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

实验报告

开课学期：2023 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：T2608

学生班级：计算机 2 班

学生学号：210110205

学生姓名：王良希

评阅教师：

报告成绩：

实验与创新实践教育中心制

2023 年 4 月

1 实验环境

请填写实验所用到的操作系统和主要开发工具。

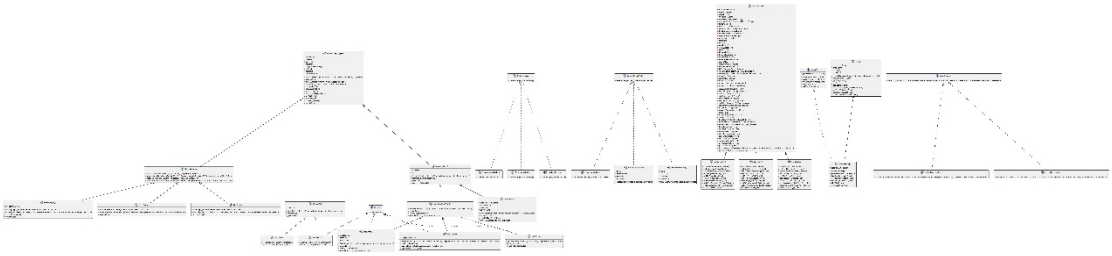
操作系统：Microsoft Windows 21H2 19044.2486 专业工作站版。

主要开发工具：Intellij IDEA 2022.3.2。

2 实验过程

2.1 类的继承关系

请根据面向对象设计原则，分析和设计游戏中的所有飞机类、道具类和子弹类，并使用 PlantUML 插件绘制相应的 UML 类图及继承关系，类图中需包括英雄机、所有敌机、道具、子弹及它们所继承的父类。



根据该 UML 类图，类之间的继承关系如下：

- AbstractFlyingObject 是 AbstractAircraft、AbstractSupply 和 AbstractAircraft 的父类。
- BaseBullet 是 HeroBullet 和 EnemyBullet 的父类。
- AbstractSupply 是 HealingSupply、BombSupply 和 FireSupply 的父类。
- AbstractAircraft 是 HeroAircraft 和 AbstractEnemy 的父类。
- AbstractEnemy 是 MobEnemy、EliteEnemy 和 BossEnemy 的父类。
- Observer 是 EnemyBullet、MobEnemy、EliteEnemy 和 BossEnemy 的实现接口。
- SupplyFactory 是 HealingSupplyFactory、BombSupplyFactory 和 FireSupplyFactory 的父类接口。
- EnemyAircraftFactory 是 MobEnemyFactory、EliteEnemyFactory 和 BossEnemyFactory 的父类接口。
- ShootStrategy 是 ScatterShootStrategy 和 DirectShootStrategy 的父类接口。
- recordDao 是 recordDaoImpl 的父类接口。
- AbstractGame 是 CommonGame、EasyGame 和 HardGame 的父类。

