魔兽世界5文档

xxxxxxxxxxx xxx

# 说明

在对应的文件夹中有类图classdiagram.svg，使用浏览器打开即可查看。

# 需求变化分析

## 需求变化总结

本次需求变化分析基于魔兽世界3进行：

### 战斗

每次战斗只有一个武士发起主动进攻，伤害为攻击力 + Sword攻击力。被攻击者没死就会进行反击，反击伤害为攻击力一半 + Sword攻击力。

战斗中杀死对方武士，司令部可以获取所在城市的所有生命元，除此以外，司令部还会奖励给该武士8个生命元，优先奖励近的武士。司令部先回收生命元再奖励生命元，不够不补发。战平不能拿走生命元。

### 司令部

司令部出现2个敌方武士时，视为被占领。

司令部不会在生命元不足时就此停止制造武士，而是等待足够后再制造。

### 城市

城市可以插旗帜，根据旗帜颜色决定进攻方，如果城市没有旗帜，进攻方的决定方式和以前一样。在某一个城市中，某一方连续胜利两次可以插旗帜，隔着平局就不算连续。旗帜一直插着，一方不会重复插旗帜，直到被另一方更换。

### 武器

Sword初始攻击力为20%武士攻击力，每次战斗一次变成原来的80%，攻击力变为0时，Sword损坏。初始获得攻击力为0的Sword，该Sword视为损坏。没有Sword视为Sword攻击力为0。

拥有Arrow的武士会放箭攻击下一次城市的敌人不被还击，攻击力为定值，Arrow能被使用3次，用完即损毁，相邻城市武士可能同时放箭射死对方。使用Arrow不算胜利也不算平局，不能拿走生命元，也不影响城市旗帜情况。但是如果被射死的城市中有自己的武士，即将要发生战斗，则认为己方获胜，但是不会有主动攻击，反击，战死等事件发生。而战斗之后该发生的还是会发生。

Bomb是一次性消耗品，战斗前先预测自己是否会被战斗杀死，如果会被杀死就与敌人同归于尽，但是不会预测敌方是否使用Bomb。使用Bomb不算胜利也不算平局，不能拿走生命元，也不影响城市旗帜情况。

每种武士最多拥有每种武器各一把。

### 武器报告顺序

报告先从西向东红武士报告，再自西向东蓝武士报告。如果没有某种武器就不用报告。

### 武士

Dragon士气战胜变多0.2，战平变少0.2。欢呼的条件多了“士气大于0.8”这一项。

Ninja挨打不反击。

Iceman每前进两步后使用9点生命值换取20点攻击力，但是生命值不足以扣除时最少减少到1，攻击力照样加。

Lion的武器没了，Lion战死时，战斗前的生命值全部转移到敌人的身上。

Wolf战胜后还是会缴获武器，但是不缴获自已有的武器。

除了Wolf，其余武士胜利后均不缴获武器。

## 以前没有考虑，现在需要考虑的细节

处于敌方司令部的Lion不会逃跑，原来不用考虑是因为司令部只要出现1个敌方武士就被占领了，现在需要出现两个，所以需要修改Lion的代码。

## 不需要考虑的细节

Arrow不能射到司令部内的敌人是不需要考虑的，因为放箭发生在武士前进之后，所以每一轮放箭时司令部内必然没有己方的武士，也就是敌方的武士无法射到己方司令部内的武士，而自己的武士永远无法射到自己司令部内的敌方武士。

汇报武器时会不会出现敌方司令部内有两个武士，两个武士都要汇报呢？不会，汇报武器事件一定发生在武士前进事件之后，如果有两个武士出现的话，游戏已经结束了。

# 需求实现

由于需求是逐步变更的，这里主要说明针对变化的部分的做法，关于魔兽世界3的部分不再赘述。

## 武器

### 武器工厂

武器工厂中的getWeapon方法中，如果发现当前取出的武器是Sword就给这把Sword赋上初始攻击力。

### 武器匣

武器匣内部有一个Map<Class<? extends Weapon>, Weapon>，装着一些武器。

有一个addWeapon方法，可以添加单把武器，添加时如果发现已经有这种武器了就直接返回。

同时还有一个重载的版本，可以添加另外一个武器匣的武器到这个武器匣，也就是循环遍历另一个武器匣的Map，一把一把的取出武器，调用添加一把武器的addWeapon方法即可。

