game 类，道具的产生没有分类清晰，使得代码冗杂。同时如果需要改变敌机的部分内容，例如速度，位置等信息，需要对客户端代码进行操作，不遵守开闭原则。

改用工厂模式后，代码相比简约，敌机的速度位置等修改可以在工厂类中进行，无需更改 Game 类中代码，安全性提高。

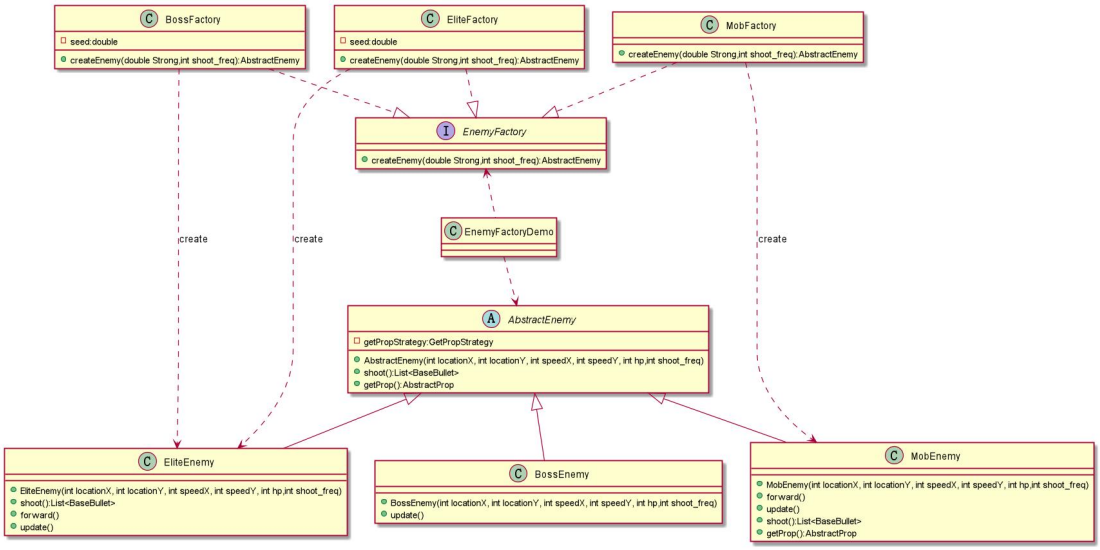
2.设计模式结构图

解决思路:

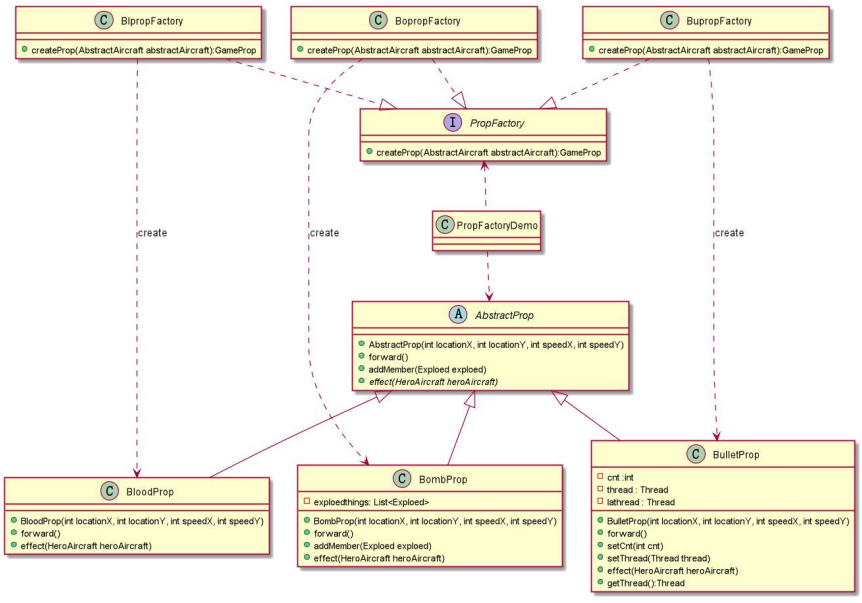
重构代码。设计创建敌机的工厂接口 EnemyFactory，其下创建三个类 MobFactory、EliteFactory、BossFactory，分别实现对应敌机的创造，实现接口功能。