2.2 设计模式应用

2.2.1 单例模式

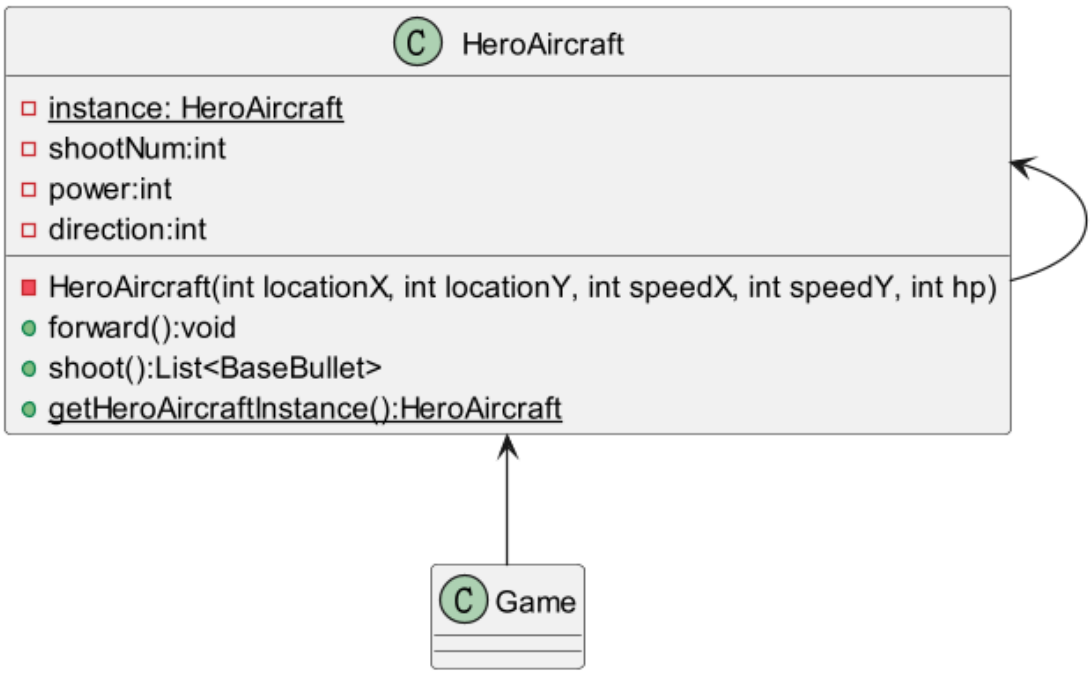
1. 应用场景分析

描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。

创建英雄机时需要用到此模式，并且，英雄机有且仅有一个，一旦被摧毁则游戏结束，因此在游戏中只能实例化一次英雄机。
问题1：目前在Game类中通过new的方式创建英雄机，违反了设计原则中的单一职责原则。
问题2：不能保证英雄机的唯一性，因为外部程序可以用new的方式创造多个实例。
优势：1.保证了英雄机的唯一性，2.遵守了单一职责原则，将对象的创建和使用分离，3.降低了代码的耦合度。

2. 设计模式结构图

结合飞机大战实例，绘制该场景下具体的解决方案（UML 类图）。描述你设计的 UML 类图结构中每个角色的作用，并指出它的关键属性和方法。



根据该 UML 图，每个类的作用如下：

- HeroAircraft：表示英雄飞机，在游戏中扮演玩家操纵的飞机角色，具有向前移动和射击的能力。其中静态成员变量instance用于保存唯一的英雄飞机实例，非静态成员变量`shootNum`

代表发射的子弹数量，power 代表英雄飞机的攻击力，direction 代表英雄飞机的方向，带参构造函数用于创建英雄飞机实例，forward()方法用于使英雄飞机向前移动，shoot()方法用于发射子弹，getHeroAircraftInstance()方法用于获取英雄飞机实例。
- Game：表示游戏主程序，负责初始化游戏场景、管理游戏状态、处理游戏逻辑等。其中包含英雄飞机实例，并与英雄飞机进行交互。

2.2.2 工厂模式

1. 应用场景分析

描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。

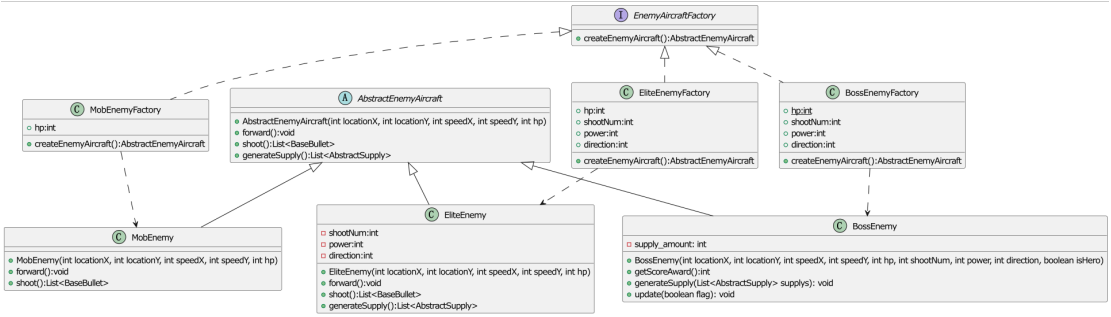
在游戏中，敌机会源源不断从界面上方产生，需要生成 Mob，Elite，Boss（暂未实现）三种敌机，敌机可以被英雄机击毁后消失，敌机也能对英雄机造成伤害。Elite，boss 被击毁的时候有概率掉落 3 种道具中的其中一种，道具和英雄机碰撞时生效。

