

（深圳）

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： 线上

学生班级： 5班

学生学号： 200110513

学生姓名： 宗晴

评阅教师：

报告成绩：

实验与创新实践教育中心制

2022年4月

# 实验环境

1. Windows 10

2. IntelliJ IDEA 2021.3.2

3. Java 11

# 实验过程

## 系统功能分析

*请结合文字、图表等方式（推荐使用功能层次图），清晰描述系统的功能。亮点功能请用\*标志。*

**

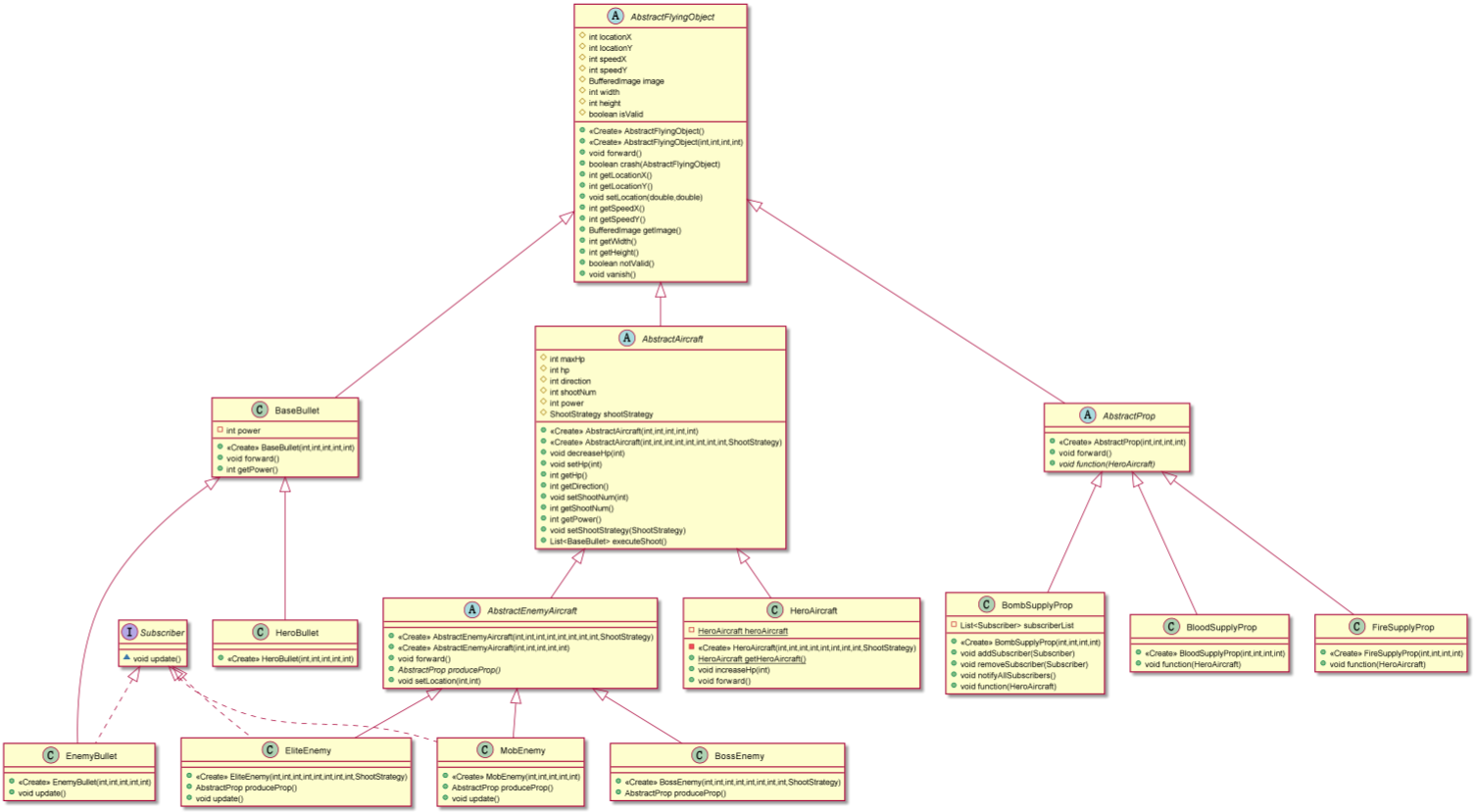
在道具生效功能中，实现了火力道具的累加功能，随着碰撞到的火力道具增加，英雄机射出的子弹数量也增加，最多同时射出3个子弹，此外道具生效的时间也会相应延长。

## 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。*



## 设计模式应用

### 2.3.1单例模式

1. 应用场景分析

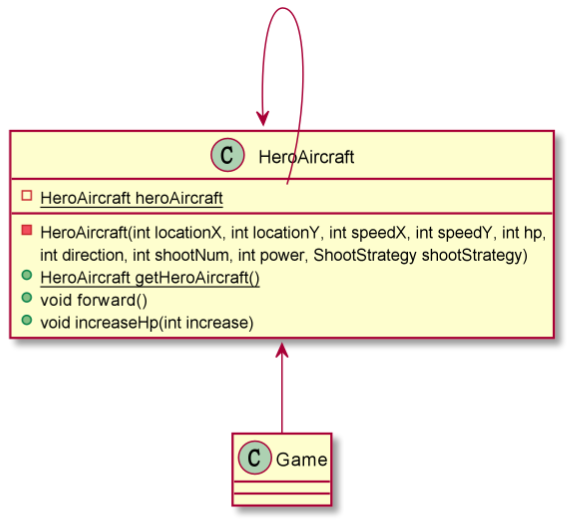
*描述飞机大战游戏中哪个应用场景需要用到此模式，设计中遇到的实际问题，使用该模式解决此问题的优势。*

应用场景：在实现英雄机时需要用到单例模式。因为在整个飞机大战中的设计只有一种英雄机，并且每局游戏中每位用户只有一架英雄机。

在设计中遇到的实际问题与该模式的优势：在Game中创建英雄机实例时无法保证创建的英雄机实例唯一，可能会出现重复创建的问题。若不使用该模式，则英雄机实例的创建和使用都在Game中完成，违反了单一职责原则，对象的创建和使用没有分离；在对英雄机的构造函数的参数等进行扩展时，均需对Game中的代码进行修改，并未做到对扩展开放、对修改关闭，违反了开闭原则。使用该模式可以保证在整场游戏中英雄机是唯一的，同时也可以保证单一职责原则和开闭原则。