同理设计创建道具的工厂接口 ProbFactory，由类 BlpropFactory、BopropFactory、BupropFactory，实现具体道具的创建工作。 客户端 Game 则通过调用对应敌机工厂类的创建方法达到加入敌机的效果。此外，道具工厂类的创建方法由应用策略模式实现道具掉落的接口 GetPropStrategy 下的掉落方法类调用，相应的敌机用不同的策略方法，再通过敌机类中的方法将产生的道具加入 Game 类。

敌机工厂 UML：



道具工厂 UML:



功能分析：

首先分析敌机创建的工厂模式类图：EnemyFactory 是一个接口， 它拥有一 个方法 createEnemy。 由 MobFctory、BossFactory、EliteFactory 三个敌机工厂类分别具体实现该方法，分别创建出普通敌机 MobEnemy、Boss 敌机 BossEnemy、精英敌机 EliteEnemy。MobFctory、BossFactory 两个类内部额外用一个随机数种子 seed 控制产生飞机的 x 轴方向。createEnemy 方法调用时需要设定两个参数 Strong 和 shoot_freq 代表机体属性的强化属性和射击频次。图中三个敌机类是类 AbstractEnemy 的子类，MobEnemy 通过重写 AbstractEnemy 中的方法 shoot 和 getProb 使其返回空进而让普通敌机不射击，不掉落道具，Boss 敌机 BossEnemy、精英敌机 EliteEnemy 通过设置策略接口类型变量实现射击和生成道具的效果，再由客户端调用公共方法加入道具和子弹。

对于道具创建的工厂模式类图:ProbFactory 是一个接口，它拥有一个方法 createProb。BuPropFactory、 BoPropFactory、BlPropFactory 则分别具体实现该抽象方法，分别创建子弹道具 BulletProp、炸弹道具 BombProp、以及 回血道具 BloodProp。这三个道具类是抽象类 GameProb 的子类。可以实现 GameProp 中的抽象方法 effect，分别产生增强火力，爆炸，回血等效果。createProb 方法调用时需要设定一个抽象飞机类参数 abstractAircraft 用于设定道具掉落位置及速度。

2.3.3 策略模式

1. 应用场景分析

应用策略模式的场景是飞机完成射击功能以及敌机坠毁时产生道具两种场景，在飞机大战游戏中，英雄机和各种类型的敌机所发射子弹的图片、子弹数量、火力值和弹道都不相同。普通敌机不发射子弹，英雄机、精英敌机和 Boss 敌机按照各自方式发射子弹。且火力道具生效后英雄机弹道会发生改变。故可以使用策略模式。