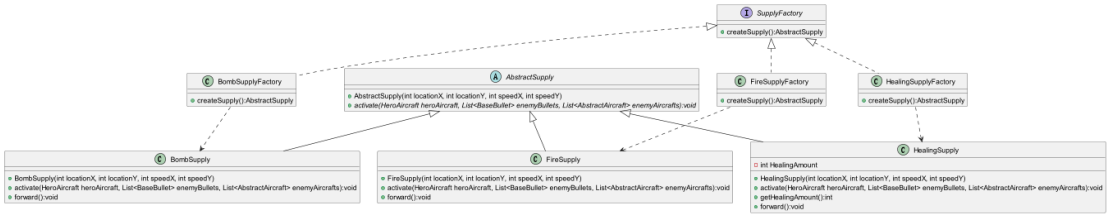
问题 1：没有对 Game 类隐藏生成敌机 or 道具的细节,违反单一职责原则。
问题 2：违反了开闭原则，因为不方便增加新类型的敌机和道具。
问题 3：违反依赖倒转原则，因为创建过程面向具体实现而不是接口，如果想要增删敌机的属性，要在所有的 new 运算中修改。

使用工厂模式的优势：
符合单一职责原则：对 game 类隐藏创建细节，降低耦合度。
符合开闭原则：方便增加新的类型的道具和敌机，只需要增加新的工厂累。
符合依赖倒转原则：创建过程依赖于抽象接口。

2. 设计模式结构图

结合飞机大战实例，绘制该场景下具体的解决方案（UML 类图）。描述你设计的 UML 类图结构中每个角色的作用，并指出它的关键属性和方法。





2.2.3 策略模式

1. 应用场景分析

描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。

场景:

在实现飞机发射子弹的时候需要用到策略模式。

问题

1：目前飞机发射子弹的实现过程分别是在各自的类中，重复代码多，代码难以维护，容易引入 bug。

2：如果想要增加新的一种机型，或者一种新的弹道，需要修改多处代码，违反了开闭原则和合成复用原则。

优势：

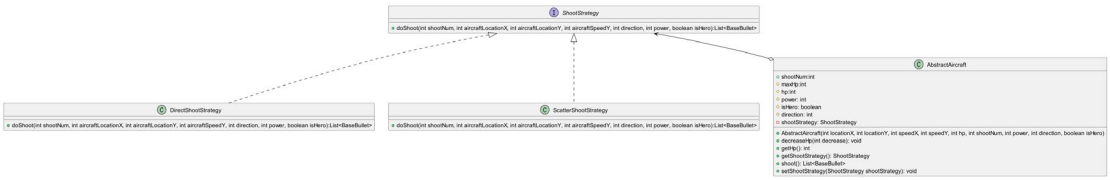
1.策略模式可以把每一个算法封装起来， 放在独立的类中，方便替换以及新的算法的加入。

2.避免多重条件判断，可以改变使用多个 if-else 语句来判断相似代码的情况，使得代码更具有维护性和可读性。

3.提供了一种代替继承的方法，既能代码重复利用， 也比继承更加灵活，可以任意扩展。

2. 设计模式结构图

结合飞机大战实例，绘制该场景下具体的解决方案（UML 类图）。描述你设计的 UML 类图结构中每个角色的作用，并指出它的关键属性和方法。



ShootStrategy 是射击策略接口。

ScatterShootStrategy 和 DirectShootStrategy 分别是散射和直射策略类，都实现了 ShootStrategy 射击策略接口。

AbstractAircraft 类中有一个私有的 ShootStrategy 类型的成员变量，其实现依赖于 ShootStrategy 接口，因此是聚合关系。

其中关键属性和方法如下：

shootStrategy	当前 aircraft 的射击策略

getShootStrategy	Public 的方法，返回当前 aircraft 的射击策略
setShootStrategy	设置当前 aircraft 的射击策略
shoot	当前 aircraft 的射击方法
doshoot	射击策略接口中的实现算法

2.2.4 数据访问对象模式

1. 应用场景分析

描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。

场景:

在每局游戏结束之后，可能需要保存当局游戏的得分,时间,玩家的名称,名次等等。以及玩家可能有查找某项记录，删除某项记录的需求。

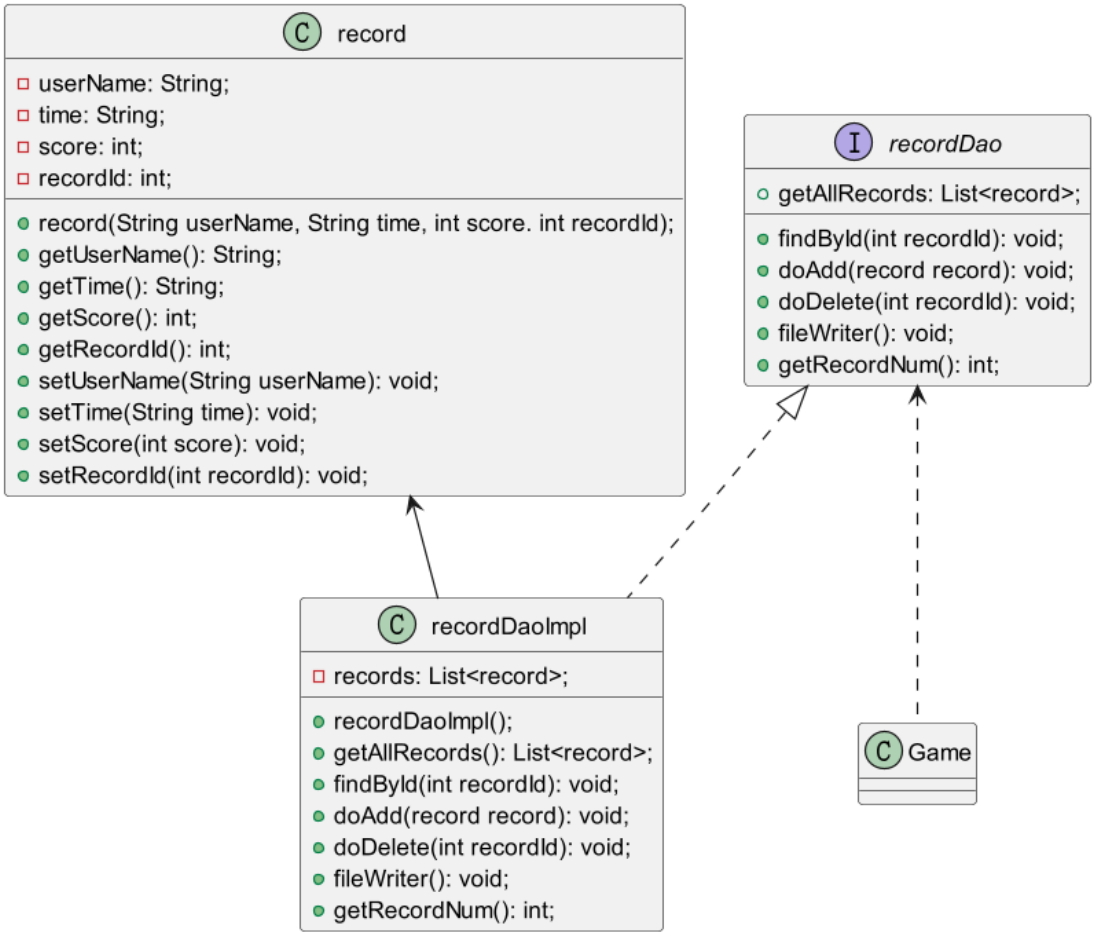
可以通过数据对象访问模式来解决这些问题。

优势：

可以把低级的数据访问 API 或者操作从高级的业务中分离出来，使得访问数据的过程面向抽象，符合依赖倒转原则。同时，也能够降低代码的耦合度，因为新增的 DAO 层可以隔离业务代码和文件、数据库，避免业务代码直接接触数据库文件等等。