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。*



HeroAircraftSingletonPattern的UML类图如上。

类图说明：

1. HeroAircraft类：应用单例模式的英雄机类，在整个飞机大战游戏中只有唯一的实例，外界通过getHeroAircraft这一静态方法获取该实例。可自动发射子弹，通过生命值（血）生存，被敌机子弹射中会损失部分生命值，与敌机碰撞则全部损失，当其生命值为0时游戏失败。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| heroAircraft | private | HeroAircraft | 英雄机的唯一静态实例 |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| HeroAircraft | private | / | int locationX, int locationY, int speedX, int speedY, int hp, int direction, int shootNum, int power, ShootStrategy shootStrategy  （横纵坐标、横纵速度、初始生命值、子弹的射击方向、子弹一次发射数量、单个子弹造成的伤害、子弹射击策略） | 构造函数 |
| getHeroAircraft | public | HeroAircraft | / | 实例化唯一的英雄机的静态方法，供外界获取英雄机实例 |
| forward | public | void | / | 控制英雄机的移动位置 |
| increaseHp | public | void | int increase（每次加血量） | 在获取加血道具后增加血量 |

1. Game类（包括三类难度的具体类）：用来实现游戏整体功能，执行游戏逻辑、实现游戏动画等。在Game类中调用getHeroAircraft()方法就可以获取英雄机的唯一实例。

### 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。*

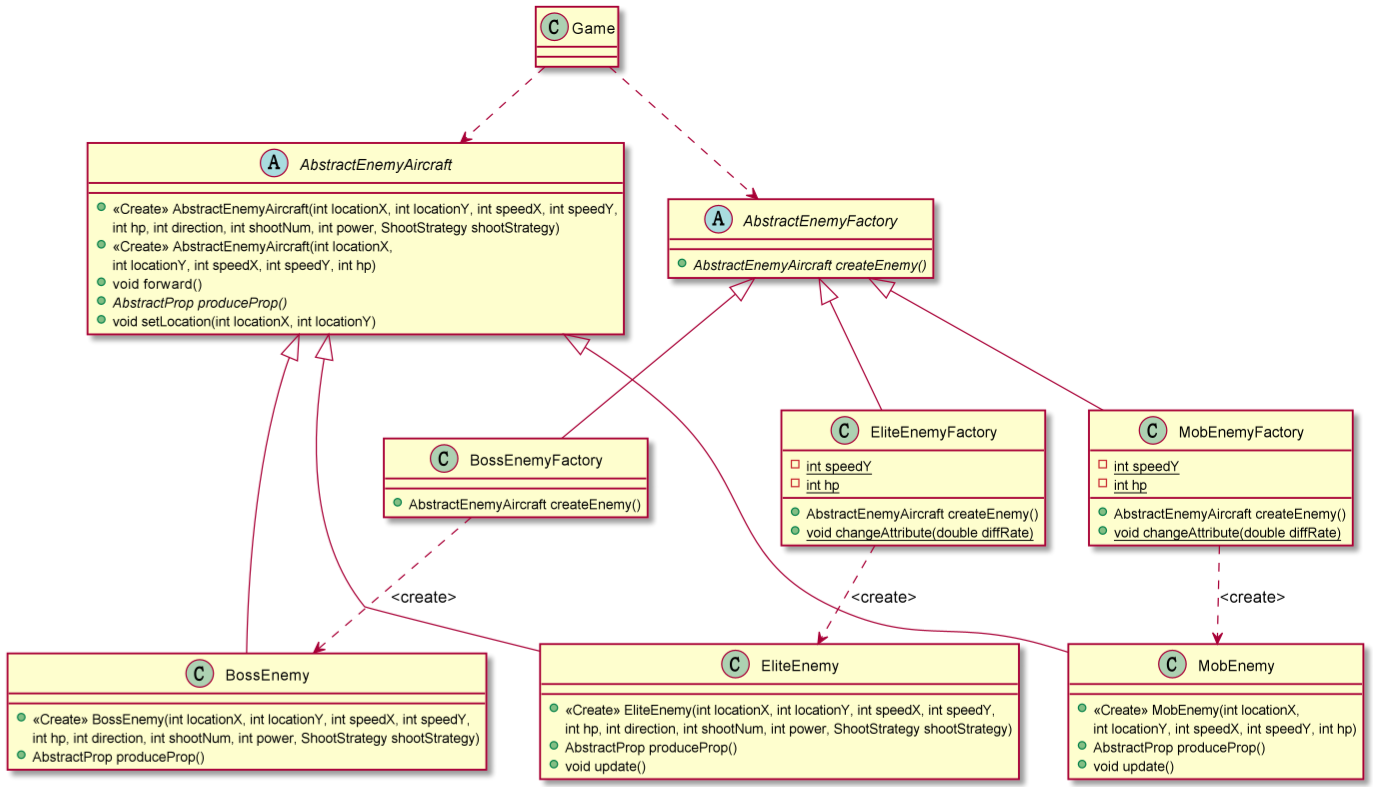
应用场景：在实现3种类型的敌机以及3种类型的道具时需要用到工厂模式。该模式在父类中提供一个创建对象的方法，由子类决定实例化对象的类型，运用了多态，可以有效避免实验一代码中存在的问题。

在设计中遇到的实际问题与该模式的优势：若不使用该模式，则敌机实例以及道具实例的创建和使用都在Game中完成，违反了单一职责原则，对象的创建和使用没有分离；在对敌机和道具的构造函数的参数等进行扩展时，均需对Game中的代码进行修改，并未做到对扩展开放、对修改关闭，违反了开闭原则；高层模块的实现依赖于低层模块，如Game模块依赖于敌机类模块，抽象层次依赖于具体层次，如AbstractAircraft的产生道具的动作依赖于具体道具类的实现，违反了依赖倒转原则。若使用该模式，则可以保证单一职责原则、开闭原则以及依赖倒转原则。

