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

实验报告

开课学期：2024 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：

学生班级：22 级 5 班

学生学号：220110519

学生姓名：邢瑞龙

评阅教师：

报告成绩：

实验与创新实践教育中心制

2024 年 5 月

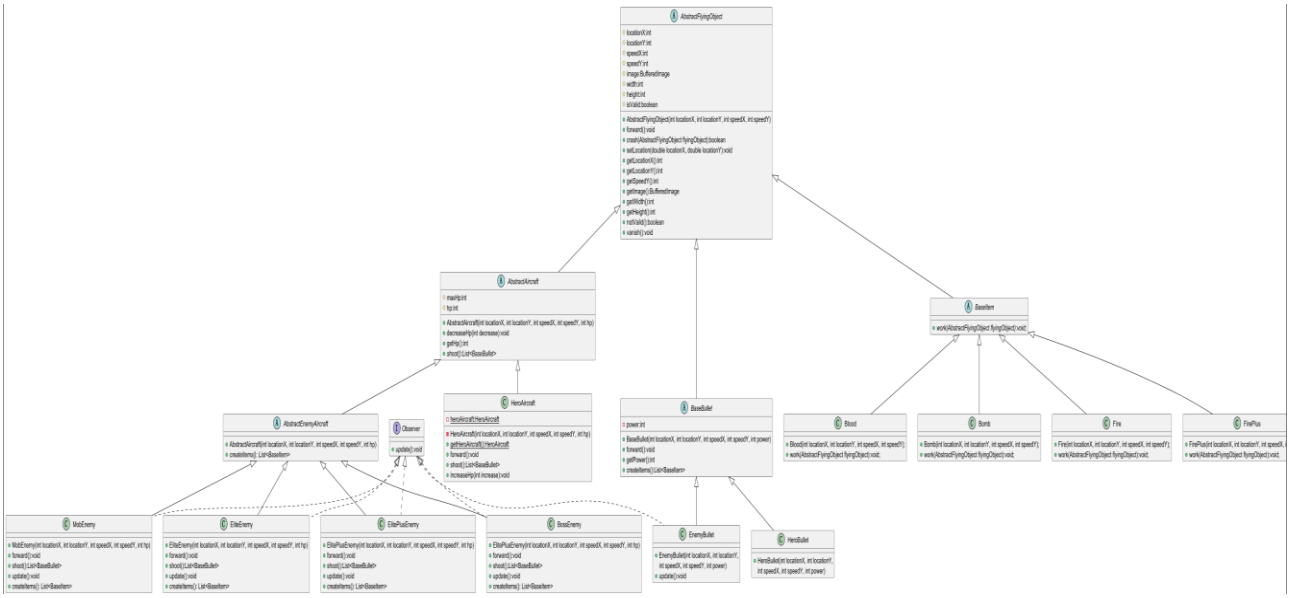
1 实验环境

操作系统：Windows11
使用 IDEA：IntellijIDEA

2 实验过程

以下 UML 结构图请更新为最终提交版本。

2.1 类的继承关系



2.2 设计模式应用

2.2.1 单例模式

1. 应用场景分析

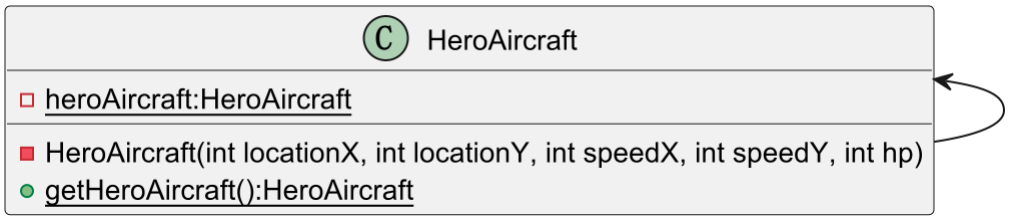
因为游戏中英雄机只有一个且不可再生，因此飞机大战中的英雄机需要用到单例模式。

目前代码中英雄机的构造函数是 public 的，意味着我可以在程序的任意地方创建多个英雄机，这存在安全隐患。

单例模式的优势在于:

1. 保证类在整个程序中仅有一个实例化对象，避免了误创建多个对象，提高了代码的稳定性和一致性
2. 简化访问：由于单例模式提供了全局访问点，因此在整个应用程序中可以方便地访问单例对象。

2. 设计模式结构图



本类是英雄机类的单例模式，是抽象类 AbstractAircraft 的子类。

（1）属性：

private static 的单例对象 HeroAircraft heroAircraft，用于保存英雄机唯一实例的引用。private 保证其他类不可随意访问修改，static 保证不需要创建实例化就能得到

（2）方法：

public static getHeroAircraft()，用于得到英雄机的唯一实例

同时实现使用了双重检查锁避免了多个线程同时创建对象实例的问题且提升了性能。

2.2.2 工厂模式

1. 应用场景分析

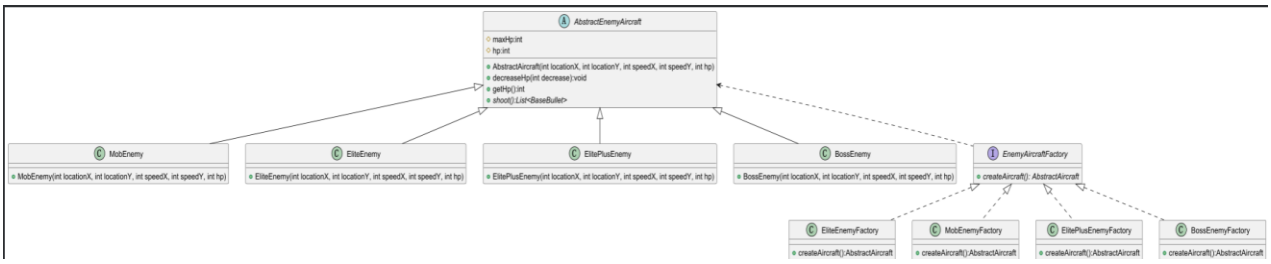
游戏中会有各式各样的敌机和道具，它们具有不同的外观、速度、生命值等属性。目前的代码是直接在使用时通过 new 关键字直接创建对象，对象的创建和使用没有分开。如果在多个地方都使用了类似的代码来实例化敌机、道具，会导致代码重复，增加了维护成本。

工厂模式的优势：

1. 解耦合：工厂模式将对象的创建和使用分离开来，只需要通过工厂类来获取对象实例，不需要关心具体的实现细节，从而降低了类之间的耦合度，保证了单一职责原则
2. 可维护性高：如果需要修改对象的创建逻辑或者添加新的对象类型，只需要创建新的工厂类即可，原来的代码不必更改。保证了开闭原则