有一个hasWeapon方法，判断当前的武器匣中有没有某种武器，传入的是一种武器的Class，调用Map的contains方法实现。

getWeapon方法首先调用hasWeapon判断有没有某把武器，有就从Map中取出返回，没有就返回null。

toString方法则按照题目的要求，打印Map中的所有武器。

use方法是使用一把武器，首先判断有没有这种武器，如果没有就直接返回，如果有就调用武器的use方法，如果返回值为假，说明武器耐久耗完了。

### Arrow

改变use方法，从百分比伤害变成固定伤害。

重写toString方法，按照题目要求返回信息。

### Bomb

改变use方法，从百分比伤害变成调用攻击者和被攻击的die方法，让他们了直接死亡并善后。

重写toString方法，按照题目的要求返回信息。

### Sword

增加一个attack属性，表示这把剑的攻击力。

use方法中直接让受击方的生命值减少attack即可。同时用完需要将attack的值变为原来的80%，由于剑也有了耐久，所以不能永远返回真了，需要将其修改为返回attack > 0。

重写toString方法，按照题目的要求返回信息。

## GameObject处理内存泄漏问题

之前的版本遗留的内存泄露问题在这个版本解决。在游戏对象基类中添加一个detach方法，默认为空实现，子类需要重写detach方法，并在其中解绑所有其依赖的对象，接触循环引用的情况。这样垃圾回收器就能处理这些垃圾了。

## 建筑物

因为这个版本的城市和司令部的表现很不一样，城市可以插旗帜，司令部不行。城市可以生产生命元，司令部不行。城市不可以产生武士，司令部不行。城市和司令部能处理的事件个数不同，处理相同事件的逻辑也有很大的差别。所以抽象出一个公共的基类Building。

将原来City类中的成员与方法移动到Building中，并让司令部类继承Building类。同时在这个Building类中实现那个detach方法，将后一个城市，前一个城市，以及两个武士的引用都置为null，进行解绑。

acceptEvent中只保留LionRunEvent和ReportWeaponEvent两个事件的处理，其他的代码移动到City类中。

其中因为武器报告的顺序需求改了，所以这里的处理也要做出相应的改变，当武器报告事件发生时，如果事件中的side（武士报告事件的side后面会解释）为red则调用Logger打印redWarrior的武器，否则调用Logger打印blueWarrior的武器。

### 司令部

考虑到效率上的原因，将原来的队列改为了List<WarriorBuilder>，同时新增了一个成员变量i，表示当前轮到列表中的哪一个武士建造者建造武士，默认为0。

修改acceptEvent的代码，首先调用super.acceptEvent，然后在列表中使用get方法拿第i % list.size个武士建造者来建造武士，如果建造的结果为null，表明当前司令部的生命元无法建造武士，直接返回。否则，除了魔兽世界3里的操作外，还需要让武士的headquarter和building指向自己，并且让i + 1。

对于武士行进事件的处理也需要再做修改，增加多一步检查。对于某一方的司令部，在把敌方的武士接过来时，需要查看自身是否已经有一个敌方的武士了，如果有那就是被两个敌方的武士占领了，走魔兽世界3中司令部被占领的逻辑，否则就没有被占领，走魔兽世界3中司令部没有被占领的逻辑。

### 城市

城市类中添加一个旗帜类（旗帜类继承GameObject类，空实现）的对象，代表当前城市中现在的旗帜是什么，为null说明没有旗帜。

添加一个String成员，表示上一次的赢家是哪一方(“red”、“blue”)，默认为null。

添加一个lifeElements，代表这个城市现在有的生命元数量。

增加一个getAttackSide方法，用于决定哪一边先发起进攻，如果当前的旗帜不为空，返回旗帜的选边，如果当前的旗帜为空，按照魔兽世界3的代码决定哪一边先发起进攻。

重写acceptEvent方法，对于各种事件：

#### 城市生产生命元事件

直接让城市的生命元 + 10

#### 拿走城市生命元事件

当这个事件发生时，首先检查这个城市内是否只有一个武士，如果没有武士或者两个武士都在就什么都不做。否则就把lifeElements添加到那个武士的司令部中，然后使用Logger打印拿走生命元的信息。之后再将lifeElements设置为0。