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。*

1. EnemyFactory



EnemyFactoryPattern的UML类图如上。

类图说明：

1. AbstractEnemyAircraft类：抽象敌机类，是三类具体敌机的父类。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| AbstractEnemyAircraft | public | / | int locationX, int locationY, int speedX, int speedY, int hp, int direction, int shootNum, int power, ShootStrategy shootStrategy  （横纵坐标、横纵速度、初始生命值、子弹的射击方向、子弹一次发射数量、单个子弹造成的伤害、子弹射击策略） | 有子弹的飞机(如EliteEnemy等)的构造函数，需要额外传入相应的子弹初始射击策略等相关参数 |
| AbstractEnemyAircraft | public | / | int locationX, int locationY, int speedX, int speedY, int hp（横纵坐标、横纵速度、初始生命值） | 无子弹的飞机(如MobEnemy)的构造函数 |
| forward | public | void | / | 控制敌机按照预定的参数运动，使其在超出左右边界后反向，在超出下边界后消失 |
| produceProp | public | AbstractProp | / | 控制敌机产生道具的*抽象方法*，由子类具体实现 |

1. AbstractEnemyFactory类：抽象敌机工厂类，是三类具体敌机工厂的父类。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createEnemy | public | AbstractEnemyAircraft | / | 创建各类敌机的*抽象方法*，由子类决定所产生敌机的种类 |

（3）MobEnemyFactory类：普通敌机工厂类，用来创建普通敌机实例。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| speedY | private | int | 普通敌机的纵向速度 |
| hp | private | int | 普通敌机的生命值（血量） |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createEnemy | public | AbstractEnemyAircraft | / | 随机在屏幕内任意位置创建一架普通敌机，该敌机能够按照设定血量和速度延直线向下运动 |
| changeAttribute | public | void | double diffRate（难度倍数） | 按比例改变普通敌机的纵向速度和生命值，从而增加难度 |

（4）EliteEnemyFactory类：精英敌机工厂类，用来创建精英敌机实例。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| speedY | private | int | 精英敌机的纵向速度 |
| hp | private | int | 精英敌机的生命值（血量） |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createEnemy | public | AbstractEnemyAircraft | / | 随机在屏幕内任意位置创建一架精英敌机，该敌机能够按照设定血量和速度，以50%的概率向左下或右下运动 |
| changeAttribute | public | void | double diffRate（难度倍数） | 按比例改变精英敌机的纵向速度和生命值，从而增加难度 |

（5）BossEnemyFactory类：Boss敌机工厂类，用来创建Boss敌机实例。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createEnemy | public | AbstractEnemyAircraft | / | 随机在屏幕内创建一架Boss敌机，该敌机能够悬浮在界面上方直至被消灭 |

（6）MobEnemy类：普通敌机类，实现普通敌机的发射子弹、产生道具等功能。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| MobEnemy | public | / | int locationX, int locationY, int speedX, int speedY, int hp（横纵坐标、横纵速度、初始生命值） | 构造函数 |
| produceProp | public | AbstractProp | / | 普通敌机不产生道具，因此返回null |
| update | public | void | / | 在炸弹爆炸时使普通敌机消失 |

（7）EliteEnemy类：精英敌机类，实现精英敌机的发射子弹、产生道具等功能。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| EliteEnemy | public | / | int locationX, int locationY, int speedX, int speedY, int hp, int direction, int shootNum, int power, ShootStrategy shootStrategy  （横纵坐标、横纵速度、初始生命值、子弹的射击方向、子弹一次发射数量、单个子弹造成的伤害、子弹射击策略） | 构造函数 |
| produceProp | public | AbstractProp | / | 按照概率控制坠毁的精英敌机产生产生三种道具之一，或者不产生道具 |
| update | public | void | / | 在炸弹爆炸时使精英敌机消失 |

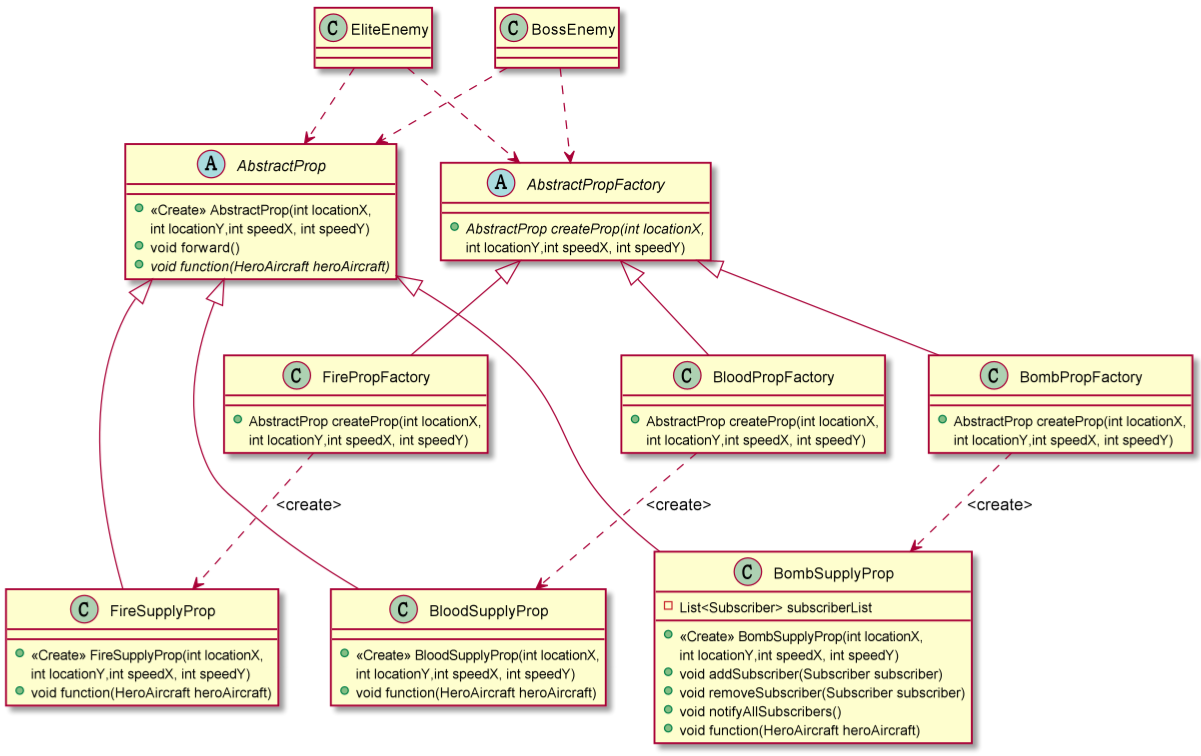
（8）BossEnemy类：boss敌机类，实现boss敌机的发射子弹、产生道具等功能。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| BossEnemy | public | / | int locationX, int locationY, int speedX, int speedY, int hp, int direction, int shootNum, int power, ShootStrategy shootStrategy  （横纵坐标、横纵速度、初始生命值、子弹的射击方向、子弹一次发射数量、单个子弹造成的伤害、子弹射击策略） | 构造函数 |
| produceProp | public | AbstractProp | / | 按照概率控制坠毁的boss敌机产生产生三种道具之一，或者不产生道具 |