2. 设计模式结构图

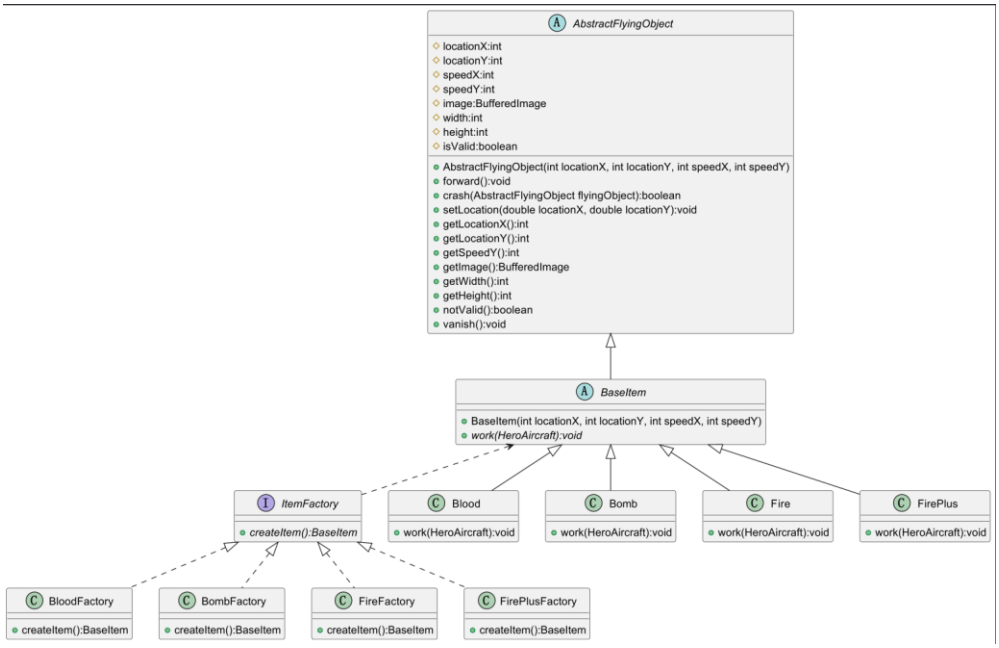
（1）敌机类工厂：



产品抽象类：AbstractEnemyAircraft
具体产品:MobEnemy,EliteEnemy,ElitePlusEnemy,BossEnemy
创建者接口：EnemyAircraftFactory
具体创建者:

- MobEnemyFactory 依赖于具体产品 MobEnemy
- EliteEnemyFactory 依赖于具体产品 EliteEnemy
- ElitePlusEnemyFactory 依赖于具体产品 ElitePlusEnemy
- BossEnemyFactory 依赖于具体产品 BossEnemy
  其对应的 createAircraft()方法返回创建对应的敌机种类实例

（2）道具工厂:



产品抽象类：BaseItem 是 AbstractFlyingObject 的子类
具体产品: Blood,Bomb,Fire,FirePlus
创建者接口: ItemFactory
具体创建者：

- BloodFactory 依赖于具体产品 Blood
- BombFactory 依赖于具体产品 Bomb
- FireFactory 依赖于具体产品 Fire
- FirePlusFactory 依赖于具体产品 FirePlus

BaseItem 的 work 方法为道具的奏效抽象方法
不同具体创建者对应的 createItem()方法返回创建对应的道具种类实例

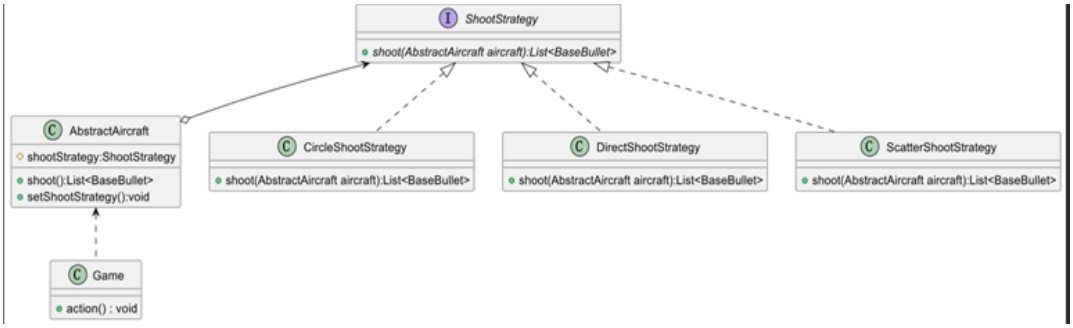
2.2.3 策略模式

1. 应用场景分析

飞机大战中每个飞机都存在射击的方法，而射击有三种不同的模式，直射，环射，散射，使用策略模式可以将主飞机和射击的策略分离开来，减少耦合，封装变化。且未来如果想要增添新的射击模式，直接添加即可，原来的代码不用做更改，易于维护，符合开闭原则。

