1．需求分析

（1）炸弹人玩家段位等级设计，不同级别可以选择对应难度的关卡地图。

一共有3个段位，分别为青铜、白银、黄金。挑战模式中，每赢一局就能获得一颗星反之则失去一颗星。获得的星数与段位存在一种联系，当星数达到一定的程度，系统会匹配出相应的段位。玩家等级越高，可选择的关卡越多，同时难度也更大。

（2）不同级别难度下游戏地图可以产生不同类型的建筑及道具。

2．概要设计

在很多情况下，可以处理某个请求的对象不止一个。很多对象都可以处理申请，而且它们构成一条链，请求沿着这条链传递，这条链就称为责任链。责任链可以是一条直线、一个环或者一个树形状，最常见的责任链是直线型，即沿着一条单向的链来传递请求。链上的每个对象都是请求处理者，责任链模式可以将请求的处理者组织成一条链，并使请求沿着链传递，由链上的处理者对请求进行相应的处理，客户端无须关心请求的处理细节以及请求的传递，只需将请求发送到链上即可。将请求的发送者和请求的处理者解耦，这就是责任链的模式动机在本游戏实例中，玩家等级星的数量先从青铜类评定，评定过了会传递给后面责任链，直到被处理。

职责链模式包含以下角色：

Handler(抽象处理者) 抽象处理者定义了一个处理请求的接口，它一般设计为抽象类，由于不同的具体处理者处理请求的方式不同，因此在其中定义了抽象请求处理方法。因为每个处理者的下家还是一个处理者，因此在抽象处理中定义了一个自类型(抽象处理者类型)的对象，作为其对下家的引用。通过该引用，处理者可以连为一条链。

Base Handler(基础处理者) 是一个可选的类，可以将所有处理者共用的样本代码放置在其中。通常情况下，该类中定义了一个保存对于下个处理者引用的成员变量。客户端可通过将处理者传递给上个处理者的构造函数或设定方法来创建链。该类还可以实现默认的处理行为：确定下个处理者存在后再将请求传递给它。

ConcreteHandler(具体处理类) 具体处理类是抽象处理类的子类，它可以处理用户请求，在具体处理者类中实现了抽象处理者中定义的抽象请求处理方法，在处理请求之前需要进行判断，看是否有对应的处理权限，如果可以处理请求就处理它，否则将请求转发给后继者；在具体处理类中可以访问链中下一个对象，以便请求转发。

Client(客户类) 客户类用于向链中的对象提出最初的请求，客户类只用关心链的源头，而无须关心请求的处理细节以及请求的传递过程。

抽象工厂模式是工厂方法模式的泛化版，工厂方法模式是一种特殊的抽象工厂模式。在工厂方法模式中，每一个具体工厂只能生产一种具体产品，而在抽象工厂方法模式中，每一个具体工厂可以生产多个具体产品。在本实例中，高级别难度下生产的产品建筑和道具功能复杂，支持高段位等级的玩家挑战，低级别难度下生产的产品建筑和道具功能简单，支持低段位等级的玩家挑战。针对高、中、低分辨率的移动设备使用不同的显示资源和帧率。

抽象工厂模式涉及的角色有：

AbstractFactory(抽象工厂) 抽象工厂用于声明生成抽象产品的方法，在一个抽象工厂中可以定义一组方法，每一个方法对应一个产品等级结构。

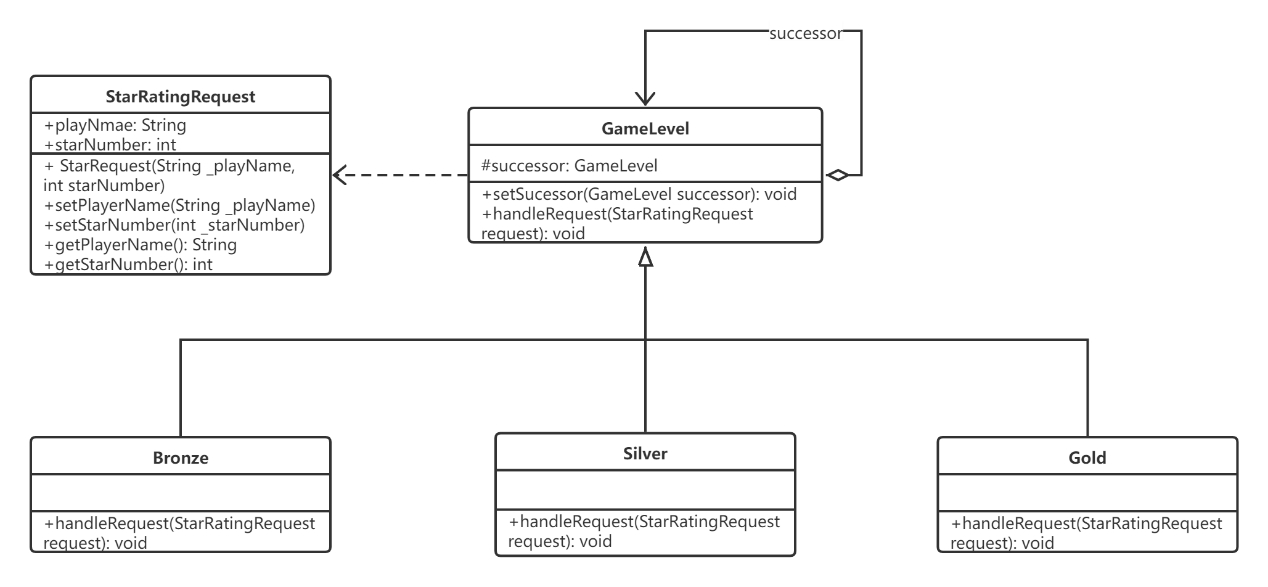
ConcreteFactory(具体工厂) 具体工厂实现了抽象工厂声明的生成抽象产品的方法，生成一组具体产品，这些产品构成了一个产品族，每一个产品都位于某个产品等级结构中。