起初射出子弹的代码均写在英雄机、精英敌机和 Boss 敌机类的内部，产生道具的代码写在精英敌机和 Boss 敌机类的内部，如果发射方式或产生道具方式需要替换，需

要更改代码。效率很低，且不符合开闭原则。

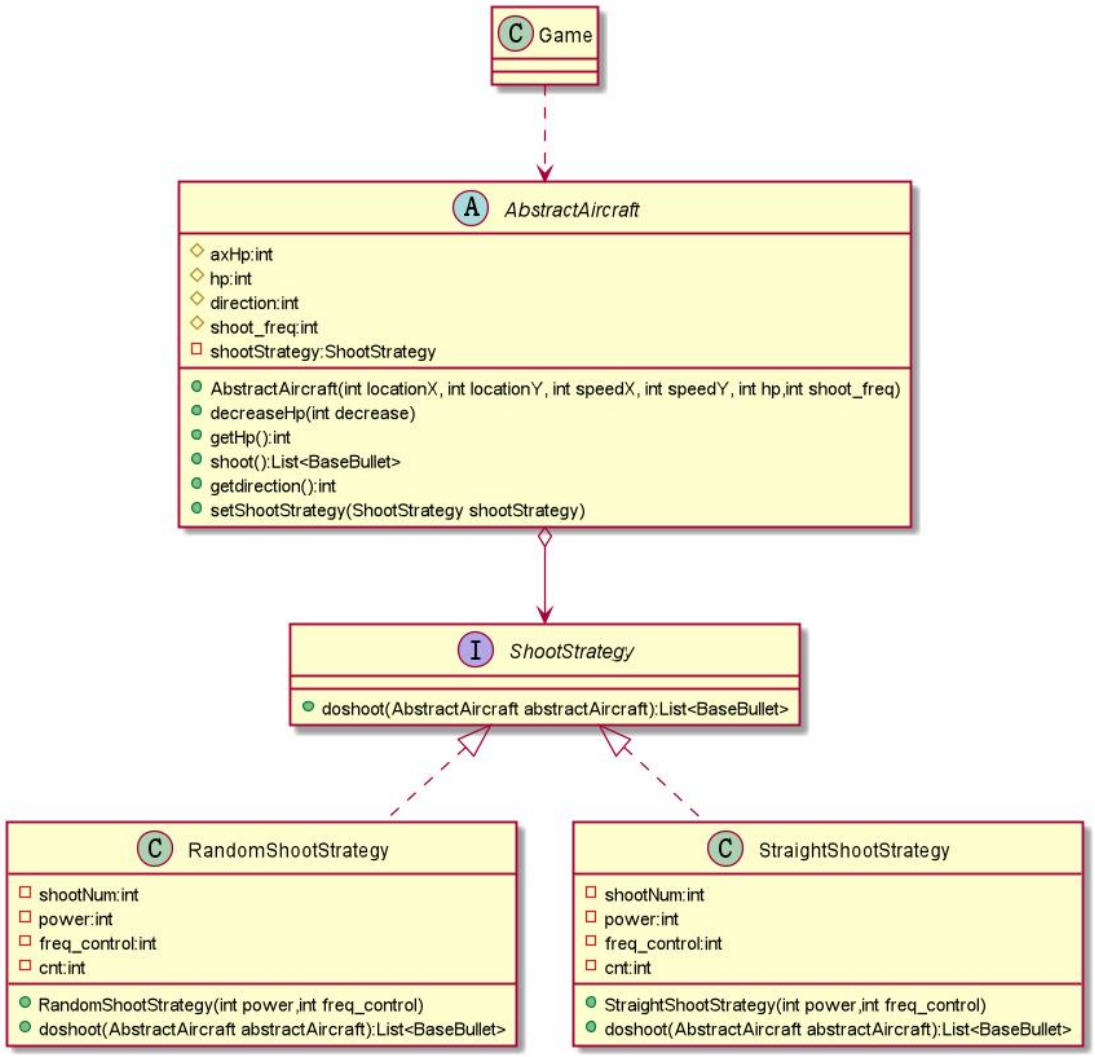
改用策略模式后，解决了英雄机射击方法相互切换的问题，更加灵活多变。同时，使用策略模式还使得代码更易于维护和拓展，降低了耦合性。

2. 设计模式结构图

解决方案：

运用策略模式的思想，将射击的算法封装成一个一个类，实现同一个射击方法接口，产生道具的算法封装成一个一个类，实现同一个产生道具方法的接口。所有飞机的父类 AbstractAircrafct 使用射击方法接口完成射击功能。所有敌机的父类 AbstractEnemy 使用产生道具方法完成产生道具功能。飞机子类中只需设置相应的策略。

射击策略 UML:



