

（深圳）

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： 飞机大战游戏的设计与实现

实验性质： 设计型

实验学时： 16 地点： T2 608

学生班级： 计科8班

学生学号： 210110820

学生姓名： 余圣源

评阅教师：

报告成绩：

实验与创新实践教育中心制

2023年4月

# 实验环境

*请填写实验所用到的操作系统和主要开发工具。*

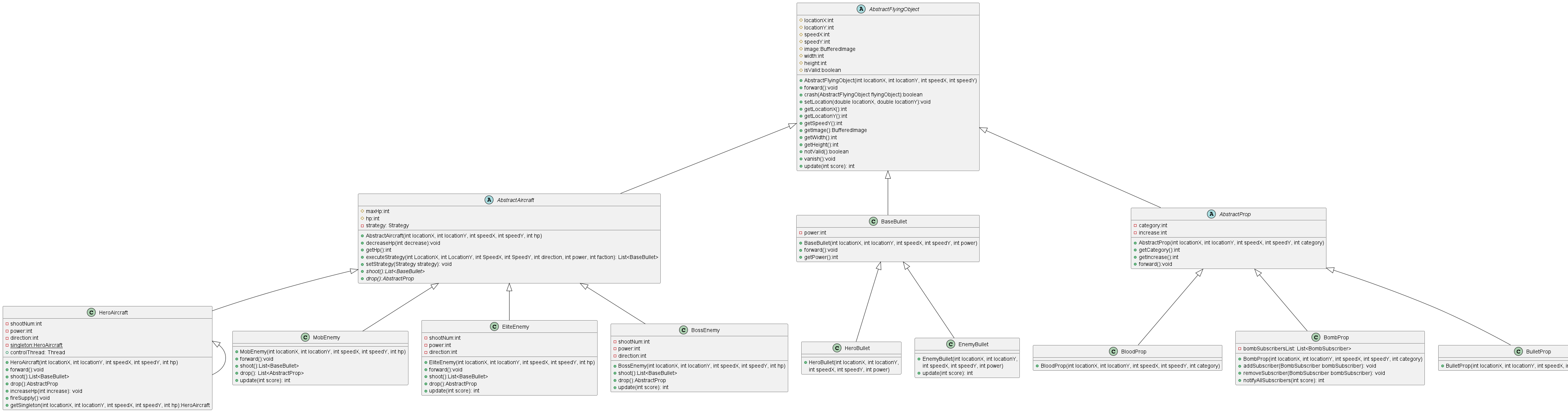
本次实验用到的操作系统是windows，用到的开发环境（IDE）是IntelliJ IDEA Community Edition 2022.3.2

# 实验过程

## 类的继承关系分析

*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。*

根据面向对象的设计原则，所有飞机类、道具类和子弹类都实现了“抽象飞行物”（AbstractFlyingObject）之下，因为它们的共同属性都是能够运动。更详细地说，其中有“抽象飞机”（AbastractAircrafts）、“抽象道具”（AbstractProps）、“抽象子弹”（AbstractBullets）实现了“抽象飞行物”（AbstractFlyingObject）抽象类。而每一个抽象类下面又继承了不同的子类



## 设计模式应用

### 2.3.1单例模式

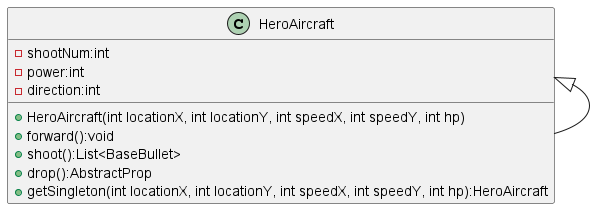
1. 应用场景分析

*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。*

飞机大战中创建英雄机实例的时候需要用到单例模式。因为在此游戏的逻辑之中，只允许存在一个英雄机，且一局游戏之中也不会重复创建英雄机。此时我们可能创建多个英雄机对象，没有保障只能存在一架英雄机。因此需要通过单例模式来限制创建英雄机实例的数量。

1. 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。*

**

单例模式UML类图中只有HeroAircraft一个角色。HeroAircraft是英雄机类，其中最重要的方法是getSingleton()方法，此方法通过双重检查锁定方法使得每次调用此方法实例化对象时都会先检查一次是否已经存在实例，若不存在实例则产生新的对象，反之无效。