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。



在此场景下，game 为客户端，每个飞机为 context，其都继承了 AbstractAircraft，而飞机的 shoot 方法就需要用到策略模式

（1）客户端：

game 类在 action 方法中会调用每个飞机的 shoot 方法

(2)上下文:

AbstractAircraft 类中：

1. protected shootStrategy 属性保存了单个对具体射击策略模式的实例化，
2. setShootStrategy()方法用于后续射击模式的更改
3. shoot 方法调用 shootStrategy 中的 shoot 方法来射击

(3) 策略：

策略接口 ShootStrategy：声明所有射击模式的通用策略
抽象方法 shoot(AbstractAircraft aircraft):List<BaseBullet>是射击的策略接口中的抽象方法

- 具体策略 CircleShootStrategy：
  shoot(AbstractAircraft aircraft):List<BaseBullet>中实现环射
- 具体策略 DirectShootStrategy：
  shoot(AbstractAircraft aircraft):List<BaseBullet>中实现直射
- 具体策略 ScatterShootStrategy：
  shoot(AbstractAircraft aircraft):List<BaseBullet>中实现散射

2.2.4 数据访问对象模式

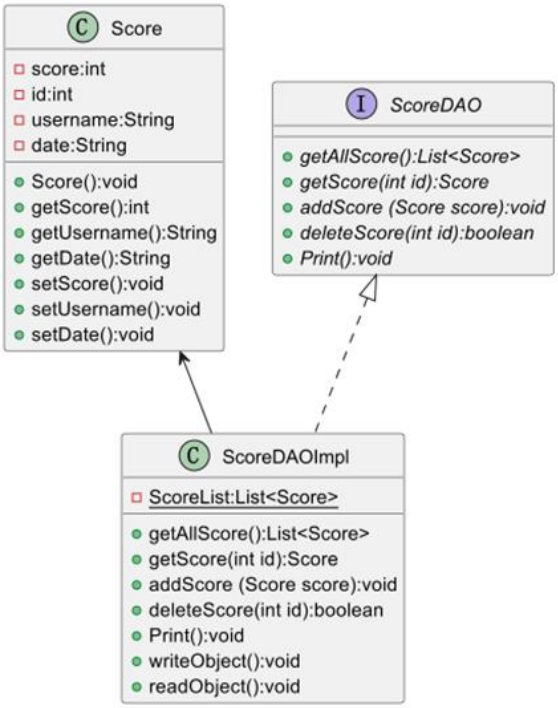
1. 应用场景分析

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。

飞机大战中的排行榜功能中，需要读取历史玩家的分数数据，这就需要用到 DAO 模式，将低级的数据访问操作从高级的业务服务中分离出来。这样用户如果想要对数据进行操作，只需要使用 DAO 中已经封装好的方法，并且如果想要更换数据库，直接新写一个 DAO 的子类即可，减少了服务层和数据存储设备之间的耦合度

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。。



(一)数据库的数值对象类 Score:

属性：

private int id：记录对应的 ID
private int score:记录对应的分数
private String userName:记录对应的用户名
private String date:记录对应的日期,

方法：

为以上属性的 get 和 set

(二)数据对象访问接口 ScoreDAO：

List<Score> getAllScore:获取历史的所有分数记录数据
public int getScore(int id)：通过 id 获取单条数据
public void addScore(Score score):向数据库中增添一条数据
public void deleteScore(int id)：通过 id 向数据库中增添一条数据
public void Print():打印排行榜

（三）数据对象访问实体类 ScoreDAOImpl：

List<Score> getAllScore:获取历史的所有分数记录数据，并赋值给 ScoreList
public int getScore(int id)：通过 id 获取单条数据

public void addScore(Score score):向数据库中增添一条数据
public void deleteScore(int id)：通过 id 向数据库中增添一条数据
public void Print():打印排行榜
public void writeObject()：将数据写入文件
public void readObject()：从文件中读取数据