描述：

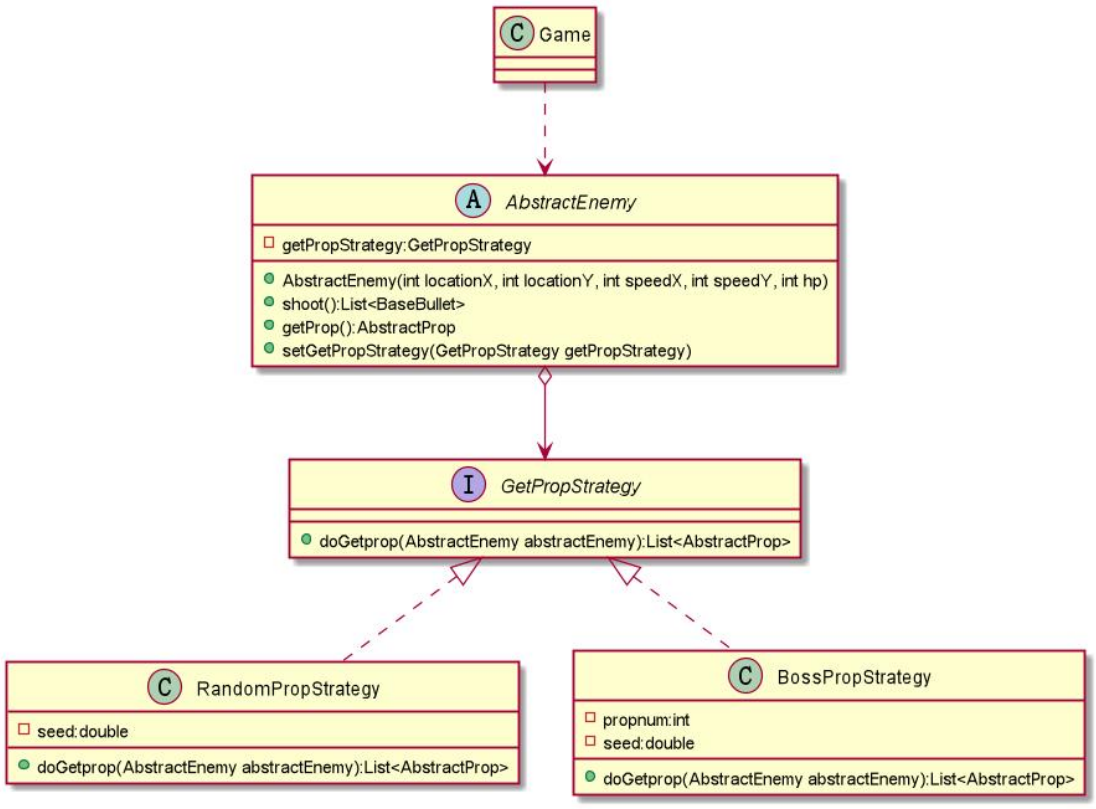
图中 ShootStrategy 为射击策略接口，RandomShootStrategy 为散射策略类，StraightShootStrategy 为直射策略类,内部定义了 int 型变量 shootNum,power,freq_control,cnt,表示单次射击子弹数，子弹伤害,射击频次。通过 doshoot 函数产生子弹集合完成对应的子弹射击功能。

AbstractAircraft 作为环境类使用射击策略,AbstractAircraft 内部设定 int 类型的变量 direction 和 ShootStrategy 类型的变量 shootStrategy。通过 setShootStrategy 方法设置飞机所使用的射击策略。通过 shoot 方法调用射击方法的 doshoot 方法获得子弹集合，doshoot 通过 direction 判断射击方向，从而辨别英雄机或敌机，使用对应子弹图片。火力道具只需通过 setShootStrategy 方法更改英雄机所使用的射击策略达到更改弹道的效果。

MobEnenmy 并未采用策略方法，而是通过重写 AbstractEnemy 中的方法 shoot 和 getProb 使其返回空进而让普通敌机不射击，不掉落道具。

Game 类作为游戏端呈现效果。

道具掉落策略 UML:



描述:

图中 GetPropStrategy 为产生道具策略接口,RandomPropStrategy 为随机产生单个道具的策略类，内部定义随机数种子 seed，StraightShootStrategy 为 Boss 机坠毁产生道具策略类，内部定义 propnum 表示一次产生道具数量同时定义随机数种子 seed 随机产生 propnum 个道具。通过 doGetprop 方法产生道具集合完成对应的生成道具功能。

由于敌机坠毁才产生道具，AbstractEnemy 作为环境类使用产生道具策略，内部设定 GetPropStrategy 类型的变量 getPropStrategy.通过 setGetPropStrategy 方法设置飞机所使用的射击策略。进而通过 getProp 方法调用对应策略的 doGetProp 方法得到道具集合。