### 2.3.2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。*

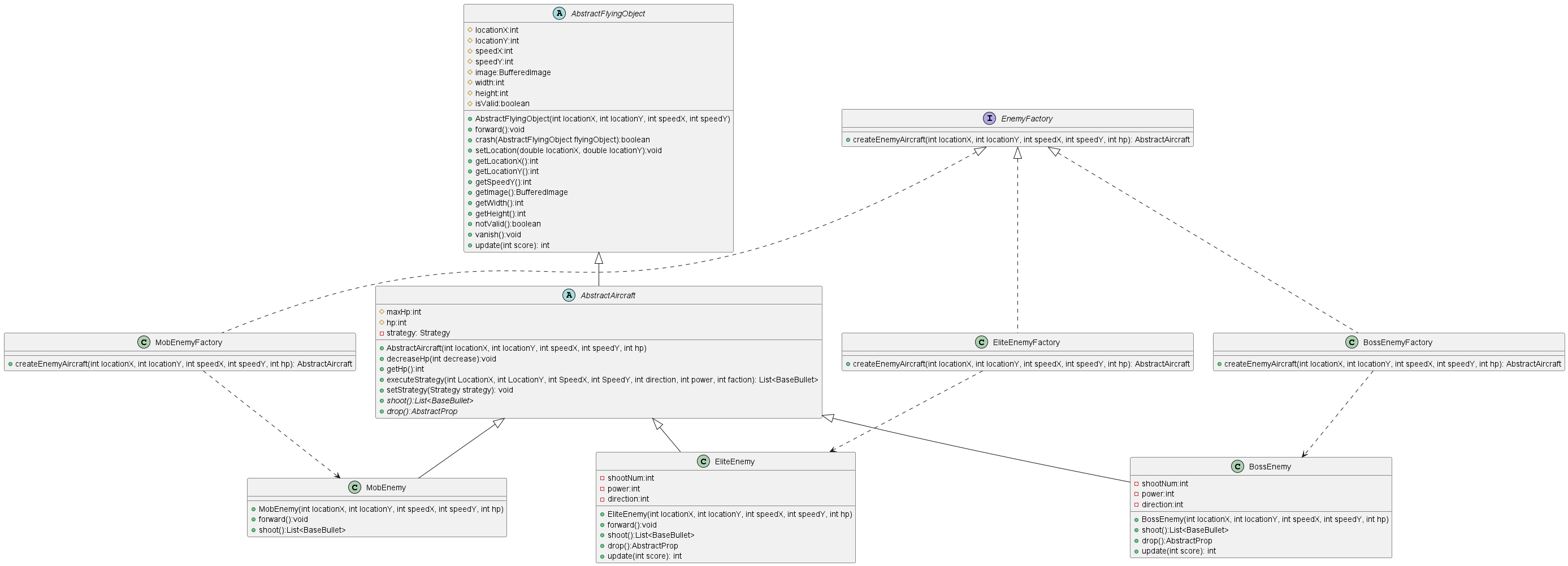
工厂模式用于游戏中创建两种敌机和道具的场景。倘若我们在写好代码之后需要添加一种新的道具或者敌机，按照以前的想法，我们需要再向抽象层中加入一些属性和方法，之后才能拓展继承下来。

运用工厂模式之后，我们可以直接增加新的工厂和新的道具或者敌机，满足了“对修改关闭，对拓展开放”的开闭原则。解除了Game类与具体飞机、道具类之间的耦合；提高了设计规范性和效率。