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。

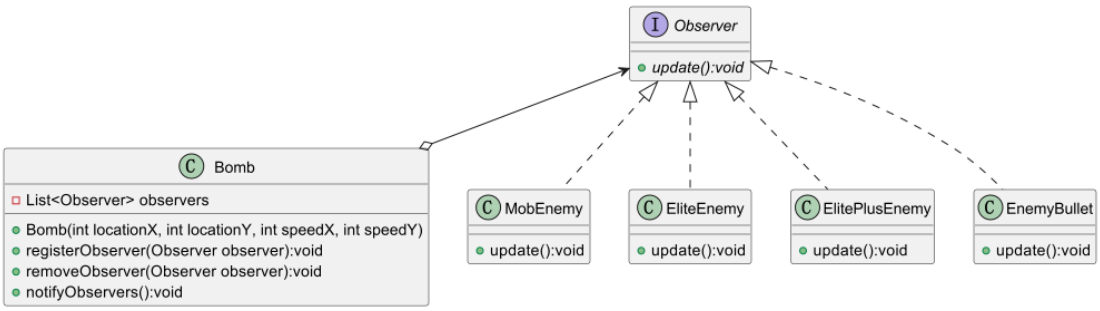
在飞机大战中，当炸弹道具生效时，界面上存在的所有普通，精英，超级精英敌机和敌机子弹都需要清空，这种多对一的依赖关系很适合使用观察者模式。当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。

观察者模式优点：

a) 实现表示层和数据逻辑层的分析。
b) 在观察者和观察目标之间建立了一个抽象的耦合
c) 支持广播通信，实现一对多系统的设计维度
d) 符合开闭原则。

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。



（一）观察目标 Bomb:

属性：
private final List<Observer> observers:观察者列表
方法：
public void registerObserver(Observer observer)：添加观察者
public void removeObserver(Observer observer)：删除观察者
public void notifyObservers()：通知全部观察者

（二）观察者接口：Observer

方法：

public void update()：对观察目标的通知做出更新

而炸弹对所有所有普通，精英，超级精英敌机和敌机子弹都有作用效果，所以将这些类都实现 Observer 接口

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。

在飞机大战中，不同难度对应的游戏流程整体一样，只是其中一些参数和细节方法不同，适合用模板模式来实现。

优势：将不变的部分直接继承给子类，而一些需要变化的行为延迟到子类来实现，可实现代码复用和增强程序的扩展性。

游戏难度设置：

因素\模式	简单	中等	困难
最大敌机数量	5	7	8
难度增加周期	无	10s	6s
刷新周期	800ms	720ms且随时间减少最低400ms	600ms且随时间减少最低200ms
普通，精英，超级精英敌机血量	不变	不变	随时间增加，最大增长为初始的1.5倍
Boss机血量	无Boss机	600	600 + 遇见Boss次数+120（上限为2000）
精英敌机刷新概率	50%并保持不变	50%并随时间增长最大为65%	50%并随时间增长最大为75%
奖励：当支撑到最高难度后飞机基础活力增加	无	当难度变为最大时飞机基础子弹变为2颗	当难度变为最大时飞机基础子弹变为2颗

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。

（一）抽象 Game 类：

游戏的逻辑流程：

```
Runnable task = () -> {
    time += timeInterval;
    // 周期性执行（控制频率）
    if (timeCountAndNewCycleJudge()) {
        System.out.println(time);
         //新敌机产生
        createEnemyAircraft();
        //产生boss机
        createBossAircraft();
        // 飞机射出子弹
        shootAction();
    }
    //随时间的推移增加难度
    if(diffTimeCountAndChangeJudge()){
        increaseDifficulty();
    }
    // 子弹移动
    bulletsMoveAction();
    // 飞机移动
    aircraftsMoveAction();
    // 撞击检测
    crashCheckAction();
    // 后处理
    postProcessAction();
    //每个时刻重绘界面
    repaint();
    // 游戏结束检查英雄机是否存活
   checkIfGameOver();
};
```

