**项目报告 - 闯塔通关游戏**

朱宇翔 MF1832279

一．游戏思路概述

总体上，我设计了一个以回合制攻防战斗为内核（类似梦幻西游，玩家和敌人轮流向对方发起进攻）的塔防通关的游戏。项目以被囚禁的王子为背景，描述了一个奋起反击的公主的拯救王子的故事。由于游戏制作经费有限，未能在游戏实现中加入剧情有关要素，我希望在这里简单介绍一下。邪恶女王巴拉拉求王子的爱而不得，愤而将其囚禁在白色巨塔顶层（在游戏内暂时设置为三层），公主为了拯救心爱的王子，拿起武器一层层杀遍每一层的怪物，当杀完最顶层的怪物之后，王子终于和公主获得了团圆。

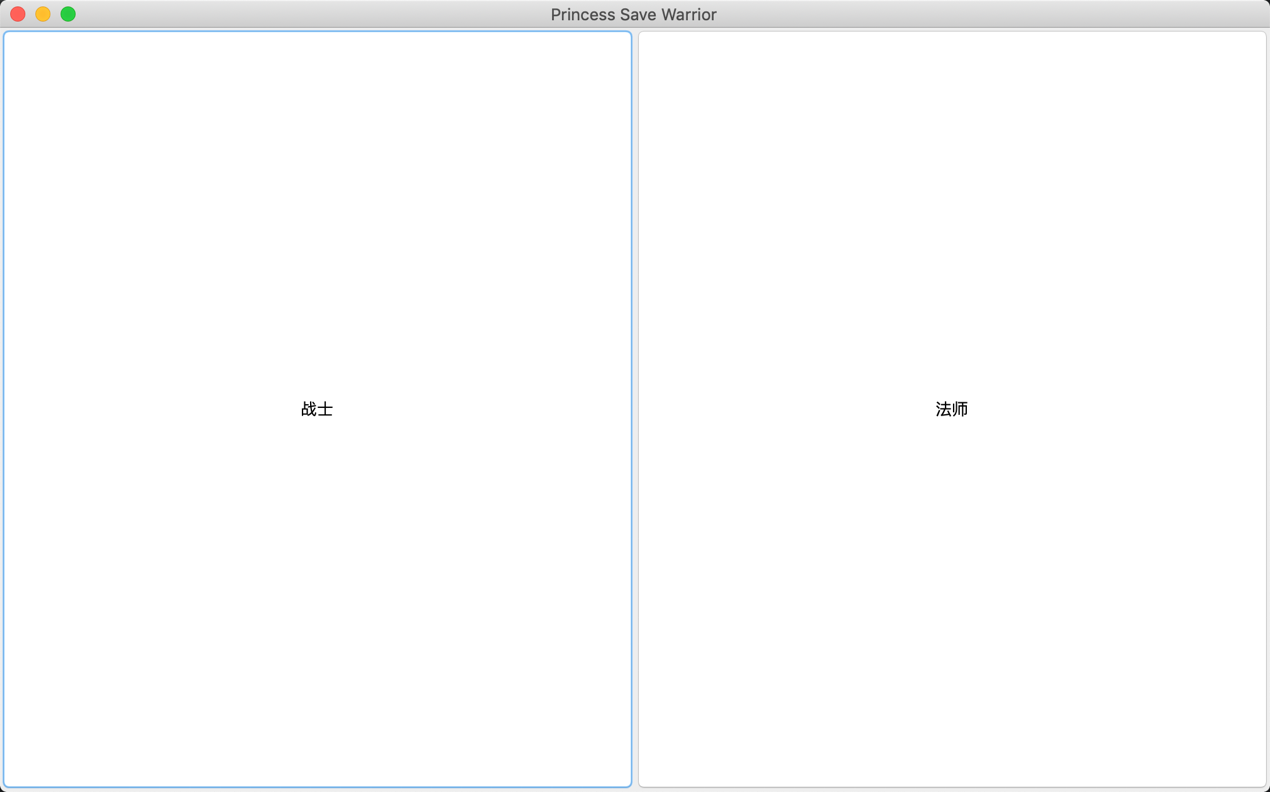
首先，公主将从多个职业（暂时设置为战士和法师两种）中选择一种来开始旅途，每个职业的技能不同，主要思路可能也不同。在公主艰辛的拯救历程中，怪物每一层越来越强大，而她打完怪物后也会获得相应的金钱和经验。公主不断地升级，从而提升自己的防御力和攻击力；公主也能使用自己的金钱去购买武器和强化武器，以额外提高自己的攻击力和防御力。此外，公主还能为自己的技能购买秘籍来强化自己技能的伤害和效果。有些技能组合使用还有特殊效果，如有个技能能为其他技能提升伤害。