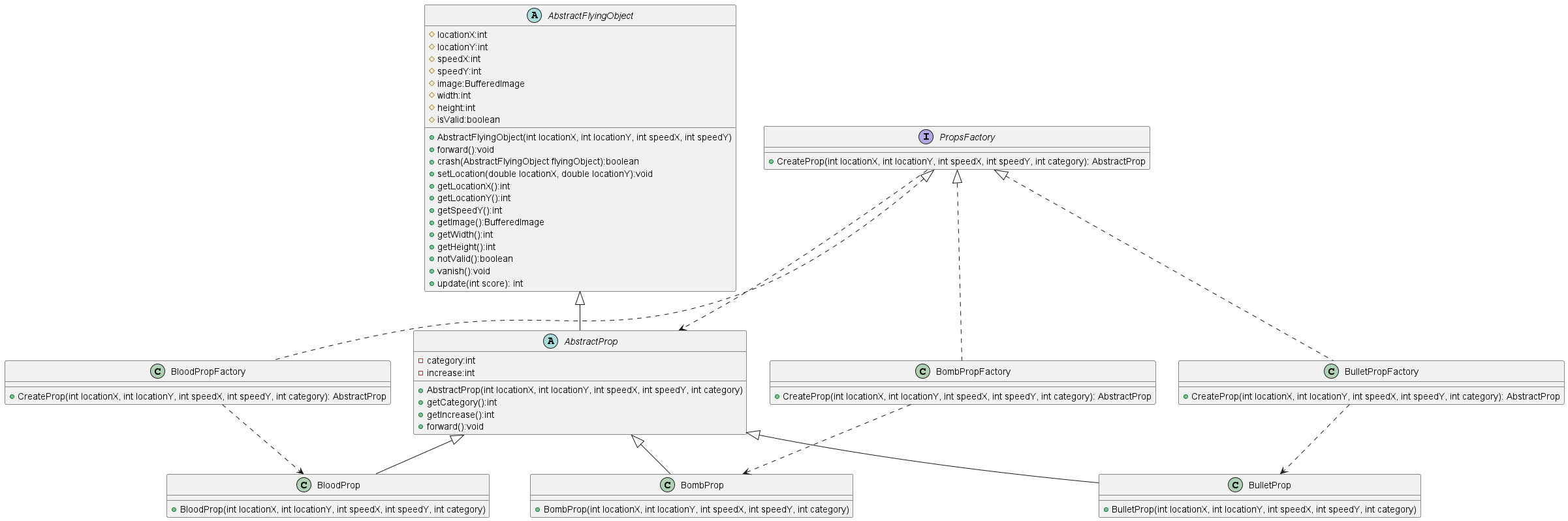
1. 设计模式结构图

*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。*

敌机工厂：



道具工厂：



创建道具和敌机的工厂模式较为相似，此处以道具工厂模式为例解释。

首先创建了一个interface “PropsFactory”作为各种道具工厂的依赖项，其中唯一的一个抽象方法是“CreateProp”，在抽象工厂的各种实现工厂之中，此方法用于创建并且返回一个对应的道具实例。

道具方面与原来基本一致，都是依靠抽象道具层引出各种道具类。抽象道具类中的“getCategory”方法是用于返回此道具的种类。

### 2.3.3策略模式

1. 应用场景分析

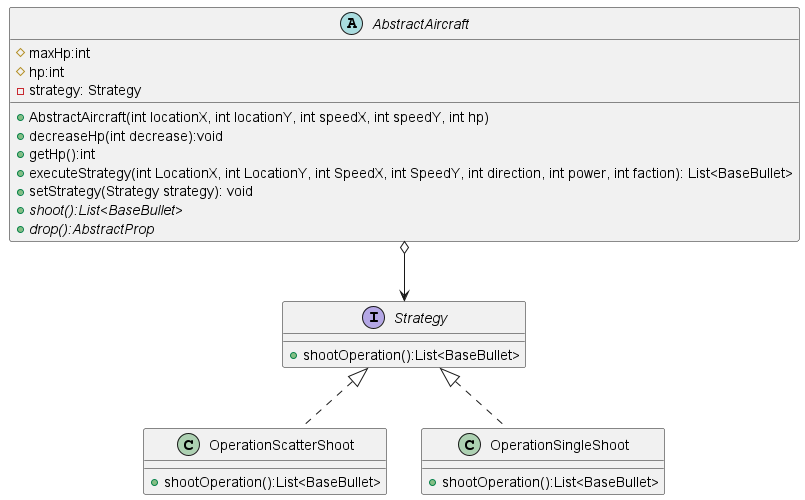
*描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

飞机大战中存在两种射击状态——单射状态和散射状态。在常规状态下，英雄机和精英敌机都是单射状态，而boss敌机是散射模式。当英雄机获得火力道具时，英雄机也变为散射模式，过一段时间还原。多种情况之中，shoot方法差别不大，主要是射出子弹数量的区别，倘若用多重if语句和实现多种shoot方法，会导致代码可复用性低，累赘、可读性差。因此我们使用策略模式。