属性:

难度的不同参数在子类的构造函数中修改

方法:

只有 increaseDifficulty 和 createBossAircraft 两个行为在不同难度的游戏中逻辑流程不同，因此将此两个行为作为抽象方法延迟到子类实现。

（二）具体 Game 类：

（1）EasyGame：

increaseDifficulty()和 createBossAircraft()都为空。

（2）MediumGame 类：

1. increaseDifficulty：增加产生精英敌机概率，敌机血量和减少刷新周期

```
    protected void increaseDifficulty() {
        boolean tag = false;
        if(ratioOfEliteEnemy < maxRatioOfEliteEnemy){
            ratioOfEliteEnemy += 0.02;
            tag = true;
        }
        if(cycleDuration < minCycleTime){
            cycleDuration -= 10;
            tag = true;
        }
        if(tag) {
            System.out.printf("已提高难度，目前精英机出现概率为 %f, 敌机产生周期：%d \n",ratioOfEliteEnemy,cycleDuration);
        }
        else{
            HeroAircraft.setHero_basicFire(2);
            System.out.printf("已达最高难度\n");
        }

    }
}
```

2. createBossAircraft:根据分数阈值产生 Boss 机，每次 Boss 机血量不上升

```java
@Override
protected void createBossAircraft() {
    if(score / bossScoreThreshold >= cntOfMeetingBoss && !bossFlag){
        EnemyAircraftFactory factory;
        AbstractEnemyAircraft enemyAircraft;
        factory = new BossEnemyFactory();
        enemyAircraft = factory.createAircraft();
        enemyAircrafts.add(enemyAircraft);
        cntOfMeetingBoss += 1;
        bossFlag = true;
    }
}
```

（3）HardGame 类：

1. increaseDifficulty：增加产生精英敌机概率，敌机血量和减少刷新周期

```java
protected void increaseDifficulty() {
    boolean tag = false;
    tag |= MobEnemyFactory.increaseRate();
    tag |= EliteEnemyFactory.increaseRate();
    tag |= ElitePlusEnemyFactory.increaseRate();
    if(ratioOfEliteEnemy < maxRatioOfEliteEnemy){
        ratioOfEliteEnemy += 0.02;
        tag = true;
    }
    if(cycleDuration < minCycleTime){
        cycleDuration -= 10;
        tag = true;
    }
    if(tag) {
        System.out.printf("已提高难度，目前精英机出现概率为 %f，敌机产生周期：%d \n",ratioOfEliteEnemy,cycleDuration);
    }
    else{
        HeroAircraft.setHero_basicFire(2);
        System.out.printf("已达最高难度\n");
    }

}
```

2. createBossAircraft:根据分数阈值产生 Boss 机，每次 Boss 机血量增加

```java
@Override
protected void createBossAircraft() {
    if(score / bossScoreThreshold >= cntOfMeetingBoss && !bossFlag){
        EnemyAircraftFactory factory;
        AbstractEnemyAircraft enemyAircraft;
        factory = new BossEnemyFactory();
        enemyAircraft = factory.createAircraft();
        enemyAircrafts.add(enemyAircraft);
        cntOfMeetingBoss += 1;
        bossFlag = true;
        //困难模式增加boss机血量
        BossEnemyFactory.increaseBossHp();
    }

}
```

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！

在本次飞机大战实验中，我学习并实践了面向对象思想，领悟到了封装，继承，多态的强大和便捷。同时我还学习了单例模式，工厂模式，策略模式，数据库访问对象模式，观察者模式，模板模式等等诸多设计模式思想并在该项目中灵活运用，虽然过程中遇到诸多困难，但我也很享受这个过程。希望自己在日后的学习和工作中也能写一份“好的”代码，谢谢老师提供了这么好的一门课程！