Game 类作为游戏端呈现效果。

2.3.4 数据访问对象模式

1. 应用场景分析

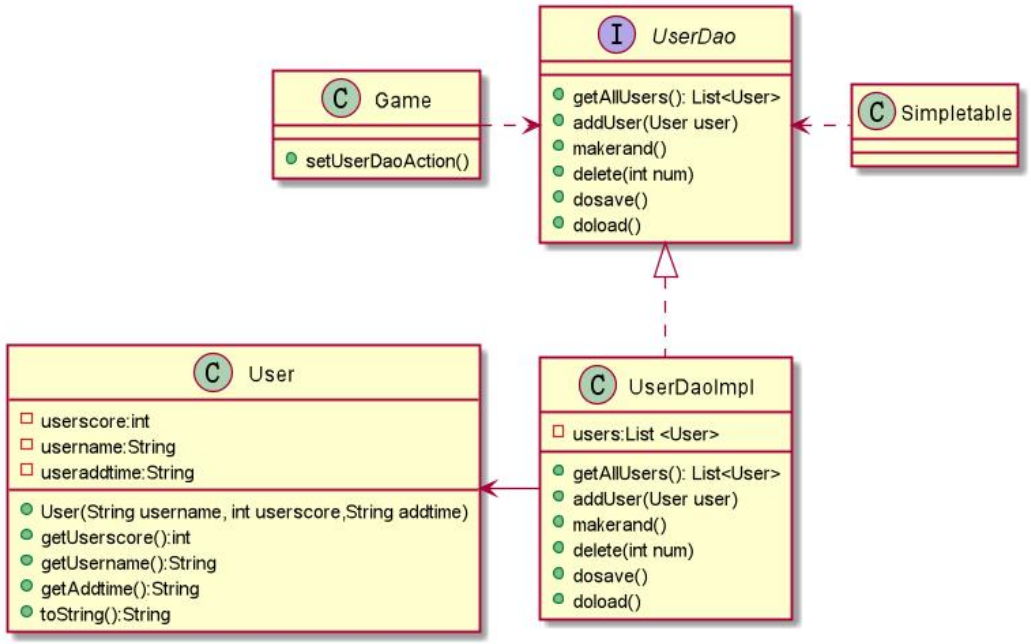
飞机大战游戏中，需要实现简单、普通和困难三个难度的排行榜。排行榜包括玩家名、玩家得分和添加时间三个属性。显示排行榜时，需要根据不同难度以玩家的得分来排名次，并将其保存在本地文件当中。此时就需要使用 DAO 模式。

使用 DAO 模式将低级别的数据访问逻辑和高级别的业务处理分离，避免在 Service 层上直接操作底层数据。同时，使用 DAO 模式不需要修改代码就可以对底层数据库进行更换，也能够让程序逻辑更见清晰，利于扩展。

2. 设计模式结构图

解决方案：采用数据访问对象模式的思想，将用户数据看作一个类，建立一个数据访问对象接口，在建立一个实现接口的实体类处理用户数据。

UML 图：



描述：

UserDao 是数据访问对象接口，UserDaolmpl 是实现接口的具体类，User 则是用户类，UserDao 中设定 getAllUsers 方法用于得到用户数据集合，addUser 方法用于添加用户，makerand 方法用于对用户按照分数排序，print 方法打印得分排行榜，dasave 方法将用户数据存入文本，doload 方法从文本读入数据存入 User 数组集合，delete 通过传入的 int 变量 num 删除用户。UserDaolmpl 定义模拟数据库数据的 User 类型链表 users，并具体实现接口的方法。

User 类中定义用户数据，玩家名 username、 得分 userscore 和记录时间 addtime，名次用在链表所处位置记录。并设置 getUserscore, getUsername, getAddtime 方法可以分别获取用户相应数据。