在实际战斗中，怪物和公主轮流攻击对方。公主释放技能攻击怪物后屏幕上会显示对怪物造成的伤害。然后进入怪物回合，屏幕上会显示怪物对公主造成的伤害，周而复始，直至公主死亡或所有怪物死亡。

二．游戏界面展示

下面，让我们来具体地看一看游戏效果。

一进入，简单明了的界面显示两个巨大的按钮平分屏幕，提示玩家应选择职业。我们选择听上去较为暴力的战士开始游戏。



随之而来的即是游戏主界面，左上角是一个明显的楼层跳转和去往商店的”电梯“，供玩家快速跳转楼层。左下角面板是角色个人属性面板，包括了角色个人能力值，武器信息，经验金钱信息和血量槽。右侧最大的面板则是战斗面板，显示了公主在这一层遇到的敌人，在第一层中有六只怪物，其中四只攻击力防御力较低，两只攻击力防御力较高；战斗面板还清晰地显示了怪物血量百分比（下面蓝色槽即代表他们的血量值）。右下角则分别是技能选择面板和日志面板，分别代表了可选的技能和记录的日志信息。



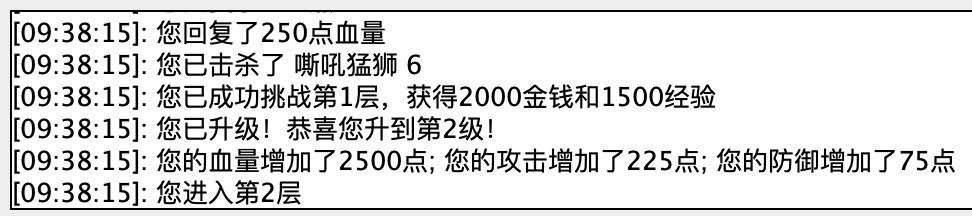
当我们使用一个技能后，战斗面板就会以优美的渐变效果显示怪物扣的血量。



当血量显示动画结束后，轮到怪兽对玩家发起攻击，经过计算后面板显示怪物对玩家造成的伤害。（截图上显示的红色有深有浅正是由于渐变特效带来的）



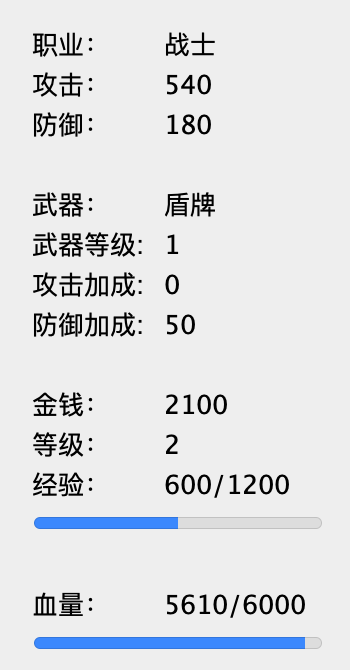
当所有怪物都被击杀后，公主会通过本层关卡，获得相应奖励和金钱。由于经验达标，公主还升级了，提升了各项基本属性。这些信息都会贴心地打印在下方的日志面板中。