2. 设计模式结构图

结合飞机大战实例，绘制该场景下具体的解决方案（UML 类图）。描述你设计的 UML 类图结构中每个角色的作用，并指出它的关键属性和方法。。



2.2.5 观察者模式

1. 应用场景分析

描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。

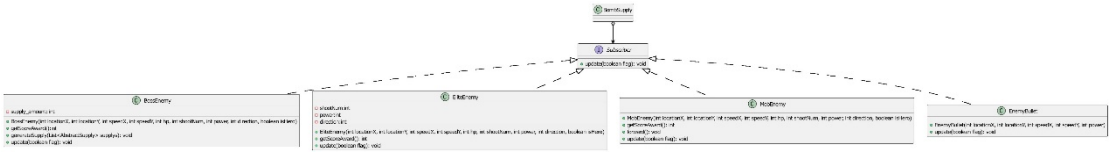
在飞机大战游戏中，观察者模式可以用于敌机的移动、子弹的发射和补给的生成等场景中，当这些状态发生变化时，通知订阅者进行相关的处理。

在实际设计中，使用观察者模式可以避免订阅者与被订阅者之间的紧耦合关系，提高代码的可维护性和可扩展性。例如，在飞机大战游戏中，如果不使用观察者模式，那么每个敌机、子弹和补给对象都需要手动管理其订阅者列表和通知订阅者，这样会导致代码的冗余和复杂度的增加。

使用观察者模式，可以将订阅者和被订阅者解耦，订阅者只需要实现 Subscriber 接口，并注册到相应的被订阅者中即可。被订阅者只需要在状态发生变化时，调用订阅者的 update()方法通知其进行处理即可。这样可以大大简化代码的实现，并提高代码的可读性和可维护性。

2. 设计模式结构图

结合飞机大战实例，绘制该场景下具体的解决方案（UML 类图）。描述你设计的 UML 类图结构中每个角色的作用，并指出它的关键属性和方法。



- Subscriber：表示订阅者接口，用于订阅和更新游戏状态的变化。
- BossEnemy：表示 Boss 敌机，具有较高的血量和攻击力，并且可以生成补给箱。其中带参构造函数用于创建 Boss 敌机实例，getScoreAward()方法用于获取 Boss 敌机被消灭后玩家能够获得的分数，generateSupply()方法用于生成补给箱，update()方法用于接收游戏状态的更新。
- EliteEnemy：表示精英敌机，与普通敌机相比，具有更高的攻击力和速度，但血量较低。其中带参构造函数用于创建精英敌机实例，getScoreAward()方法用于获取精英敌机被消灭后玩家能够获得的分数，update()方法用于接收游戏状态的更新。
- MobEnemy：表示普通敌机，是游戏中最基本的敌机类型，血量、攻击力和速度都较低。其中带参构造函数用于创建普通敌机实例，getScoreAward()方法用于获取普通敌机被消灭后玩家能够获得的分数，forward()方法用于使敌机向前移动，update()方法用于接收游戏状态的更新。
- EnemyBullet：表示敌机子弹，具有一定的攻击力和速度。其中带参构造函数用于创建敌机子弹实例，update()方法用于接收游戏状态的更新。
- BombSupply：表示炸弹补给，可以让玩家在游戏中使用炸弹来消灭敌机。其中包含一个订阅者列表，用于通知订阅者炸弹补给的生成。
- Subscriber：表示订阅者接口，用于订阅和更新游戏状态的变化，包括敌机的移动、子弹的发射和补给的生成等。其中包含一个 update()方法，用于接收游戏状态的更新。

2.2.6 模板模式

1. 应用场景分析

请简单描述你对三种游戏难度是如何设计的，影响游戏难度的因素有哪些。描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。

影响游戏难度的因素可能包括以下几个方面：

1. 敌机的数量和种类：敌机数量和种类的增加会增加游戏的难度

2. 敌机的攻击力和速度：敌机的攻击力和速度越高，游戏难度越大。
3. 玩家的生命值和攻击力：玩家的生命值和攻击力越低，游戏难度越大。
4. 敌机产生概率。
5. boss 机产生阈值
6. 敌机射击周期
7. 敌机数量最大值