（9）Game类（包括三类难度的具体类）：用来实现游戏整体功能，执行游戏逻辑、实现游戏动画等。在Game中声明抽象敌机工厂enemyFactory。按照概率随机产生普通敌机或精英敌机，在产生某类敌机时，需要先创建该类的工厂并赋给enemyFactory，然后再调用enemyFactory.createEnemy()创建对应的敌机实例。

1. PropFactory



PropFactoryPattern的UML类图如上。

类图说明：

（1）AbstractProp类：抽象道具类，是三类具体道具的父类。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| AbstractProp | public | / | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 构造函数 |
| forward | public | void | / | 控制道具按照预定的参数运动，使其在超出下边界后消失 |
| function | public | void | HeroAircraft heroAircraft, （英雄机） | 控制道具生效的*抽象方法*，由子类具体实现 |

（2）AbstractPropFactory类：抽象道具工厂类，是三类具体道具工厂的父类。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createProp | public | AbstractProp | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 创建各类道具的*抽象方法*，由子类决定所产生道具的种类 |

（3）BloodPropFactory类：加血道具工厂类，用来创建加血道具实例。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createProp | public | AbstractProp | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 在敌机坠毁位置创建一个加血道具，该道具能够增加英雄机血量 |

（4）FirePropFactory类：火力道具工厂类，用来创建火力道具实例。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createProp | public | AbstractProp | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 在敌机坠毁位置创建一个火力道具，该道具能够改变英雄机弹道 |

（5）BombPropFactory类：炸弹道具工厂类，用来创建炸弹道具实例。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| createProp | public | AbstractProp | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 在敌机坠毁位置创建一个炸弹道具，该道具能够清除敌机和子弹 |

（6）BloodSupplyProp类：加血道具类，实现加血道具的移动、加血等功能。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| BloodSupplyProp | public | / | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 构造函数 |
| function | public | void | HeroAircraft heroAircraft, （英雄机） | 加血道具生效后英雄机血量增加，同时道具消失 |

1. FireSupplyProp类：火力道具类，实现火力道具的移动、改变弹道等功能。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| FireSupplyProp | public | / | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 构造函数 |
| function | public | void | HeroAircraft heroAircraft, （英雄机） | 火力道具生效后，一段时间内英雄机改变弹道、发射子弹数量增加，同时道具消失，该道具效果可叠加 |

（8）BombSupplyProp类：炸弹道具类，实现炸弹道具的移动、清除敌机和子弹等功能。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| BombSupplyProp | public | / | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 构造函数 |
| addSubscriber | public | void | Subscriber subscriber  （订阅者） | 增加指定订阅者 |
| removeSubscriber | public | void | Subscriber subscriber  （订阅者） | 删除指定订阅者 |
| notifyAllSubscribers | public | void | / | 通知所有订阅者 |
| function | public | void | HeroAircraft heroAircraft, （英雄机） | 炸弹道具生效后，清除界面上除boss机外的所有敌机和敌机子弹，同时道具消失 |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| BombSupplyProp | public | / | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 构造函数 |
| function | public | void | HeroAircraft heroAircraft, （英雄机） | 炸弹道具生效后，清除界面上除boss机外的所有敌机和敌机子弹，同时道具消失 |

（9）EliteEnemy类/BossEnemy：用来实现道具工厂以及道具实例的创建等。在实际中，将创建道具工厂和实例化道具等过程整合到了产生道具的精英敌机和boss敌机类的function函数中。首先声明抽象道具工厂propFactory。按照概率随机产生三类道具中的一类或不产生道具，在产生某类道具时，需要先创建该类的工厂并赋给propFactory，然后再调用propFactory.createProp()方法创建对应的道具实例。

### 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。*

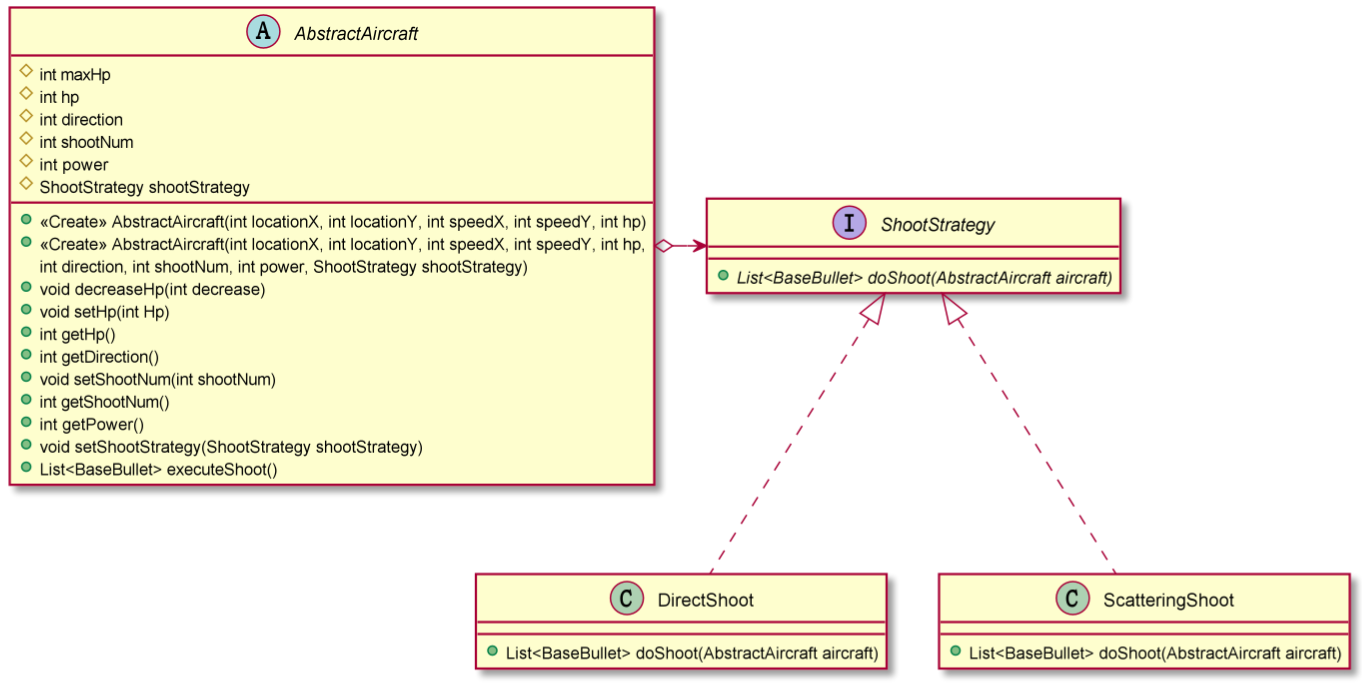
应用场景：在飞机大战中，英雄机子弹直射，在碰到火力道具后改为散射；精英敌机子弹直射；Boss敌机子弹散射。因此，在实现不同机型的弹道发射以及火力道具的加成时，需要用到策略模式。