AbstractProduct(抽象产品) 抽象产品为每种产品声明接口，在抽象产品中定义了产品的抽象业务方法。

ConcreteProduct(具体产品) 具体产品定义具体工厂生产的具体产品对象，实现抽象产品接口中定义的业务方法。

3．详细设计

炸弹人三个段位：青铜，白银，黄金。星数量少于15，对应段位为青铜；星数量在15-35之间对应段位为白银；星数在35-55之间对应段位为黄金；多于55颗星，表示为黄金(X-55)星。



**创建请求类(StarRatingRequest)**

/\* \* 创建请求类，由客户端指定其属性

\* @param playerName玩家名称

\* @param starNumber星的数量 \*/

public class StarRatingRequest {

private String playerName;

private int starNumber;

public StarRatingRequest(String \_playerName, int \_starNumber) {

this.playerName = \_playerName;

this.starNumber = \_starNumber;

}

public String getPlayerName() {

return playerName;

}

public void setPlayerName(String playerName) {

this.playerName = playerName;

}

public int getStarNumber() {

return starNumber;

}

public void setStarNumber(int starNumber) {

this.starNumber = starNumber;

}

}

**创建抽象处理者类(GameLevel)**

/\* \* 创建抽象处理者类 \*/

public abstract class GameLevel {

//定义自类型

protected GameLevel successor;

//接收下一家对象

public void setSuccessor(GameLevel successor) {

this.successor = successor;

}

//抽象处理方法

public abstract void handleRequest(StarRatingRequest request);

}

**创建具体处理者类(Bronze)**

/\*

\* 创建具体处理者类

\* 当客户端传入的星数少于15，由该对象处理，否则传递给客户端指定的下家

\*/

public class Bronze extends GameLevel {

@Override

public void handleRequest(StarRatingRequest request) {

if(request.getStarNumber() < 15) {

System.out.println("玩家[" + request.getPlayerName() + "]统计为"

+ request.getStarNumber() + "颗星，评定为等级：青铜");

}else {

if(this.successor != null) {

this.successor.handleRequest(request);

}

}

}

}

**创建具体处理类(Silver)**

/\*

\* 创建具体处理者类

\* 当客户端传入的星数在15-35，由该对象处理，否则传递给客户端指定的下家

\*/

public class Silver extends GameLevel {

@Override

public void handleRequest(StarRatingRequest request) {

if(request.getStarNumber() < 35) {

System.out.println("玩家[" + request.getPlayerName()

+ "]统计为" + request.getStarNumber() + "颗星，评定为等级：白银");

}else {

if(this.successor != null) {

this.successor.handleRequest(request);

}

}

}

}

**创建具体处理类(Gold)**

/\*

\* 创建具体处理者类

\* 当客户端传入的星数在35-55，由该对象处理，否则评定为王者段位

\*/

public class Royalty extends GameLevel {

@Override

public void handleRequest(StarRatingRequest request) {

if(request.getStarNumber() < 55) {

System.out.println("玩家[" + request.getPlayerName() + "统计为"

+ request.getStarNumber() + "颗星，评定为等级：黄金");

}else {

System.out.println("玩家[" + request.getPlayerName()