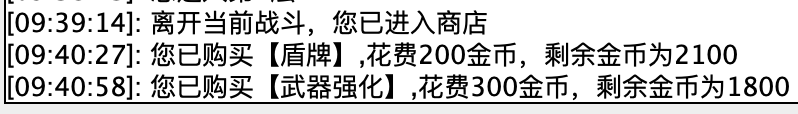


获得金钱后，可进入商店面板选购公主心仪的商品。商品详细写出了效果，所需金币和名称。明码标价，童叟无欺。



比如我们选购一项武器，就能在左侧面板看到武器的实装效果，再进行一次强化查看效果。在左侧面板更新的同时，下方日志面板也会显示这些操作。





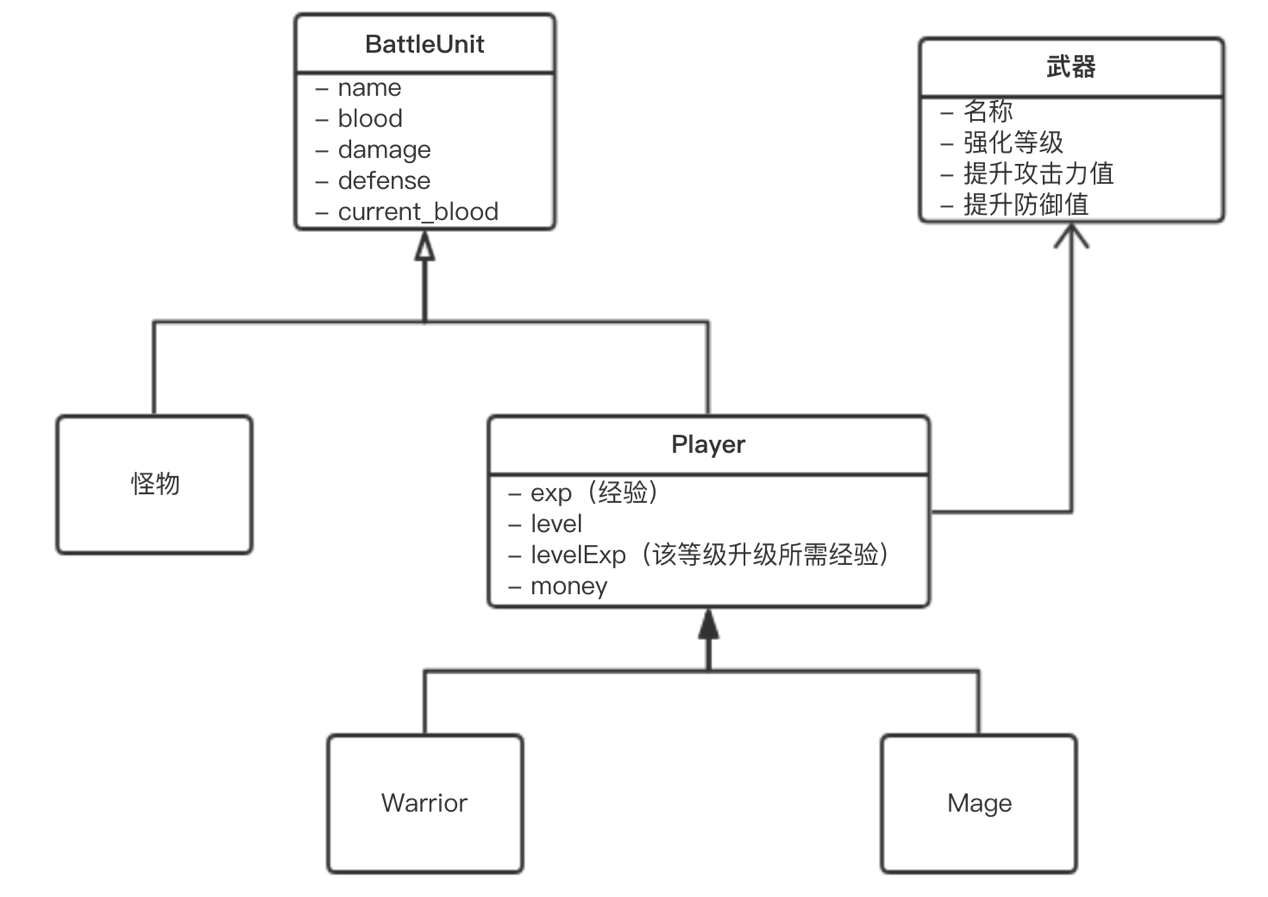
当所有关卡都结束后，本游戏也就挑战成功了。

三. 需求分析和针对设计

**一．职业选择系统**

虽然暂时现在只有两个职业，但未来可能会加入更多职业。在游戏设计中，职业之间的区别通过技能的区别来体现。因此在设计游戏时，考虑到了未来加入新职业的空间。