在设计中遇到的实际问题：若不使用该模式，则各类飞机发射子弹的方法在具体飞机类中实现，若需要改变子弹发射的方式，则需要对相应具体飞机类的代码进行修改，不符合开闭原则；此外，当两种机型使用同一种子弹发射方式时（如英雄机默认的子弹直射以及精英敌机的子弹直射），它们需要在各自的类中分别实现该方法，会导致代码重复。

该模式的优势：各类飞机一共只有两种子弹发射方式，即直射和散射，飞机依据自身所处的环境（如是否碰到火力道具）而选择使用不同的子弹发射方式。因此使用策略模式最为合适。使用该模式后，可以实现代码重用，避免重复；同时符合开闭原则，把采取哪一种算法或行为的逻辑与算法本身分离，更加灵活、易于扩展，若想增加一种子弹发射方式，只需要增加一种策略即可，而不需要修改原本的代码；高内聚、低耦合。

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。*



StrategyPattern的UML类图如上。

（1）AbstractAircraft 抽象飞机类

该类在策略模式中等价于上下文（Context），用于维护指向具体策略（即直射策略和散射策略）的引用，且仅通过策略接口ShootStrategy与其进行交流。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| maxHp | protected | int | 飞机最大血量 |
| hp | protected | int | 飞机当前血量 |
| direction | protected | int | 子弹的射击方向 (向上发射：1，向下发射：-1) |
| shootNum | protected | int | 子弹一次发射的数量 |
| power | protected | int | 单个子弹造成的伤害 |
| shootStrategy | protected | ShootStrategy | 射击策略 |

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| AbstractAircraft | public | / | int locationX, int locationY, int speedX, int speedY, int hp（横纵坐标、横纵速度、初始生命值） | 无子弹的飞机(如MobEnemy)的构造函数 |
| AbstractAircraft | public | / | int locationX, int locationY, int speedX, int speedY, int hp, int direction, int shootNum, int power, ShootStrategy shootStrategy  （横纵坐标、横纵速度、初始生命值、子弹射击方向、子弹最大发射数量、单个子弹造成的伤害、子弹发射方法） | 有子弹的飞机(如Hero等)的构造函数，需要额外传入相应的子弹初始射击策略等相关参数 |
| setShootStrategy | public | void | ShootStrategy shootStrategy  （子弹发射方法） | 设置飞机的子弹发射策略，等价于上下文（Context）中setStrategy方法 |
| executeShoot | public | List<BaseBullet> | / | 调用实际策略相对应的子弹发射方法，等价于上下文（Context）中doSomething方法 |

1. ShootStrategy 射击策略接口

该类在策略模式中等价于策略接口（Strategy），它是所有具体策略（即直射策略和散射策略）的通用接口，声明了一个上下文（AbstractAircraft 抽象飞机类）用于执行策略的方法。

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| doShoot | public | List<BaseBullet> | AbstractAircraft aircraft  （飞机类） | 各类子弹发射方法的抽象方法，等价于策略接口（Strategy）中的execute方法 |

（3）DirectShoot 子弹直射策略

该类在策略模式中等价于具体策略（ConcreteStrategy），它是所有具体策略的其中一种。

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| doShoot | public | List<BaseBullet> | AbstractAircraft aircraft  （飞机类） | 实现具体的子弹直射方法，根据传入飞机的类型及最大子弹数等信息返回相应的子弹，等价于具体策略（ConcreteStrategy）中的execute方法 |

（4）ScatteringShoot 子弹散射策略

该类在策略模式中等价于具体策略（ConcreteStrategy），它是所有具体策略的其中一种。

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| doShoot | public | List<BaseBullet> | AbstractAircraft aircraft  （飞机类） | 实现具体的子弹散射方法，根据传入飞机的类型及最大子弹数等信息返回相应的子弹，等价于具体策略（ConcreteStrategy）中的execute方法 |

### 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。*

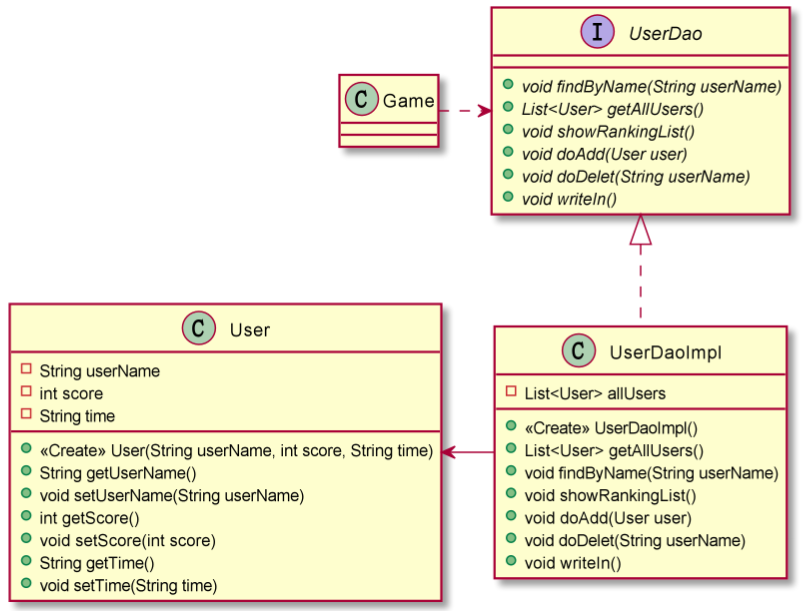
应用场景：在游戏结束后，需要显示该难度的得分排行榜。内容包括：名次、玩家名、得分和记录时间。我么需要把低级的数据访问操作从高级的业务服务中分离出来，因此使用数据访问对象模式较为合适。

