# 实验二报告

## 一、 单例模式

1. 应用场景分析

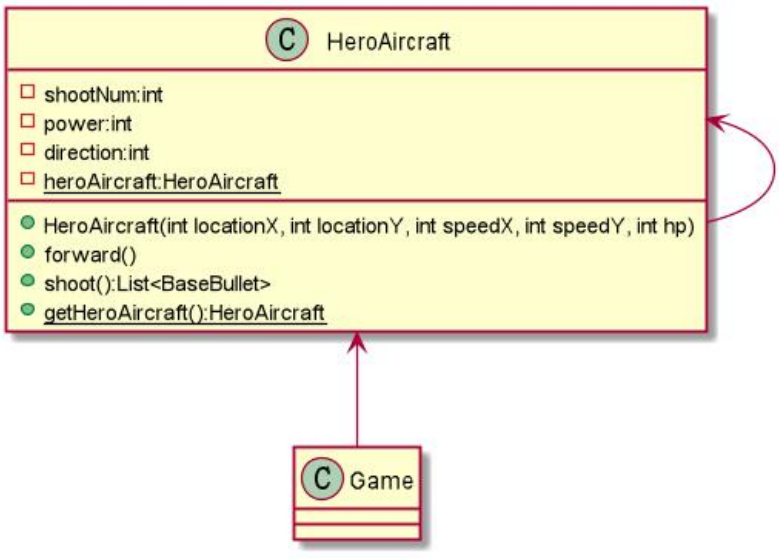
应用单例模式的场景是英雄机的构建，由于在飞机大战游戏中只有一种英雄机，且每局游戏只有一架英雄机，要保证英雄机的类必须只有一个实例存在，所以考虑使用单例模式。

目前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。

2. 解决方案

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。

uml 图:




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，以及血量等基本属性。
getHeroAircraft 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。

关键代码如下:

```
private volatile static  HeroAircraft heroAircraft;

public static HeroAircraft getHeroAircraft(){

        if (heroAircraft==null){

            synchronized (HeroAircraft.class) {

                if (heroAircraft == null) {

                    heroAircraft = new HeroAircraft(Main.WINDOW_WIDTH / 2,

                            Main.WINDOW_HEIGHT -

ImageManager.HERO_IMAGE.getHeight(),

                            0, 0, 100);

                }

            }

 }

return  heroAircraft

;}
```