#### 武士射箭事件

首先让红方射箭，调用红方武士的shotArrow方法，如果发现蓝方武士死亡，则调用其setDeadOfArrow，表示该武士死于弓箭。最后调用Logger类打印射箭事件。蓝方射箭的过程同理。注意的是，这里不要直接调用武士的die方法，因为还有战斗胜利之后发生的事件的逻辑需要处理。

#### 城市清除事件

这个事件是自己添加的事件，题目中并没有这样一个事件，所以在这个事件中不会调用Logger类的方法打印任何内容。

这个事件主要是清理因为中箭身亡但所在城市中没有敌人的武士或者处理某个城市中两个武士都因为中箭身亡的情况，否则就没有任何的代码能清理这些中箭身亡的武士了。这些武士不会触发任何“战斗胜利后的事件”，而且必须现在清理，晚清理了会影响剩下程序的逻辑。

这个事件的逻辑非常简单，只需要处理“只有红方存在，红方死亡”、“只有蓝方存在，蓝方死亡”、“双方存在，均死亡”这三种情况的城市即可，分别调用这些武士的die方法即可进行善后处理。

#### 使用炸弹事件

首先调用getAttackSide根据选边决定先攻击方，然后调用先手方的evaluate方法，得到一个int类型的返回值，根据返回值进行拆解，得到红方和蓝方的死亡情况。

先判断红方，如果红方死亡并且红方有炸弹，就调用Logger类打印使用炸弹的信息，之后调用红方的useBomb方法即可。

接着再判断蓝方，逻辑与红方类似。

#### 战斗事件

战斗之前先调用双方武士的beforeFight方法，然后根据getAttackSide的结果决定主动攻击方和反击方，接着把主动攻击方设置到FightEvent的activeAttacker中（FightEvent后面介绍）。

接着如果双方都还活着，就调用Logger类打印主动攻击的信息，然后主动攻击方调用attack方法攻击被攻击方，如果被攻击方没有死亡，就调用Logger类打印被动攻击的信息，然后被攻击方调用fightBack方法对主动攻击方进行反击。

然后输出武士的死亡情况，先判断红武士是否死亡，如果死亡就调用Logger类进行输出，接着再判断蓝武士是否死亡，同理调用Logger类进行输出。（注意，这里没有调用die方法）

接着分别按顺序调用红蓝双方的afterFight方法，进行战斗后的处理。

然后处理旗帜和生命元，如果双方平局，那就把lastWinSide设置为null。否则就调用Logger输出胜利方取走城市生命元的事件然后调用胜利方的getSide方法获得胜利方的选边。

如果胜利方的选边与lastWinSide不等，说明上一次不是本方胜利，直接将lastWinSide设置为胜利方即可，不然就说明上一次也是本方胜利，那就要判断当前城市中插着的是不是自己方的旗帜，如果是就直接返回，如果没有旗帜或者是敌方的旗帜就new一个旗帜出来，将它的side设为胜利方的side，并将这个旗帜赋给城市的flag成员，调用Logger类打印旗帜升起事件即可。

以上的事件都处理完毕后，再次依次判断红方蓝方是否死亡，调用其die方法进行善后即可。

#### 武士前进事件

这部分与魔兽世界3的代码一致，不再赘述。

## 武士

### 武士基类

移除原来武士基类中的武器列表，不需要了，然后添加一个武器匣成员。

移除robWeapon和robWarrior方法，新增addWeapon，hasWeapon方法，这些方法直接调用武器匣成员对应的方法，增加getWeaponPack方法用于获取武器匣。

同时将原来的City的引用改为Building，新增一个指向自己司令部的成员headquarter。

需要判断武士是否死于箭，所以新增一个属性isDeadOfArrow，默认为false。

新增一个getPrize方法，用于从司令部获取生命元奖励，如果司令部生命元够的话就让司令部奖励8个生命元给自己。

由于有点武士打架前需要做事情，打架后需要做事情，所以借鉴SpringAOP的实现，在武士类中添加两个方法，一个是beforeFight，一个是afterFight，分别代表打架前需要做的事和打架后需要做的事，也有点代理模式的意思，但是自己是自己的代理。

在afterFight中判断自己是否还活着，如果还活着并且击杀了对方，就判断自己的选边，如果是红方的武士，就调用FightEvent的addWaitingWarrior方法将自己加入到等待队列中。如果是蓝方的武士，直接调用getPrize方法即可（原因后面解释）。