在设计中遇到的实际问题：若不使用数据访问对象模式，则低级的数据访问操作与其它高级业务糅合在一起，不符合单一职责原则；同时在增加对于数据的其它操作时，需要在原有的Game类中修改，不符合开闭原则

该模式的优势：该模式隔离的数据层：新增了DAO层，不会影响到服务或者实体对象与数据库的交互，发生的错误也会在该层进行抛出。该模式将操作数据的部分分离了出来，符合单一职责原则；也便于增加对于数据的其它操作，符合开闭原则。

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。*



DAOPattern的UML类图如上。

（1）UserDao 用户数据访问对象接口

该类在数据访问对象模式中等价于数据访问对象接口（Data Access Object interface），该接口定义了在User对象上要执行的标准操作的抽象方法。

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| findByName | public | void | String userName  （用户名） | 查找某用户的记录 |
| getAllUsers | public | List<User> | / | 返回所有用户信息的列表 |
| showRankingList | public | void | / | 输出所有用户的得分排行榜，按照分数降序排列 |
| doAdd | public | void | User user  （用户对象） | 加入当前用户的信息 |
| doDelet | public | void | String userName  （用户名） | 删除某用户的记录 |
| writeIn | public | void | / | 将用户信息写回文件 |

（2）UserDaoImpl 用户数据访问对象实体类

该类在数据访问对象模式中等价于数据访问对象实体类（Data Access Object concrete class），该类实现了UserDao 用户数据访问对象接口。负责从文件中获取数据，并实现相应操作。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| allUsers | private | List<User> | 所有用户信息的列表 |

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| UserDaoImpl | public | / | / | 构造函数，负责读取历史用户信息的文件 |
| findByName | public | void | String userName  （用户名） | 查找某用户的记录 |
| getAllUsers | public | List<User> | / | 返回所有用户信息的列表 |
| showRankingList | public | void | / | 输出所有用户的得分排行榜，按照分数降序排列，同时将现有的全部信息写入文件中 |
| doAdd | public | void | User user  （用户对象） | 加入当前用户的信息 |
| doDelet | public | void | String userName  （用户名） | 删除某用户的记录 |
| writeIn | public | void | / | 将用户信息写回文件 |

（3）User 用户对象类

该类在数据访问对象模式中等价于模型对象/数值对象（Model Object/Value Object），该类是具体的用户对象类，包含了get/set等方法来存储通过使用了DAO类检索到的数据。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| userName | private | String | 用户名 |
| score | private | int | 用户的游戏得分 |
| time | private | String | 用户的记录时间 |

主要方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| User | public | / | String userName, int score, String time  （用户名、得分、记录时间） | 构造函数，按照传入的用户名、得分和时间构造一个用户对象 |
| getUserName | public | String | / | 获取用户名 |
| setUserName | public | void | String userName  （用户名） | 设置用户名 |
| getScore | public | int | / | 获取用户的游戏得分 |
| setScore | public | void | int score  （得分） | 设置用户的游戏得分 |
| getTime | public | String | / | 获取用户的记录时间 |
| setTime | public | void | String time  （时间） | 设置用户的记录时间 |

（4）Game类

在Game类中调用UserDao 用户数据访问对象接口，实现新增当前用户信息、打印所有用户的得分排行榜等功能。

### 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。*

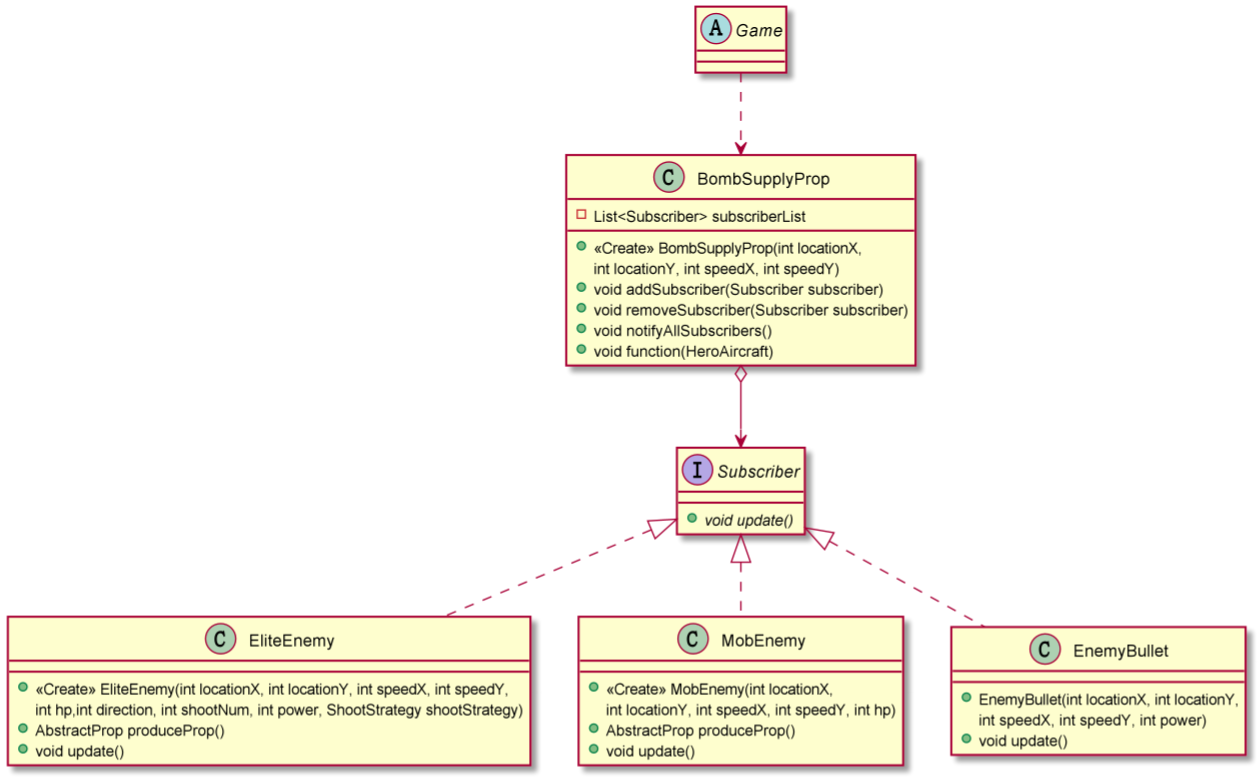
应用场景：在游戏中，若英雄机碰撞炸弹道具，炸弹道具生效后，可清除界面上除boss敌机外的所有敌机和敌机子弹，并把炸毁敌机所得的分数加到英雄机得分里。在清除敌机与敌机子弹时需要用到观察者模式。