策略模式中，程序结构更加灵活，具有更好的可维护性和可扩展性。同时，策略模式中的算法地位是平等的，因此可以相互替代，彼此之间没有耦合、依赖。

1. 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。*



AbstractAircraft新增方法：

|  |  |
| --- | --- |
| executeStrategy(int LocationX, int LocationY, int SpeedX, int SpeedY, int direction, int power, int faction) | 调用策略类中的射击方法 |
| setStrategy(Strategy strategy) | 设置某一实例的策略对象 |

*Strategy接口：*

|  |  |
| --- | --- |
| shootOperation() | strategy接口中的抽象方法，用于各策略子类的射击方法实现 |

*OperationScatterShoot类：*

|  |  |
| --- | --- |
| shootOperation() | 接口方法的重写，实现子弹散射的具体功能 |

*OperationSingleShoot类：*

|  |  |
| --- | --- |
| shootOperation() | 接口方法的重写，实现子弹单射的具体功能 |

### 2.3.4数据访问对象模式

1. 应用场景分析

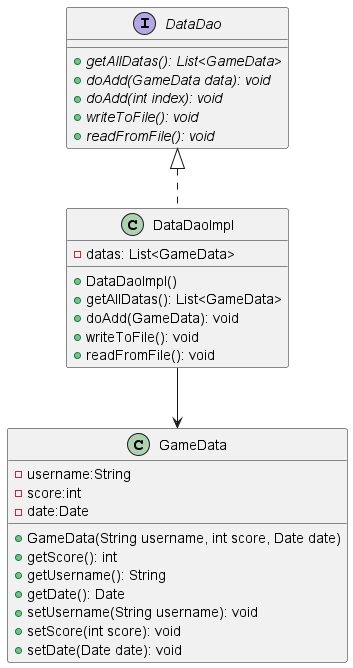
*描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

在飞机大战游戏之中，我们需要记录玩家信息以及得分情况，并在游戏结束时输出得分总榜。在之前的代码中，我们不能存储保存每一次的游戏数据，也没有一套对数据的操作方法和系统，因此为了更好的记录、存储、管理玩家数据，我们需要一整套数据访问对象模式。

Dao在应用之中提供了更好的解耦，将逻辑操作层与持久层访问方法分离，使得逻辑操作层无序关注底层数据库访问的实现。在飞机大战中，规范了用户数据的一些列操作，提供了数据操作方法。所有的数据访问集中在独立的一层， 这层独立的DAO就将数据访问的实现与系统的其余部分剥离，使得系统更具可维护性、可复用性。

1. 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。。*



*DataDao接口：*

|  |  |  |
| --- | --- | --- |
| getAllDatas() | List<GameData> | 抽象方法，从数据库中返回所有数据提供给操作者 |
| doAdd（） | Void | 抽象方法，向数据库中插入新的游戏数据 |
| writeToFile() | Void | 抽象方法，将游戏数据写入文件中保存 |
| readFromFIle | void | 抽象方法，从文件中读出数据进行操作 |

*DataDaoImpl类：*

|  |  |  |
| --- | --- | --- |
| getAllDatas() | List<GameData> | 从数据库中返回所有数据提供给操作者 |
| doAdd（） | Void | 通过插入排序按照降序向数据库中插入新的游戏数据 |
| writeToFile() | Void | 将游戏数据写入文件中保存 |
| readFromFIle（） | void | 从文件中读出数据进行操作 |
| datas | List<GameData> | 保存从文件中读出的所有游戏数据，以及即将要写入，相当于一个链表数据库 |

GameData类：

|  |  |  |
| --- | --- | --- |
| username | String | 私有成员，此条游戏数据对应的用户名 |
| score | Int | 私有成员，此条游戏数据对应的总得分 |
| date | Date | 私有成员，此条游戏数据对应的时间 |
| GameData(String username, int score, Date date) |  | 创建方法 |
| getScore() | Int | 返回私有变量score的值 |
| getUsername() | String | 返回私有变量username的值 |
| getDate | Date | 返回私有变量date的值 |
| setUsername(String username) | void | 将私有变量username重新赋值 |
| setScore(int score) | Void | 给私有变量score赋值 |
| setDate(Date date) | void | 给私有变量date赋值 |