改写attack方法，首先让被攻击方的生命值减少自身攻击力，然后调用武器匣的use方法，使用Sword攻击敌人（如果没有Sword的情况武器匣会处理好，上面有介绍）。

新增加一个fightBack方法，首先让被攻击方的生命值减少自身攻击力的50%，然后调用武器匣的use方法，使用Sword攻击敌人（如果没有Sword的情况武器匣会处理好，上面有介绍）。

新增加一个evaluate方法，里面是评估是否需要使用炸弹的流程，也就是模拟一次对战。首先创建两个局部变量，赋值为自己的生命值和敌方的生命值，然后从自己和敌方的武器匣中拿出各自的Sword，获得其攻击力，保存到另外两个局部变量中，如果没有Sword视为攻击力为0。紧接着就是用敌方的生命值减去自己的攻击力和自己Sword的攻击力，如果敌方还活着并且不是敌方不是Ninja就发起反击，自己的生命值减去敌方的攻击力和敌方Sword的攻击力。之后再将战斗的结果保存到一个int中返回。（或许可以使用基于序列化和反序列化的备忘录模式做存档？不过这样实现也相当于一个变种的备忘录模式了吧）

增加一个shotArrow方法，调用武器匣的use方法。

增加一个useBomb方法，调用武器匣的use方法。

重写detach方法，把建筑的引用，司令部的引用设置为null。

在die方法内调用自身的detach方法，做好善后工作，防止内存泄露

### Dragon

Dragon重写afterFight，首先调用基类的afterFight方法，之后查看敌人是否还活着，如果活着自己的士气减少0.2，否则自己的士气增加0.2，最后就是查看自己是不是还活着，是不是主动攻击方，士气是否大于0.8，如果都满足，就调用Logger类打印欢呼信息。

Dragon的getBornMessage中增加打印士气的语句，同时使用Settings类中的浮点数格式化器来格式化士气并返回（这个格式化器后面介绍）。

### Iceman

在Iceman中增加一个counter成员，代表自己已经走了多少步。

修改原来的acceptEvent方法，每次行进时向将counter + 1，然后如果发现counter是偶数就将自身的生命值减少9，攻击力加20，但是如果发现减少到0或以下，就将生命值重新设置为1。

### Lion

在Lion中增加一个lifeBeforeFight成员，代表某次战斗前自己的生命值。

重写beforeFight方法，每次战斗前先调用基类的beforeFight，然后将自己的生命值记录到lifeBeforeFight成员中。

重写afterFight方法，每次战斗后先调用基类的afterFight，然后如果自己死了，敌方武士活着，并且自己不是被弓箭射死的（武士战死事件不会发生），就让敌方武士的生命值加等于lifeBeforeFight。（也是备忘录模式的一种体现）如果自己没有战死，但是敌方也没死，那就将自己的忠诚度减少一定的数值。

修改acceptEvent方法中对狮子逃跑事件的处理逻辑，如果发现当前所在的建筑是司令部并且选边与自己的选边不同就不触发逃跑逻辑，其余逃跑部分的代码同以前的代码。

### Ninja

重写fightBack方法，将基类的实现变为空实现，因为忍者无法反击

### Wolf

重写afterFight方法，首先调用基类的afterFight方法，接着如果自己没有死亡，敌方武士死亡，就获得敌方的武器匣，然后调用自己的addWeapon方法缴获其武器匣中的武器缴获。

## 事件的变化

### 事件基类

hour成员变量默认值设置为0，将原来的空参构造方法变为带一个参数的构造方法，将minute成员变量赋值为这个参数的值。所有的Event子类都提供一个无参数的构造方法，然后调用super(x)，设置minute变量的值，表示这个事件是哪一分钟发生的。

给handle方法提供默认实现，每次调用handle时将hour + 1，minute不变，handle不再具有返回值，不再具有参数。所有的子类的handle方法也随之修改，不再判断轮没轮到这个事件发生，而是直接执行这个事件发生时的逻辑，并且所有的子类都需要先执行事件基类的handle方法（super.handle）。