在设计中遇到的实际问题：若不使用观察者模式，则需要在炸弹生效时，逐一清除除boss敌机外的所有敌机和敌机子弹，它们与炸弹之间高耦合；同时，增加新的需要一起清除的类时，需要修改原有代码，不符合开闭原则。

该模式的优势：该模式可以实现表示层和数据逻辑层的分离；在观察目标和观察者之间建立抽象的耦合；支持广播通信，简化了一对多系统设计的难度；在增加新的观察者时无需修改原有代码，符合开闭原则。

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。*



ObserverPattern的UML类图如上。

（1）BombSupplyProp类：炸弹道具类，实现炸弹道具的移动、清除敌机和子弹等功能。是观察者模式的发布者，当自身状态改变后，会通知所有订阅者进行相应改变。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| subscriberList | private | List<Subscriber> | 所有订阅者的列表 |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| BombSupplyProp | public | / | int locationX, int locationY, int speedX, int speedY（横纵坐标、道具横纵速度） | 构造函数 |
| addSubscriber | public | void | Subscriber subscriber  （订阅者） | 增加指定订阅者 |
| removeSubscriber | public | void | Subscriber subscriber  （订阅者） | 删除指定订阅者 |
| notifyAllSubscribers | public | void | / | 通知所有订阅者 |
| function | public | void | HeroAircraft heroAircraft, （英雄机） | 炸弹道具生效后，清除界面上除boss机外的所有敌机和敌机子弹，同时道具消失 |

（2）Subscriber接口：是所有具体订阅者的抽象接口，声明了一个抽象的更新方法，在得到发布者的通知时更新自己。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| update | public | void | / | 在得到发布者的通知时更新自己 |

（3）MobEnemy类：普通敌机类，实现普通敌机的发射子弹、产生道具等功能。是具体的观察者，在得到发布者的通知时消失。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| MobEnemy | public | / | int locationX, int locationY, int speedX, int speedY, int hp（横纵坐标、横纵速度、初始生命值） | 构造函数 |
| produceProp | public | AbstractProp | / | 普通敌机不产生道具，因此返回null |
| update | public | void | / | 在炸弹爆炸时使普通敌机消失 |

（4）EliteEnemy类：精英敌机类，实现精英敌机的发射子弹、产生道具等功能。是具体的观察者，在得到发布者的通知时消失。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| EliteEnemy | public | / | int locationX, int locationY, int speedX, int speedY, int hp, int direction, int shootNum, int power, ShootStrategy shootStrategy  （横纵坐标、横纵速度、初始生命值、子弹的射击方向、子弹一次发射数量、单个子弹造成的伤害、子弹射击策略） | 构造函数 |
| produceProp | public | AbstractProp | / | 按照概率控制坠毁的精英敌机产生产生三种道具之一，或者不产生道具 |
| update | public | void | / | 在炸弹爆炸时使精英敌机消失 |

（5）EnemyBullet类：敌机子弹类。是具体的观察者，在得到发布者的通知时消失。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| EnemyBullet | public | / | int locationX, int locationY, int speedX, int speedY, int power（横纵坐标、横纵速度、单个子弹造成的伤害） | 构造函数 |
| update | public | void | / | 在炸弹爆炸时使敌机子弹消失 |

（4）Game类

在Game类中创建炸弹道具，在英雄机碰撞炸弹道具时，炸弹道具生效，并通知所有订阅者。

### 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。*

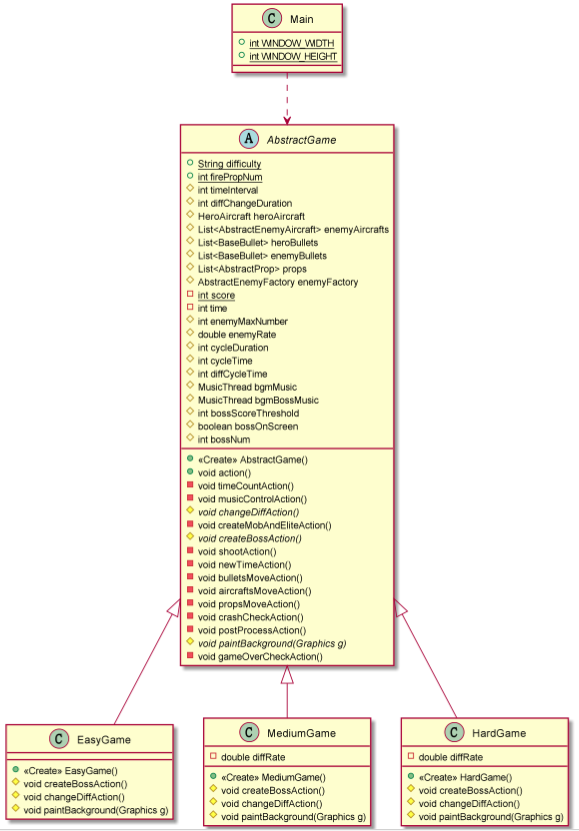
应用场景：用户进入飞机大战游戏后，可选择简单/普通/困难三种游戏难度之一。三种游戏模式的逻辑骨架相同，只有部分方法的具体实现方式不同，适合用模板模式进行解决。

在设计中遇到的实际问题：若不使用模板模式，则三种游戏模式之间的代码存在大量重复，并且在增加新的难度模式时也需要重新实现代码，不利于新增游戏难度。

该模式的优势：该模式定义了算法的骨架，把不变的行为搬到超类中，并且将一些步骤延迟到子类中实现，去除了子类中的重复代码，提高了代码的复用，同时便于子类扩展。

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。*

**

TemplatePattern的UML类图如上。

通过模板模式，实现了三种游戏模式的难度控制，包括：同一时刻出现的敌机数量最大值、各类敌机的血量和速度、各类飞机的射击周期、精英敌机的产生概率、普通和精英敌机的产生周期、boss敌机产生的得分阈值、boss敌机的血量。其中boss敌机的血量在困难模式中会随时间变化，精英敌机的产生概率、各类敌机产生的周期、各类敌机的血量和速度，在普通和困难模式中会随时间变化。