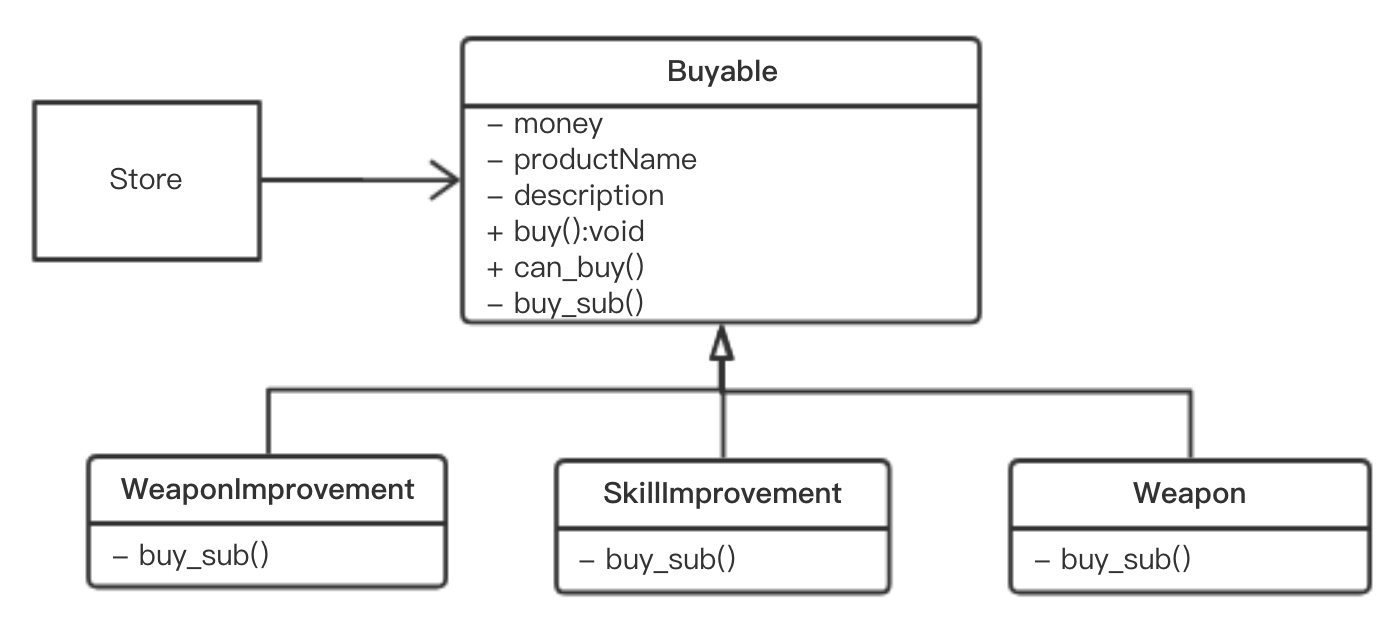
我们使用**策略模式**来实现这一解耦。如下图，公共算法由Player来实现，程序无需了解是哪一种职业，只需执行player的抽象方法即可。在战斗场景中，由BattleUnit来实现高层战斗抽象。



**二．商店系统**

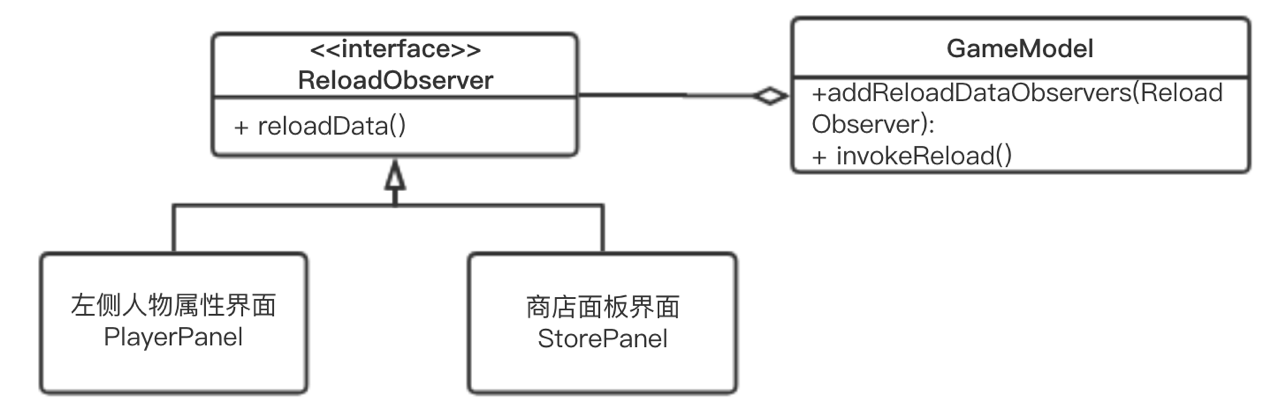
在战斗中获得的金钱，需要商店系统来合理支出。未来可能增加商店中可购买的商品，包括多样的技能升级，装备等，因此商店需要良好的可扩展性。在这里我们以**命令模式和模板方法模式**作为购买商品的架构。