### 城市清理事件

直接调用事件系统通知所有的城市即可。

### 城市生产生命元事件

直接调用事件系统通知所有的城市即可。

### 取走城市生命元事件

直接调用事件系统通知所有的城市即可。

### 战斗事件

增加一个activeAttacker的变量，表明在某一场战斗中主动攻击的一方（解释上面城市中的战斗前设置主动攻击者的操作，这将影响到部分输出，比如在龙的afterFight方法中会获取这个变量的值，判断它自己是否是主动攻击者而决定是否欢呼）。

增加一个waitingWarrior成员变量，是一个栈式的结构，用于存放红方战斗胜利后的武士，这么做是因为题目说优先奖励距离敌方司令部近的武士，因为我们是从西向东发生战斗的，所以红方的武士需要进入等待栈中等待后续处理，而蓝方的武士可以立即获得奖励。

修改handle方法，首先调用事件系统通知所有的城市本事件。在这之后，因为战斗完成了，所以这时候可以奖励在waitingWarrior中的红武士了，因为是栈结构，所以倒着拿出来，符合题目要求的顺序，依次调用其getPrize方法即可。接着需要new一个取走城市生命元事件，然后设置这个新的城市生命元事件的minute为35，这样就可以和20分钟发生的那个事件分开，Logger检测到这个事件发生的事件不在20分时就不会打印对应的信息。（题目只要求打印20分的那个取走生命元的信息，不要求打印战胜后回收生命元的信息）。接着把这个新产生的事件通过事件系统通知给所有的城市即可。

### 报告武器事件

首先新增一个成员变量为side，示意现在应该轮到哪一方汇报武器。（解释上面建筑物处理这个事件的逻辑）

首先将side设置为“red”，然后使用事件系统通知所有的建筑物，然后再将side设置为“blue”，然后使用事件系统通知所有的建筑物。

### 使用炸弹事件

直接调用事件系统通知所有的城市即可。

### 武士放箭事件

直接调用事件系统通知所有的城市即可。

## 时钟系统

时钟系统做了一些调整，与司令部装武士建造者的形式类似，使用了一个List<Event>，而不是Queue。

提供一个addEvent方法，可以添加event到List中。同时去除elapse变量，不需要了，添加一个gameEnd变量，表示当前游戏是否已经结束，默认为假。

改写acceptEvent方法，当接收到EndEvent时将其设置为真。

重写start方法，首先一开始将自己注册到事件系统，定义一个变量i，默认为0，表示现在轮到哪一个事件发生。

一直循环，直到gameEnd为真或者第i个事件的hour \* 60 + minute超出模拟的时间，循环体内每次都从列表中取出第i % list.size个事件，调用其handle方法，然后将i + 1，循环继续。

## 设置类

新增一个设置静态类，主要保存游戏的各种给定的数据。

增加一个静态成员变量，表示狮子走一步忠诚度的衰减值。

增加一个静态成员变量，表示弓箭的攻击力。

增加一个静态成员变量，表示游戏的模拟结束事件。

增加一个静态成员变量浮点数格式化器，用于给龙的士气格式化成符合题目要求的字符串。在静态代码块中配置它保留小数点后两位、舍入规则为四舍六入五留双（这种模式就是对于**最后一位**刚好为5的数，上一位舍入到偶数位，否则就正常的舍入，C++的setprecision也是这个原理）。

关于四舍六入五留双的例子：

0.5 保留小数点后0位变成0，因为5是最后一位，上一位舍入到最近的偶数0。

0.25保留小数点后1位变成0.2，因为5是最后一位，上一位舍入到最近的偶数2。

0.125保留小数点后2位变成0.12，因为5是最后一位，上一位舍入到最近的偶数2。

0.625保留小数点后2位变成0.62，因为5是最后一位，上一位舍入到最近的偶数2。

0.875保留小数点后2位变成0.88，因为5是最后一位，上一位舍入到最近的偶数8。

以上五个都是2的n次幂的线性组合，比如0.5是2的-1次方，0.25是2的-2次方，0.125是2的-3次方，0.625 = 0.5 + 0.125，0.875 = 0.5 + 0.25 + 0.125，能够被计算机精确表示，不会出现精度缺失的问题。

有些数看上去是5结尾，但是实际上却不是，因为这些数不能被计算机精确表示：

比如0.165，不是2的幂次的线性组合，实际上是0.16500000000000000777，保留小数点后2位数字时，最后一位不是5而是5.00000000000000777，根据四舍六入五留双的规则，这个数比5大，所以需要所以要进位变成0.17。