Game 和分数表 UI Simpletable 应用 UserDaolmpl 实现对数据库的操作。

2.3.5 观察者模式

1. 应用场景分析

在飞机大战游戏中，炸弹道具需要实现英雄机碰撞炸弹道具后，道具生效，可清除界面上除 Boss 机外的所有敌机和敌机子弹，同时扣除 Boss 机的一部分血量。将英雄机碰撞炸弹道具看作对象事件，敌机和子弹看作观察对象。自然想到了观察者模式。

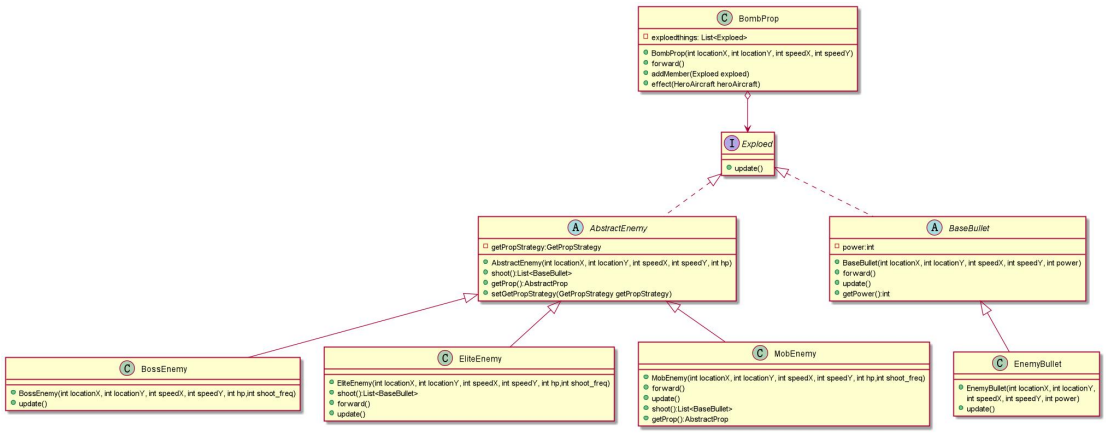
使用观察者模式可以使观察者和被观察者是抽象耦合的，并建立一套触发机制，可以较为方便的添加或减去观察者类型。

2. 设计模式结构图

解决方案：

将 BombProp 类作为对象事件的发布者，敌机和敌机子弹看作观察者，创建一个观察者类实现的接口，接口中含有根据对象事件发生而改变的方法。

UML 图：



描述：

BombProp 类作为对象事件的发布者，建立一个订阅对象类的链表，通过 addMember 方法添加订阅者具体对象于链表中。通过 effect 方法通知所有在链表中实现 Exploed 接口的订阅者类实现 update 方法达到销毁或减血的效果。effect 方法的调用在客户端中，达到一次通知多个对象的效果。

2.3.6 模板模式

1. 应用场景分析

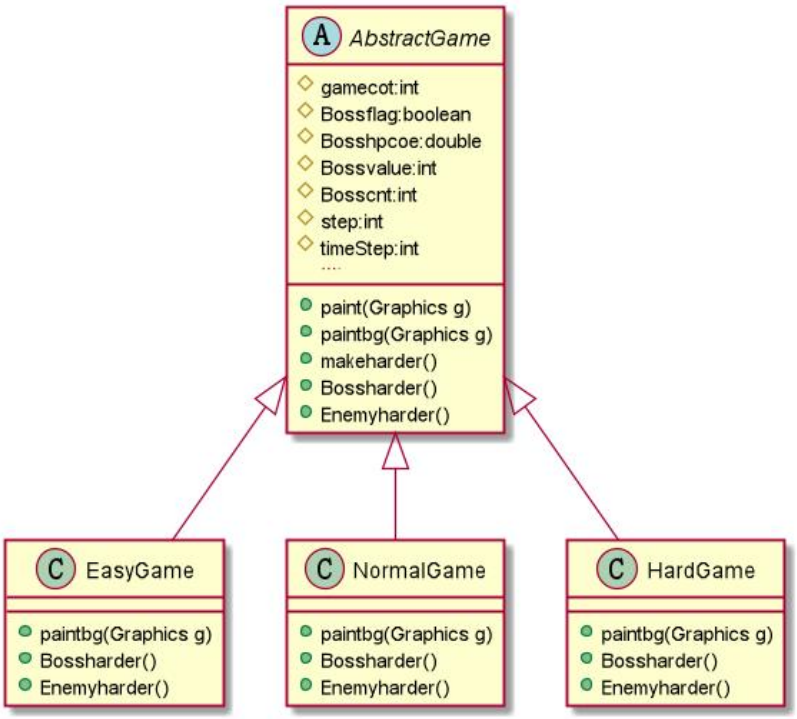
用户进入飞机大战游戏界面后，可选择某种游戏难度：简单 / 普通 / 困难。用户选择后，出现该难度对应的地图，且英雄机和敌机的战斗力会相应调整。于是游戏的运行逻辑基本不变，需要改变的只有背景的绘制，难度的调整。联想到运用模板模式，将 Game 类调整为父类下创建 3 个子类，子类拓展背景绘制方法和难度调整方法。