如下图所示，当商店收到对某个商品按钮的点击事件后，会查询对应商品（Buyable）是否canBuy，如果canBuy，会调用商品对象的buy方法。相当于调用者无需知道购买过程的具体实现，只需调用购买方法即可，这就是**命令模式**的实现。



此外，由于buy整个过程分为两个逻辑上不相关的部分：1.player金钱扣减，界面更新，打印日志等所有子类通用的过程 2.实际带来的改变，在这里有三种：技能升级，武器升级和武器购买。所以我们很自然地使用了**模板方法模式**来完成。将各个子类公用的部分（前文所述第一部分）放在模板方法的基类，然后将各个子类不同的部分（前文所述第二部分）放在buy\_sub中，作为抽象方法，延迟到子类中实现。

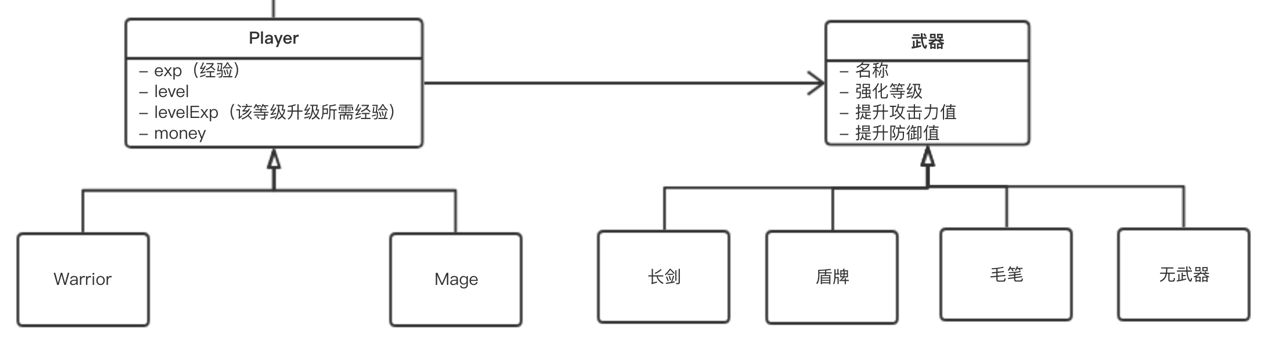
在商店中购买商品后，底层模型更新需要通知上层组件重载数据更新状态，因此很自然地，我们使用了**观察者模式**。在程序内事实上一共定义了三种Observer，分别是LevelChangeObserver（监听楼层及商店跳转），ReloadDataObserver（底层模型更新后，唤起gui组件重载数据刷新界面）和SkillUseObserver（技能释放后唤起组件更新特效）。如下图，在本例中商店购买物品后，GameModel会通知他的Observer去更新界面：1.由于玩家的钱变少了，商店面板界面需要把玩家本来买得起，之后买不起的商品按钮设为不可点击的状态。2. 左侧任务属性界面实时更新人物属性。



**三．装备加成，类型与强化系统**

如果只用人物属性来决定战斗走向过于单调，因此装备加成十分有用，如何把装备加成到人物属性上需要合理的设计。装备的类型虽然现在只有三种，分别是进攻型，防御型和均衡型，不排除未来可能增加更多装备。且对同一件装备，也有不同等级的强化。未来可能增加更多的强化等级。

在装备系统设计上，主要使用了状**态模式，空对象模式**。由于武器在游戏中可以动态更换，也可以通过强化来改变效果，因此不同的武器可以理解为同一个角色的不同状态。如下图，武器的具体的属性延迟到子类中加载，在游戏中通过改变player对象的组合武器来改变player状态。