# 二、 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。

目前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。

## 2. 解决方案

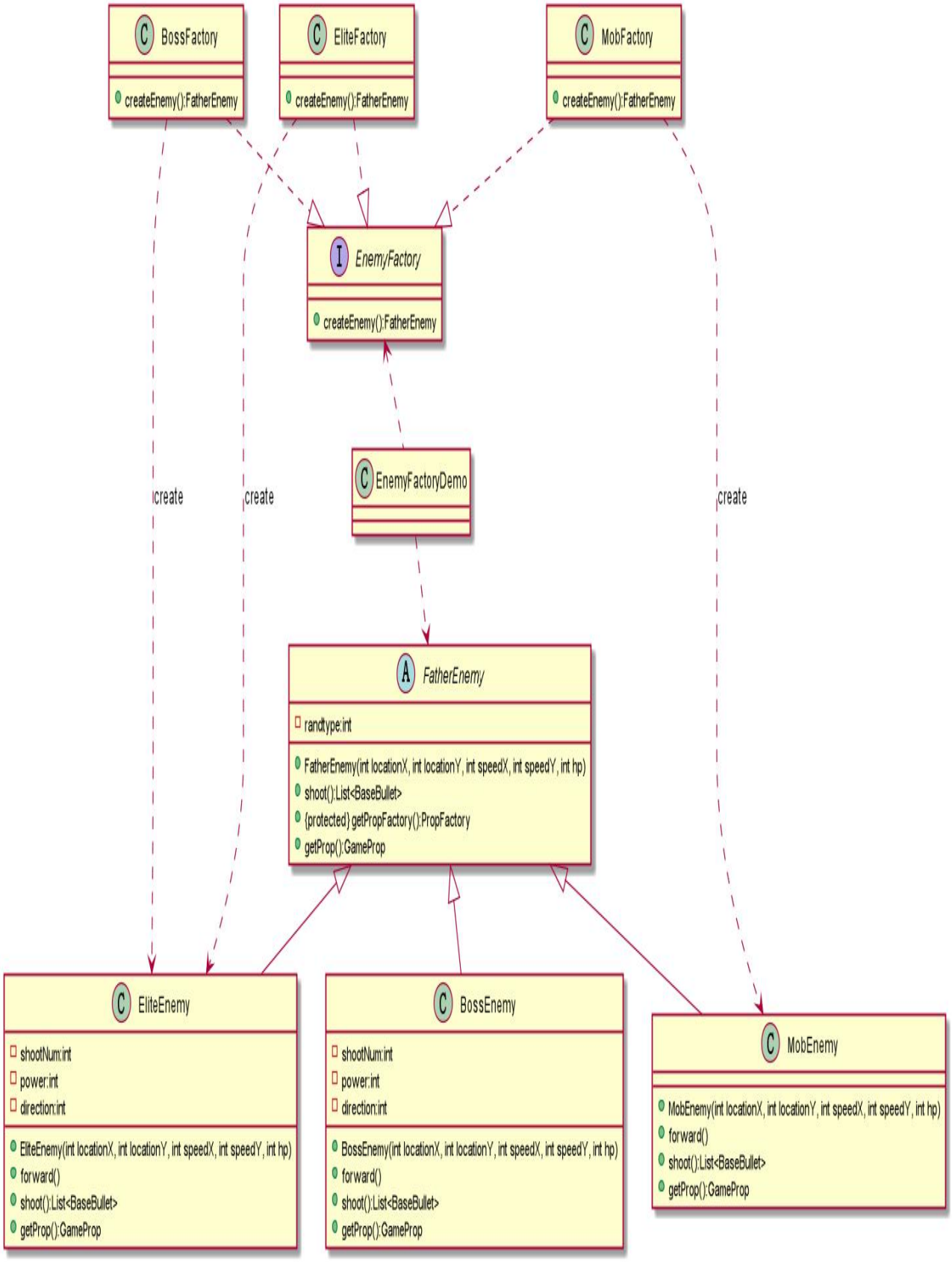
解决思路:

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。并为敌机创建一个新父类 FatherEnemy，用于进一步分割开敌机与英雄机的关联，减少代码量。