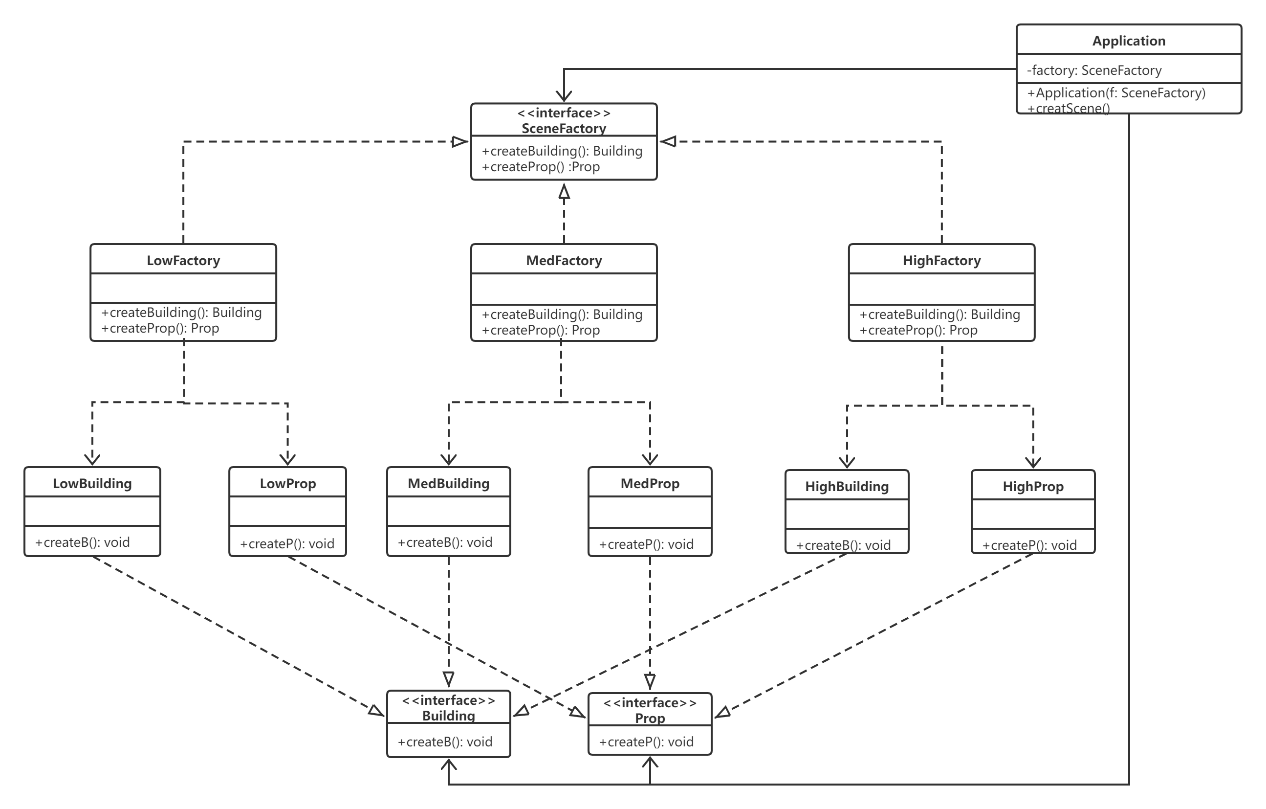
+ "]统计为" + request.getStarNumber()

+ "颗星，评定为等级：黄金" + (request.getStarNumber()-55) + "星");

}

}

}



**创建抽象产品类Building**

/\*

\* 抽象产品类

\*/

public interface Building {

public void createB();

}

**创建抽象产品类Prop**

/\*

\* 创建抽象产品类

\*/

public interface PlayGame {

public void createP();

}

**创建具体产品类LowBuilding**

/\*

\* 具体产品类，创建低级别地图建筑物

\*/

public class LowBuilding implements Building{

public void createB () {

}

}

**创建具体产品类LowProp**

/\*

\* 具体产品类，创建低级别地图道具

\*/

public class LowProp implements Prop{

public void createP () {

}

}

**创建具体产品类MedBuilding**

/\*

\* 具体产品类，创建中级别地图建筑物

\*/

public class MedBuilding implements Building{

public void createB () {

}

}

**创建具体产品类MedProp**

/\*

\* 具体产品类，创建中级别地图道具

\*/

public class MedProp implements Prop{

public void createP () {

}

}

**创建具体产品类HighBuilding**

/\*

\* 具体产品类，创建高级别地图建筑物

\*/

public class HighBuilding implements Building{

public void createB () {

}

}

**创建具体产品类HighProp**

/\*

\* 具体产品类，创建高级别地图道具

\*/

public class HighProp implements Prop{

public void createP () {

}

}

**创建抽象工厂类SceneFactory**

/\*

\* 创建抽象工厂类

\*/

public interface SceneFactory {

public Building createBuilding ();

public Prop createProp ();

}

**创建具体工厂类****LowFactory**

/\*

\* 创建具体工厂类

\*/

public class LowFactory implements SceneFactory {

public Building createBuilding () {

return new LowBuilding ();

}

public Prop createProp () {

return new LowProp ();

}

}

**创建具体工厂类MedFactory**

/\*

\* 创建具体工厂类

\*/

public class MedFactory implements SceneFactory {

public Building createBuilding () {

return new MedBuilding ();

}

public Prop createProp () {

return new MedProp ();

}

}

**创建具体工厂类****HighFactory**

/\*

\* 创建具体工厂类

\*/

public class HighFactory implements SceneFactory {

public Building createBuilding () {

return new HighBuilding ();

}

public Prop createProp () {

return new HighProp ();

}

}