（1）AbstractGame类：抽象游戏类，定义并实现了一个模板方法action()，给出了顶层逻辑的骨架。声明作为算法步骤的方法，供模板方法依次调用。算法步骤包括抽象类型以及默认实现两种方式。

关键属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| difficulty | public | String | 表示游戏难度，分为简单/普通/困难三个模式 |
| firePropNum | public | int | 表示当前火力道具生效个数，控制其效果叠加 |
| time | private | int | 当前时间 |
| timeInterval | protected | int | 显示时间的间隔 |
| cycleTime | protected | int | 当前时间按刷新的周期表示 |
| cycleDuration | protected | int | 每次刷新的间隔，控制子弹的发射和敌机产生的频率，在普通与困难模式中会随时间变化 |
| diffCycleTime | protected | int | 当前时间按难度改变的周期表示 |
| diffChangeDuration | protected | int | 每次难度改变的间隔，简单模式无，普通模式比困难模式的大 |
| enemyMaxNumber | protected | int | 同一时间的最大敌机数量，三种模式各不相同 |
| enemyRate | protected | double | 精英敌机占总敌机的比例，三种模式各不相同，且普通与困难模式的会随时间增大 |
| bossOnScreen | protected | boolean | boss敌机是否在屏幕上的标志，用于控制boss敌机的背景音乐循环播放 |
| bossNum | protected | int | 已经出现过的boss敌机的数量，用于在困难模式中每次召唤提升boss敌机血量 |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| AbstractGame | public | / | / | 构造函数 |
| action | public | void | / | 游戏启动入口，执行游戏逻辑。模板模式的模板方法，依次调用以下方法。 |
| timeCountAction | private | void | / | 计时并周期性输出时间 |
| musicControlAction | private | void | / | 控制背景音乐和boss敌机音乐等循环播放 |
| changeDiffAction | protected | void | / | 周期性改变难度的抽象方法，待子类实现 |
| createMobAndEliteAction | private | void | / | 按照比例，周期性产生不超过设定总数的普通敌机和精英敌机，其中该比例、周期和总数在三种模式中各不相同，且比例和周期在普通和困难模式中随时间改变 |
| createBossAction | protected | void | / | 产生boss敌机的抽象方法，待子类实现 |
| shootAction | private | void | / | 飞机周期性射出子弹，该周期三种模式中各不相同，且在普通和困难模式中随时间改变 |
| newTimeAction | private | void | / | 更新时间，判断各时间是否更新到新的周期 |
| bulletsMoveAction | private | void | / | 子弹移动 |
| aircraftsMoveAction | private | void | / | 飞机移动 |
| propsMoveAction | private | void | / | 道具移动 |
| crashCheckAction | private | void | / | 撞击检测 |
| postProcessAction | private | void | / | 后处理，包括删除无效的子弹、删除无效的敌机、检查英雄机生存 |
| paintBackground | protected | void | Graphics g | 按照不同游戏模式绘制不同背景图 |
| gameOverCheckAction | private | void | / | 游戏结束检查 |

（2）EasyGame类：简单游戏类，模板模式的一个具体类，按照其父类抽象游戏类中的逻辑进行执行，同时实现了其父类中的抽象方法。

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| EasyGame | public | / | / | 构造函数，同时设置敌机最大数量、刷新周期和精英敌机比例 |
| createBossAction | protected | / | / | 简单模式不产生boss敌机，产生boss敌机函数为空 |
| changeDiffAction | protected | / | / | 简单模式不改变难度，改变难度函数为空 |
| paintBackground | protected | / | Graphics g | 绘制简单模式背景图 |

（3）MediumGame类：普通游戏类，模板模式的一个具体类，按照其父类抽象游戏类中的逻辑进行执行，同时实现了其父类中的抽象方法。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| diffRate | private | double | 表示游戏难度提升的比例，此比例随时间变化 |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| MediumGame | public | / | / | 构造函数，同时设置敌机最大数量、刷新周期、精英敌机比例和boss敌机产生的得分阈值 |
| createBossAction | protected | / | / | 按照得分阈值产生boss敌机，同时播放boss敌机出场音乐 |
| changeDiffAction | protected | / | / | 周期性改变难度，包括精英敌机比例、各类敌机产生周期、各类敌机血量与速度、各类飞机射击周期等 |
| paintBackground | protected | / | Graphics g | 绘制普通模式背景图 |

（4）HardGame类：困难游戏类，模板模式的一个具体类，按照其父类抽象游戏类中的逻辑进行执行，同时实现了其父类中的抽象方法。

属性：

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 访问域 | 类型 | 含义 |
| diffRate | private | double | 表示游戏难度提升的比例，此比例随时间变化 |

方法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 访问域 | 返回值类型 | 输入参数 | 功能 |
| HardGame | public | / | / | 构造函数，同时设置敌机最大数量、刷新周期、精英敌机比例和boss敌机产生的得分阈值 |
| createBossAction | protected | / | / | 按照得分阈值周期性产生boss敌机（血量逐次增加），同时播放boss敌机出场音乐 |
| changeDiffAction | protected | / | / | 周期性改变难度，包括精英敌机比例、各类敌机产生周期、各类敌机血量与速度、各类飞机射击周期等 |
| paintBackground | protected | / | Graphics g | 绘制困难模式背景图 |

（5）Main类

在Main类中，依据抽象游戏类中的难度标签，调用不同难度等级的游戏类。

# 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！*

在本次实验中，我们从一个简单的飞机大战出发，逐步实现了一个功能完善、具有多种难度的最终版本。实验的选题十分有趣，完成整个实验后也给我们提供了极大的成就感。在每次实验中，我们都会重构代码以实现不同的模式，使得代码更加规范，同时也会相应地实现新的功能。通过实验中的实际操作，我们对于理论课上学习到的知识有了更加深刻全面的认识，也理解了这些模式的优势以及它们对于代码实现的便利。当然在实验起初时，我的进展并不是很顺利，在课内学习的知识并不能使我直接用代码实现实验所要求的功能。后来，我通过仔细阅读参考书、查看ppt，终于能顺利地完成实验内容。

我认为实验课对于我们理解理论课上学习的知识是非常重要的，就我个人而言，只有在实验课上亲自实践过后，我才会真正理解这些理论知识。诚然课后需要花费很多时间完成实验以及实验报告，并且每周刚完成上一次实验就又迎来了下一次实验，但是做出最终版的飞机大战游戏后还是非常有成就感的。