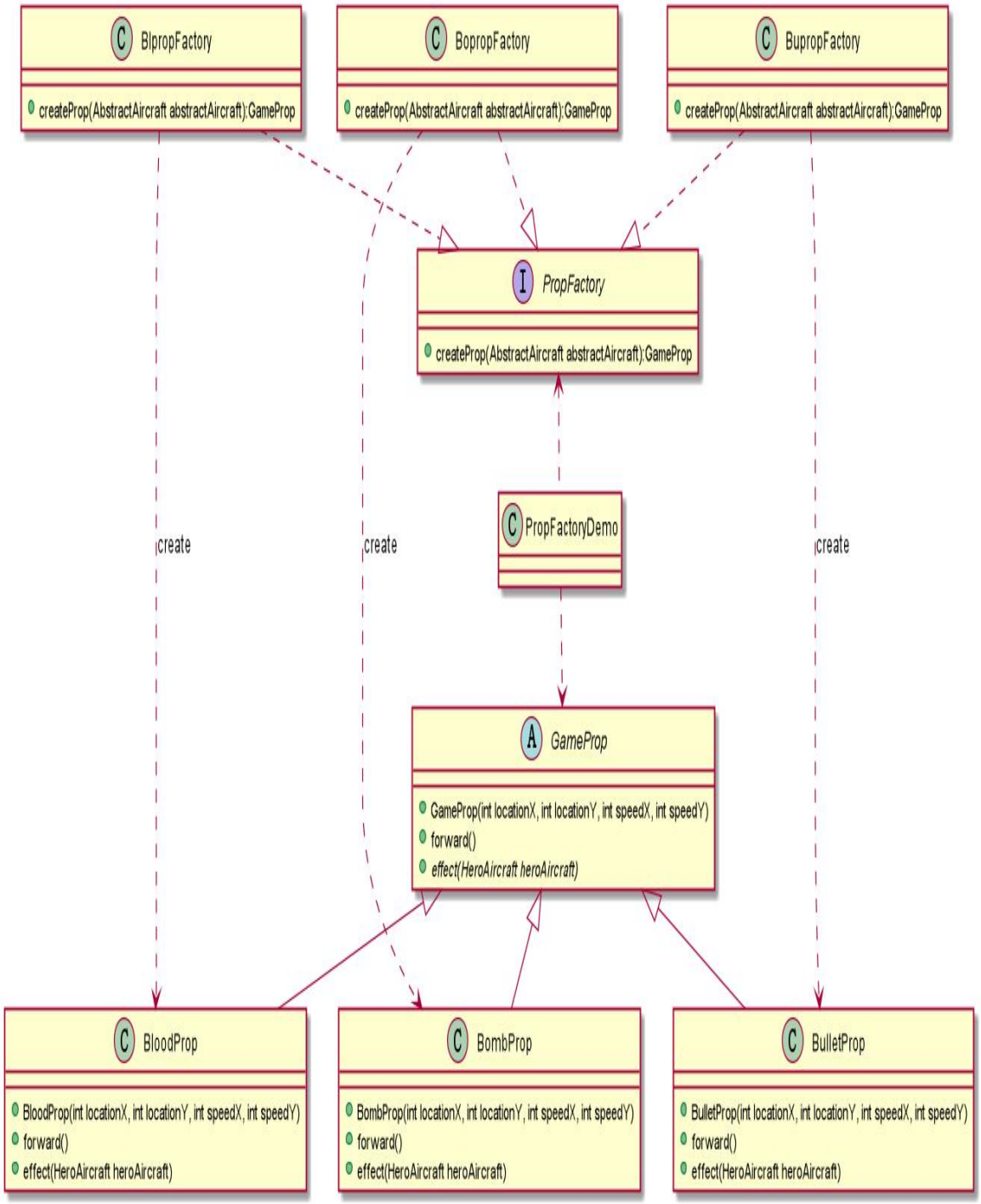
同理设计创建道具的工厂接口 ProbFactory，由类 BlpropFactory、BopropFactory、BupropFactory，实现具体道具的创建工作。

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，将道具工厂类的创建方法由敌机类调用，在敌机类中创建一个返回创建道具的方法 getProp,客户端通过调用该方法加入道具。

uml 图:
敌机工厂 Demo:



道具工厂 Demo：


BlpropFactory
createProp(AbstractAircraft abstractAircraft):GameProp
BopropFactory
createProp(AbstractAircraft abstractAircraft):GameProp
BupropFactory
createProp(AbstractAircraft abstractAircraft):GameProp
PropFactory
createProp(AbstractAircraft abstractAircraft):GameProp
PropFactoryDemo
create
create
create
GameProp
GameProp(int locationX, int locationY, int speedX, int speedY)
forward()
effect(HeroAircraft heroAircraft)
BloodProp
BloodProp(int locationX, int locationY, int speedX, int speedY)
forward()
effect(HeroAircraft heroAircraft)
BombProp
BombProp(int locationX, int locationY, int speedX, int speedY)
forward()
effect(HeroAircraft heroAircraft)
BulletProp
BulletProp(int locationX, int locationY, int speedX, int speedY)
forward()
effect(HeroAircraft heroAircraft)

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， 它拥有一个抽象方法 createEnemy。 由 MobFctory、BossFactory、EliteFactory 三个敌机类分别具体实现该抽象方法，分别创建出普通敌机 MobEnemy、Boss 敌机 BossEnemy、精英敌机 EliteEnemy。

这三个敌机类是抽象类 FatherEnemy 的子类，并实现该类的抽象方法 shoot 和 getProb，可以射击和生成道具的产生射击和道具掉落效果，无法实现的方法返回空。且 FatherEnemy 中定义一个私有变量 randtype 用于控制随机产生道具，同时定义一个得到道具工厂的保护方法 getPropFactory,只允许其子类访问，子类通过该方法得到工厂，调用对应工厂类的创建方法生成道具。再由客户端调用公共方法 getProp 加入道具。

对于道具创建的工厂模式类图:ProbFactory 是一个接口，它拥有一个抽象方法 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.