### 2.3.5观察者模式

1. 应用场景分析

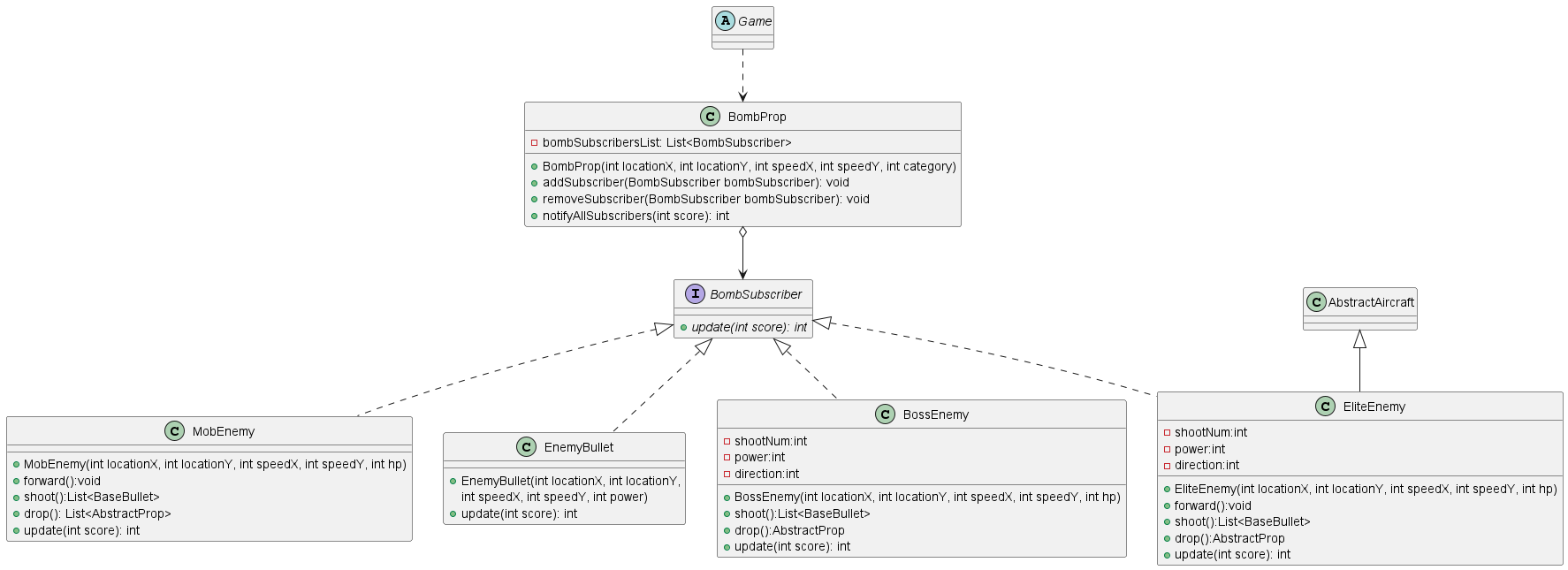
*描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

在先前的设计模式中，我们的“炸弹爆炸”的生效只是在Game类中实现简单的输出。倘若我们仍然在Game类中实现炸弹爆炸效果，此时若要生成一种新的对炸弹爆炸有反应的属性，就要在原来的语句中添加，违反开闭原则，因此我们引入观察者模式。

观察者模式可以实现表示层与数据逻辑层的分离；在观察目标和观察者之间建立了一个抽象的耦合；支持广播通信，简化了一对多系统设计的难度，在；符合开闭原则，增加新的具体观察者无须修改原有的系统代码，在具体观察者与观察目标之间不存在关联关系的情况下，增加新的观察目标也很方便。在本次实验中，用观察者模式便于建立一个炸弹道具对于多目标的系统，使得炸弹生效对象便于修改与管理。

1. 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。*



*BombSubscriber接口：*

|  |  |  |
| --- | --- | --- |
| update(int score) | int | 抽象方法，是观察者模式中观察对象改变时观察者们的对应反馈。其实现类会实现该方法，大致内容是vanish该观察者，并返回对应的分数 |

*BombProp类：*

|  |  |  |
| --- | --- | --- |
| addSubscriber(BombSubscriber bombSubscriber) | List<GameData> | 向记录表中添加观察者 |
| removeSubscriber(BombSubscriber bombSubscriber) | Void | 炸弹生效后清除所有记录表中的观察者 |
| notifyAllSubscribers(int score) | Void | 提醒记录表中的所有观察者update（） |
| BombSubscriberLists | List<BombSubscriber> | 一个订阅者（观察者）记录表，每当炸弹道具生效时都会更新表格 |