又比如0.1025，不是2的幂次的线性组合，实际上是0.10249999999999999389，保留小数点后3位数字时，最后一位不是5而是4.9999999999999389，根据四舍六入五留双的规则，这个数比5小，所以需要所以要舍弃变成0.102。

总结就是在考虑浮点数进度问题的前提下，保留小数点后x位时，如果从第x+1位数字开始往后正好是5（也就是能被计算机精确表示的浮点数，2的幂次的线性组合），第x位就舍入成偶数。如果从第x+1位数字开始往后比5小，哪怕只是小一点点，x就舍弃。如果从第x+1位数字开始往后比5大，哪怕只是大一点点，x就进位。屡试不爽！

## 武士建造者

因为后期要采用XML配置文件的方式初始化游戏环境，所以这里要对武士建造者进行一番改造。

首先在武士建造者基类中增加一个List<SimpleEntry<Integer, Integer>>成员，表明这个武士在刚开始时拥有什么武器，以键值对的形式保存增加的值和取模的值，比如题目说龙刚开始有n % 3这种武器，那就在里面存入一个(0, 3)，比如忍者有n % 3 和 （n + 1） % 3这两把武器，那就在里面存入一个(0, 3)再存入一个(1, 3)。

提供一个addInitialWeapon方法用于添加某个武士初始拥有的武器的增加的值和取模的值，其实也就是调用上面的List的add方法，添加时使用比如addInitialWeapon(0, 3)的语句会就在List里面存入一个(0, 3)。

添加一个equipWeapon方法，遍历初始武器List中的所有键值对，然后根据当前生成的武士的编号调用武器工厂的getWeapon方法从里面拿出对应的武器，拿出来时需要检查一下是不是Sword，如果是的话检查其攻击力是否为0，不为0才调用addWeapon方法将其添加到武士的武器匣中，否则就直接返回。如果不是Sword就直接调用addWeapon方法将其添加到武士的武器匣中。

接着把所有武士建造者子类中的给武士装配武器的代码删除。

最后修改基类的buildWarrior方法，在constructWarrior方法调用完成后再调用一下equipWeapon即可。

## XML配置文件

使用XML配置文件来管理游戏里的部分初始信息，达到解耦的目的。整个XML的根标签是<GameSettings>，根标签下有如下标签：

### <Weapon>

这个标签下主要存放所有的武器，以及这些武器的全类名。里面标签的存放顺序就是武器的顺序，比如Sword作为第1个子标签就是0，Bomb作为第二个子标签就是1。子标签的tag就是武器的名字，而其中的ClassPath属性指明了这把武器的全类名。

### <WarriorBuilder>

这个标签下存放着所有的武士建造者，以及建造武士时，需要给武士添加的初始的武器的情况。

第一层子标签为各个武士建造者的子类，并且其属性ClassPath中指明武士建造者的全类名。在每一个一层子标签下有一个<Weapons>的二层子标签，而这个<Weapons>二层子标签下存储的就是这个武士初始拥有的所有武器的增加的值和取模的值。比如<Weapon Add=“1” Mod=“3” />就是代表(n + 1) % 3，比较直观。

比如忍者的部分长这样：

<WarriorBuilder>  
 <NinjaBuilder ClassPath="com.pikacat.builder.NinjaBuilder">  
 <Weapons>  
 <Weapon Add="0" Mod="3"/>  
 <Weapon Add="1" Mod="3"/>  
 </Weapons>  
 </NinjaBuilder>

</WarriorBuilder>

代表有忍者建造者这个建造者子类存在，并且全类名为com.pikacat.builder.NinjaBuilder，忍者这种武士刚开始有2把武器，分别为n % 3 和 (n + 1) % 3。

### < Headquarter>

这个标签内存放着所有的司令部建造武士的顺序，里面有<Red>和<Blue>两个子标签，分别存放红方和蓝方司令部建造武士的顺序。形如：

<Headquarter>  
 <Red>  
 <IcemanBuilder/>  
 <LionBuilder/>  
 <WolfBuilder/>  
 <NinjaBuilder/>  
 <DragonBuilder/>  
 </Red>  
 <Blue>  
 <LionBuilder/>  
 <DragonBuilder/>  
 <NinjaBuilder/>  
 <IcemanBuilder/>  
 <WolfBuilder/>  
 </Blue>  
</Headquarter>