使用模板模式可以封装不变部分，扩展可变部分。提取公共代码，便于维护。 行为由父类控制，子类实现，便于增设难度。

2. 设计模式结构图

解决方法：将 Game 类调整为父类下创建 3 个子类，并创建背景绘制方法和难度调整方法，通过子类实现。

UML 图：



描述：Game 类改为抽象类，新建抽象方法 paintbg 方法用于绘制背景图片，新建 Bossharder 方法用于提升与 boss 相关的难度，Enemyharder 方法用于提升其他类型敌机的难度。makeharder 方法用于判断条件用于调用 Enemyharder 方法和 Bossharder 方法。新建的保护属性成员

gamecot：敌机强化系数

Bossflag：boss 机存在标志（简单模式一开始设为 true，保证不会出现 boss 机）

Bosshpcoe:boss 机血量提升系数

Bossvalue：boss 机首次出现的阈值

step:boss 机出现的阈值的分数间距

timestep:除 boss 机外其他机体强化的时间间距

Enemyharder 方法，Bossharder 方法通过修改上述变量，在工厂实现对应敌机的创造时导入系数参数完成难度提升。

3 收获和反思

收获：

本次实验初步完成了一个单机版的飞机大战游戏，实验课进一步帮助我们深入理解理论课上学到的面向对象思想以及各种设计模式、多线程等相关知识。

通过完整的实验过程，我们基本了解了 java 语言的语法、思想和特点，在锻炼了我们代码能力的同时，更让我们进一步理解了面向对象的设计思维。相较于之前的课程实验单元剧的实验内容，本次实验以连续剧的方式逐步完成一个游戏项目，并在设计中不断完善，让我意识到重构与维护代码的重要性。作为设计型实验，本次实验也有较大的开放性，让我们初步体会到设计项目的步骤和思路。锻炼了我们统筹一个整体项目的能力。本次实验展示了软件开发并不是只需要实现功能，更要保证开发的软件便于维护更新，积极面对可能的变化。

问题：

实验一时，对原有代码的不了解，造成初步完善功能时，存在图片未显示的错误。

实验二时，对工厂模式理解不足，道具的生成代码存在一些问题。

实验三时，检测手段单一，assertEquals 是主要检测方式。

实验四时，信息存储时解决不了序列化导出文本乱码问题，改用普通的文本读写。

实验五时，swing 布局操作不熟练，MusicThread 的代码未深入理解，使用 interrupt（）无法暂停音效，发现与代码采用写入方式有关，改用标志变量。用户交互时输入姓名中间不能含有空格，是由于设置的普通的文本读写按空格读取，可以改成序列化或玩家输入时检验去除空格得以解决。

实验六时，模板模式理解不透彻，起初未找到正确的构建难度提升方式。

意见：

实验一时，最好对代码有一个较为细致的说明，避免完善时需要通读代码，理解功能。可以对于一些功能实现细节，例如文本读写，用户交互时输入姓名框组件的使用给一些具体例子（不局限于一种方法），让大家将更多的精力放在设计模式的同时更深刻理解代码语言。