### 2.3.6模板模式

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。*

为了设计不同难度的游戏模式，我们在Game类中需要大量if判断，并且在判断之中编写该模式的代码；然而不同模式之间的代码有许多相似的，为了一些不同而全部重写会增加代码冗余量，因此我们选择引入模版模式。

在此处运用模版方法，可以减少代码冗余，代码组织更加清晰。

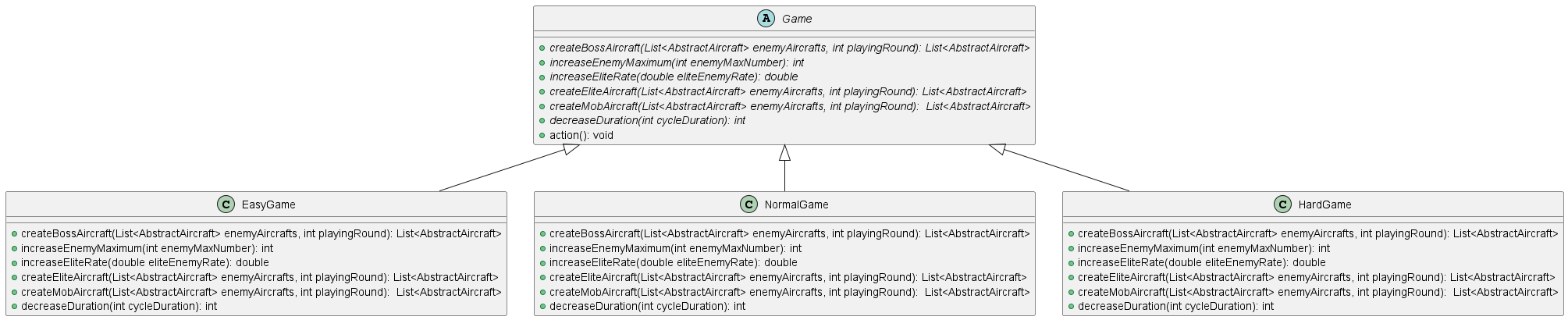
1. 三种难度

*请描述你的三种游戏难度是如何设计的，影响游戏难度的因素有哪些。*

|  |  |  |  |
| --- | --- | --- | --- |
| 因素 | 简单模式 | 普通模式 | 困难模式 |
| 敌机最大数量 | *5* | 最大敌机数量随时间增加 | 最大敌机数量随时间增加 |
| 精英敌机产生概率 | 20% | 最小20%，最大50%，匀速增加 | 最小20%，最大60%，匀速增加 |
| 精英敌机速度（y） | 5 | 最小为5，匀速增加 | 最小为5，匀速增加 |
| 普通敌机速度（y） | 6 | 最小为6，匀速增加 | 最小为6，匀速增加 |
| Boss机血量 | 无boss机 | 390（13颗子弹） | 最小为390，每一波敌机增加100 |
| 敌机和子弹刷新频率（ms/次） | 600 | 最大为600，步长为20减小，最小为100 | 最大为600，步长为40减小，最小为100 |

1. 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。*



*Game抽象类：*

|  |  |  |
| --- | --- | --- |
| createBossAircraft(List<AbstractAircraft>,int playingRound) | List<AbstractAircraft> | 生成boss敌机 |
| increaseEnemyMaximum(int enemyMaxNum) | Int | 提高最多敌机同时出现的数量 |
| createEliteAircraft(List<AbstractAircraft>,int playingRound) | List<AbstractAircraft> | 生成精英敌机 |
| createMobAircraft(List<AbstractAircraft>,int playingRound) | List<AbstractAircraft> | 生成普通敌机 |
| increaseEliteRate(double eliteEnemyRate) | double | 提高精英敌机产生的比例 |
| decreaseDuration(int cycleDuration) | Int | 削减敌机和子弹产生频率 |

# 收获和反思

在本次面向对象的实验过程中，我受益匪浅。通过各个实验，不仅对于面向对象概念的理解更加深刻了，同时也认识并理解了各种设计模式，也感觉自己的代码能力提升了一点。我觉得这门课的实验和课程关系非常密切，能通过上课预习实验，再通过实验加深理解，对于我个人而言是非常好的。

同时，我也觉得本次实验的实验报告非常好。实验前对于某个设计模式是懵懵懂懂、一知半解的，但是实验指导书上提供了充分且贴切的实例，既包括情境也包括代码，让人豁然开朗、茅塞顿开。