### <Event>

这里按事件发生的顺序装着所有的事件以及它们的全类名，简单明了，形如：

<Event>  
 <BornEvent ClassPath="com.pikacat.event.BornEvent"/>  
 <LionRunEvent ClassPath="com.pikacat.event.LionRunEvent"/>  
 <WarriorMarchEvent ClassPath="com.pikacat.event.WarriorMarchEvent"/>  
 <CityProduceLifeElementsEvent ClassPath="com.pikacat.event.CityProduceLifeElementsEvent"/>  
 <TakeCityLifeElementsEvent ClassPath="com.pikacat.event.TakeCityLifeElementsEvent"/>  
 <WarriorShotEvent ClassPath="com.pikacat.event.WarriorShotEvent"/>  
 <CityCleanEvent ClassPath="com.pikacat.event.CityCleanEvent"/>  
 <UseBombEvent ClassPath="com.pikacat.event.UseBombEvent"/>  
 <FightEvent ClassPath="com.pikacat.event.FightEvent"/>  
 <ReportElementsEvent ClassPath="com.pikacat.event.ReportElementsEvent"/>  
 <ReportWeaponEvent ClassPath="com.pikacat.event.ReportWeaponEvent"/>  
</Event>

## Configuration配置类

这个类采用了DOM方法加载了上述的配置文件，并根据上述的配置文件来配置整个游戏逻辑的运行。

里面有一个重要的成员变量是Map<String, WarriorBuilder>，里面存储着某个武士建造者名字和类的对应关系。因为插入的顺序十分重要，与题目的顺序有关，所以这里采用LinkedHashMap。

其中一些解析DOM的代码不细说，这里主要说一下游戏配置的过程。

### 静态代码块

该类的静态代码块读取XML配置文件（config.xml），并建立DOM树以备解析。

### initGame方法

这个方法主要是做全局的初始化首先解析XML中Weapon的部分，使用反射加载这些武器的字节码，并创建对象，将其添加到WeaponFactory中（调用addWeapon方法）

接着解析XML中的WarriorBuilder的部分，使用反射机制加载这些建造者的字节码，并创建对象，同时解析WarriorBuilder标签下的Weapons标签，获取Add和Mod属性的值，并调用武士建造者的addInitialWeapon方法，为武士建造者添加默认的武器信息。最后将这些对象保存到本类的Map<String, WarriorBuilder>中，以便后续使用。

### configGame方法

这个方法主要是根据每一次的输入，来配置游戏。

其实与原来的Main方法中做的事情类似，代码也是直接拿过来稍作修改，主要说一下变化的部分。

首先把一些内容读入到Settings类中，供其他类使用，比如结束事件，箭的攻击力，狮子的衰减。

原先一个一个武士建造者配置初始生命值和攻击力的部分代码现在改为从解析好的Map<String, WarriorBuilder>中读出来，按顺序调用setter方法设置进去即可。（前面使用LinkedHashMap的好处在这里体现）

原先硬编码的给红蓝方司令部一个一个添加武士建造者的代码改成解析XML中<Headquarter>中的<Red>和<Blue>中的标签，按标签的顺序和内容从解析好的Map<String, WarriorBuilder>中读出对应的武士建造者添加到司令部中。

最后构造Clock类的对象，并将在Clock类中硬编码的一个一个手动添加事件的代码改为解析XML中<Event>的标签，按标签的顺序和每个标签的类名构造事件对象并调用Clock类的addWeapon方法将其添加到时钟对象中。

最后将建筑物列表（原来Main类中的cityList）和时钟对象作为返回值返回。

## 事件系统

事件系统主要是防止内存泄漏做的修改，在reset方法被调用时调用所有注册到系统的对象的detach方法，最后才将它们清除出事件系统。

## Main方法

重写Main方法，一开始调用Configuration类的initGame方法初始化游戏。然后正常读入case个数，接着在每个循环内部调用Configuration类的configGame方法配置游戏。然后调用其返回时钟的start方法即可，模拟结束后start方法返回，最后还是一样的调用EventSystem的clear方法和cityList的clear方法做清理收尾工作，避免内存泄漏即可。

# 运行时间问题

如果把所有的System.out.println和System.out.printf注释掉，可以发现程序的运行时间只有200多毫秒，可知剩余的时间都花费在了输出上，所以这是没有办法解决的难题。

文本

描述已自动生成