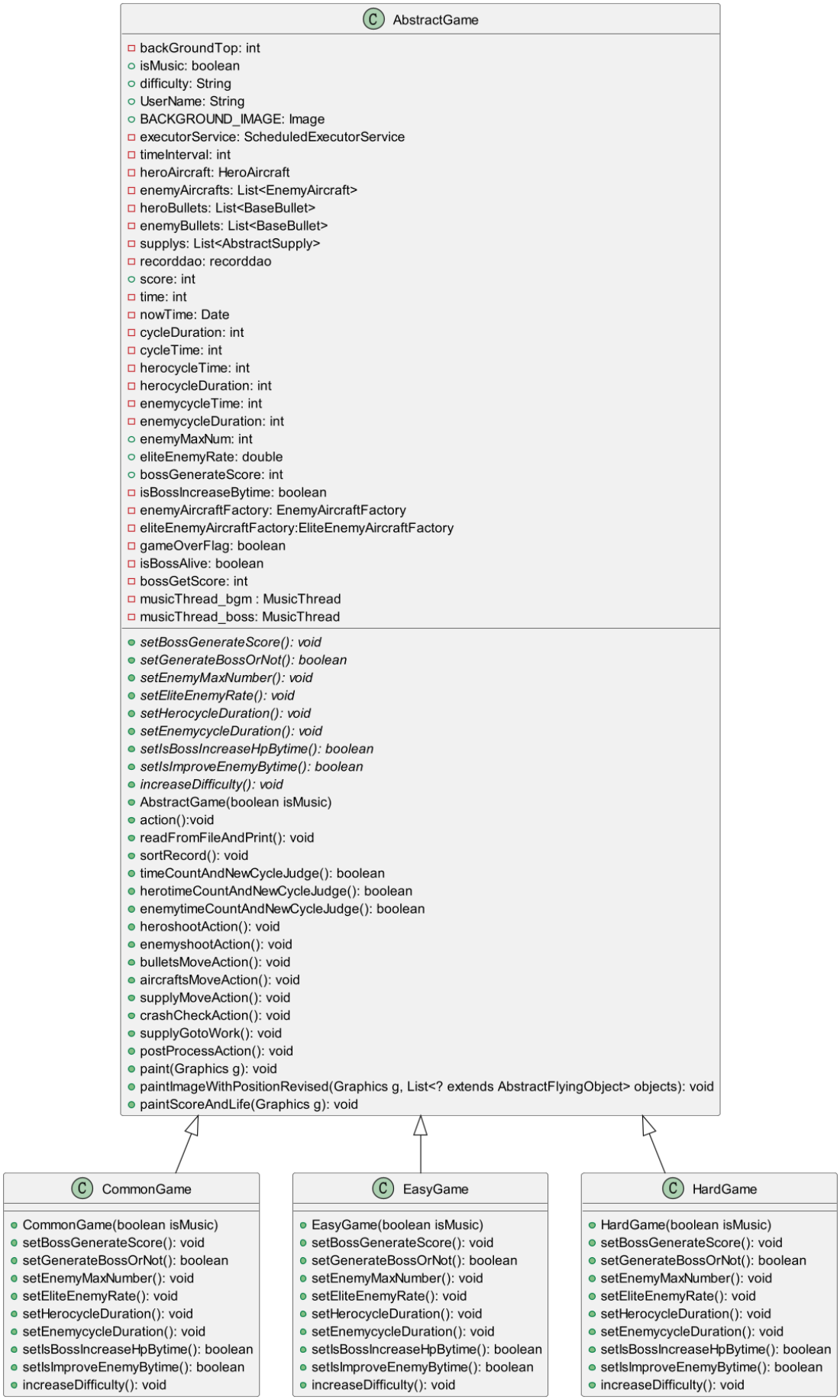
在飞机大战游戏中，观察者模式可以用于敌机的移动、子弹的发射和补给的生成等场景中，当这些状态发生变化时，通知订阅者进行相关的处理。这样可以实现游戏状态的及时更新，并保证游戏的流畅性和可玩性。

在使用模板模式对三种游戏难度进行设计时，可以将相同的部分抽象出来，放到模板方法中实现，而将不同的部分放到具体子类中实现。这样可以避免代码的重复，并且方便后续的维护和扩展。

使用模板模式可以提高代码的复用性和可维护性，同时也可以降低代码的复杂度。模板模式可以将相同的代码抽象出来，放到基类中实现，而将不同的代码放到派生类中实现。这样可以大大简化代码的实现，并提高代码的可读性和可维护性。