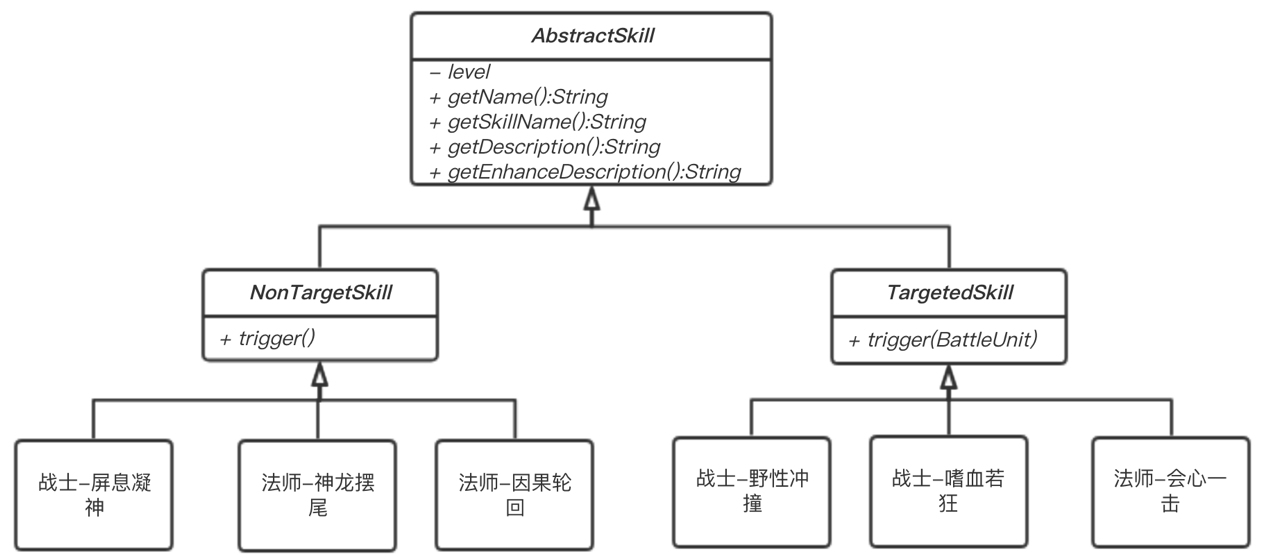


此外，由于玩家刚进入到游戏中是没有武器的，而如果把无武器定义为null的话，会使得程序中出现大量的额外的逻辑判断。因此我们使用**空对象模式**(Null Object Pattern)来更好地实现。在“无武器”对象中，我们把名称设置为“赤手空拳”，伤害加成和防御加成均设置为0.

**四．技能系统**

在技能系统中，技能需要组合使用，技能也可以增强，未来也可能增加新的技能。因此我们要将同一个技能的实现细节封装在一个类中，这样当我们增加或者删除技能的时候，直接编写或删除一个类即可，十分方便。我们主要运用**命令模式**来实现界面触发技能和技能具体实现的解耦。

如下图所示，无目标释放技能和有目标释放技能区别很大，包括在界面实现上也有显著区别（有目标释放技能在点选技能后，需要再选择一个目标，然后才开始执行技能，而无目标技能在点选技能后直接执行技能）。因此我们使用两级继承关系来表示。在命令模式中，界面在收到技能点选事件后，仅需调用技能按钮所绑定的技能对象的trigger方法，即可实现技能。而各个技能通过实现父类的trigger方法来完成技能逻辑编写。



此外，技能释放过程中有许多特效需要触发，底层模型也需要告知界面显示技能效果，因此这里采用了观察者模式来实现底层向高层的调用，避免循环依赖。前文已对观察者模式的应用有所笔墨，具体不再赘述。

**五．每层的怪物设计**

由于层次的数目在未来可能增长，而且每一层的怪物也可能不一样，怪物的属性也可能更改，因此我们需要一种创建型设计模式来完成每一层的怪物的产生。由于游戏预算极低，为了简便，我们选用了**简单工厂模式**。在游戏设定中，我们把每一层称之为一副地图Map，因此每一层的工厂命名为MapFactory，在类MapFactory中，我们实现的方法签名如下：

public static List<Monster> getMap(int level);

输入层数，即可返回该层的Monster列表。

**六．全局对象的创建和维护**

在程序中，涉及到一些全局对象，包括：

\* GameModel：维护游戏当前状态，如Player实例，Monster实例，当前层次等

\* GlobalLogger：由于日志面板是全局不变的，它会忠实地记录所有发生的信息，经过良好的设计后，调用GlobalLogger.log(String info)即可在面板上打印信息

\* Store：商店也是全局唯一的，维护当前存储的商品对象实例。

因此对于这些全局对象的存储，我们使用了**单例模式**来维护，示例代码如下。

public class Store {

private static Store instance;

public static Store getInstance() {

if(instance== null){

instance= new Store();

}

return instance;

}

}

四. 开发细节及配置要求

* 此项目为Java 8开发，请使用Java 8及以上版本运行游戏
* 此项目为**单人**完成
* 此项目不具备联网多人机制

2018.3.5