2. 设计模式结构图

结合飞机大战实例，绘制该场景下具体的解决方案（UML 类图）。描述你设计的 UML 类图结构中每个角色的作用，并指出它的关键属性和方法。


AbstractGame
backGroundTop: int
isMusic: boolean
difficulty: String
UserName: String
BACKGROUND_IMAGE: Image
executorService: ScheduledExecutorService
timeInterval: int
heroAircraft: HeroAircraft
enemyAircrafts: List<EnemyAircraft>
heroBullets: List<BaseBullet>
enemyBullets: List<BaseBullet>
supplys: List<AbstractSupply>
recorddao: recorddao
score: int
time: int
nowTime: Date
cycleDuration: int
cycleTime: int
herocycleTime: int
herocycleDuration: int
enemycycleTime: int
enemycycleDuration: int
enemyMaxNum: int
eliteEnemyRate: double
bossGenerateScore: int
isBossIncreaseBytime: boolean
enemyAircraftFactory: EnemyAircraftFactory
eliteEnemyAircraftFactory:EliteEnemyAircraftFactory
gameOverFlag: boolean
isBossAlive: boolean
bossGetScore: int
musicThread_bgm : MusicThread
musicThread_boss: MusicThread
setBossGenerateScore(): void
setGenerateBossOrNot(): boolean
setEnemyMaxNumber(): void
setEliteEnemyRate(): void
setHerocycleDuration(): void
setEnemycycleDuration(): void
setIsBossIncreaseHpBytime(): boolean
setIsImproveEnemyBytime(): boolean
increaseDifficulty(): void
AbstractGame(boolean isMusic)
action():void
readFromFileAndPrint(): void
sortRecord(): void
timeCountAndNewCycleJudge(): boolean
herotimeCountAndNewCycleJudge(): boolean
enemytimeCountAndNewCycleJudge(): boolean
heroshootAction(): void
enemyshootAction(): void
bulletsMoveAction(): void
aircraftsMoveAction(): void
supplyMoveAction(): void
crashCheckAction(): void
supplyGotoWork(): void
postProcessAction(): void
paint(Graphics g): void
paintImageWithPositionRevised(Graphics g, List<? extends AbstractFlyingObject> objects): void
paintScoreAndLife(Graphics g): void
CommonGame
CommonGame(boolean isMusic)
setBossGenerateScore(): void
setGenerateBossOrNot(): boolean
setEnemyMaxNumber(): void
setEliteEnemyRate(): void
setHerocycleDuration(): void
setEnemycycleDuration(): void
setIsBossIncreaseHpBytime(): boolean
setIsImproveEnemyBytime(): boolean
increaseDifficulty(): void
EasyGame
EasyGame(boolean isMusic)
setBossGenerateScore(): void
setGenerateBossOrNot(): boolean
setEnemyMaxNumber(): void
setEliteEnemyRate(): void
setHerocycleDuration(): void
setEnemycycleDuration(): void
setIsBossIncreaseHpBytime(): boolean
setIsImproveEnemyBytime(): boolean
increaseDifficulty(): void
HardGame
HardGame(boolean isMusic)
setBossGenerateScore(): void
setGenerateBossOrNot(): boolean
setEnemyMaxNumber(): void
setEliteEnemyRate(): void
setHerocycleDuration(): void
setEnemycycleDuration(): void
setIsBossIncreaseHpBytime(): boolean
setIsImproveEnemyBytime(): boolean
increaseDifficulty(): void

3 收获和反思

请填写本次实验的收获，记录实验过程中出现的值得反思的问题及你的思考。
欢迎为本课程实验提出宝贵意见！

收获:
借助飞机大战游戏的实现巩固丰富了 Java 编程技能，加深了对”面向对象”的理解。

实验中出现的问题：
实验五实现排行榜的时候，选择通过 txt 文件存取数据，因为对 java 中文件读写的操作不熟悉，花费了较多时间。
实验五实现界面跳转的时候，一开始没有想到解决“游戏结束才跳转而不是一开始就跳转”，花费了很多时间 Debug 为什么一开始游戏就跳转页面，后来才想到用 MAIN_LOCK，wait(), notify()的解决方法

建议:
可以把实验中出现的各个关键知识点对应地制作独立的 Demo，让学生了解一下大致的写法但不用太仔细。
可以把实验课和理论课相结合，即在理论课刚开课就可以短暂地穿插实验课，这样可以马上利用刚刚在理论课上学的知识点。