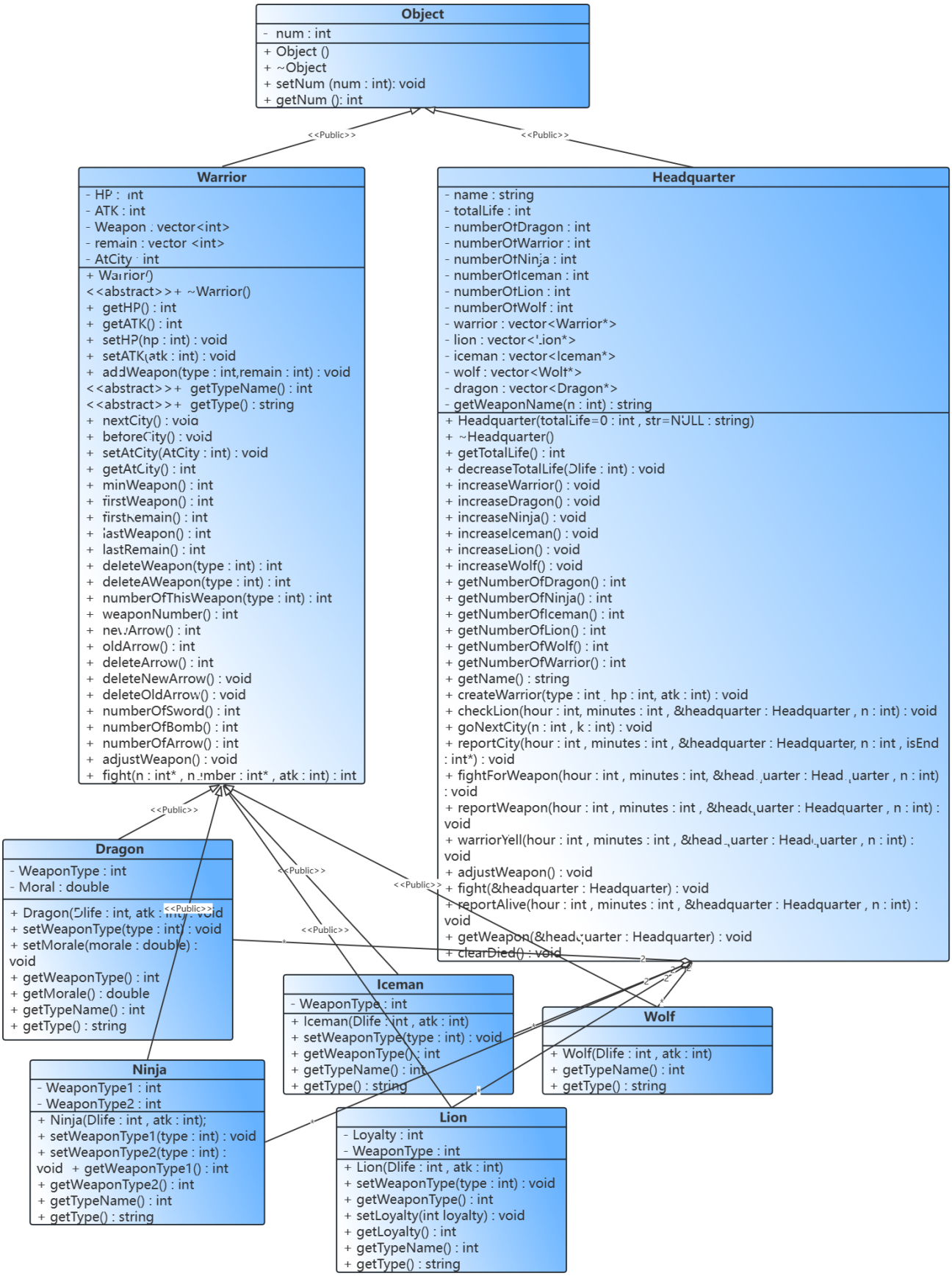


**注意：这里类中的某些函数和变量是用于难度一和难度二的，但是在难度三中为体现，但是因为一二三是继承关系，所以难度三的代码是在难度二的基础上增添而来，所以有些函数和变量未被使用**

## 1.2框架详细解释

Object类：所有类的共同的基类，提供最基本的特性——编号

Warrior类：所有不同种类的士兵的共同基类，继承了Object类，包含了士兵的相同的功能，如：生命值、攻击力、武器、武器剩余使用次数、所在城市，以及对于这些变量的相关操作（因为这些变量是private的，所以必须要创建public的相关函数对其进行操作），如：攻击、增加武器、减少特定武器、减少生命值、前往下一个城市、获取第一个武器的类型、获取最后一个武器的类型、获取第一个武器的剩余使用次数、获取最后一个武器的剩余使用次数，但是这个类是一个抽象类，其中的某些函数并没有实现，如：获取士兵的种类，因为Dragon、Ninja、Wolf、Lion、Iceman的种类并不相同，所以无法在基类中给出特定的函数，因此设计成纯虚函数，所以改类为虚基类。

（注意：武器可以单独设计成一个武器类，其中包含武器的种类和武器的剩余使用次数，但是这里为了方便起见并没有单独设计和封装武器类，而是在士兵类中设置了两个vector数组，一个用于存放武器种类，另一个用于存放武器的使用次数，这里的两个数组虽然是独立的，但是在操作过程中均保证了角标的一致性）

（这里如果对武器进行封装，优点是可以重载武器的<运算符，这样可以更加方便的进行排序，但是缺点是为了将武器类保护起来，武器的种类和剩余的使用次数必须要private，所以士兵类无法直接获取相关的信息，需要格外新增一个函数来获取相关信息。而不对武器进行封装，虽然需要直接进行排序，但是士兵可以直接访问武器的相关信息。这里虽然看起来是没有封装起来，但是因为士兵类也是经过了封装的，所以也可以对武器的种类和使用次数进行保护）

Dragon类：继承于Warrior类，其中重写了获取士兵种类的函数、Dragon的构造函数、Dragon的士气值

Ninja类：继承于Warrior类，重写获取士兵种类的函数、Ninja的构造函数

Iceman类：继承于Warrior类，重写获取士兵种类的函数、Iceman的构造函数

Lion类：继承于Warrior类，重写获取士兵种类的函数、Lion的构造函数、Lion的忠诚度

Wolf类：继承于Warrior类，重写获取士兵种类的函数、Wolf的构造函数

Headquarter类：继承于Object类，其中包含了vector数组用于存储所有士兵的地址值，（这里用到的是将子类的指针转换成基类的指针，并且储存起来），为了方便起见，额外设置了不同种类的士兵的vector数组，在一些特殊的情况，比如Lion逃跑、Wolf抢夺武器时，直接使用，一定程度上通过内存的增加、减少了代码的运行时间。指挥部类中还包含了士兵的绝大部分的调用操作，因为士兵的任何举动都是指挥部进行调度的，所以在指挥部类中，含有了检查Lion忠诚度、派遣士兵前进、要求士兵对武器排序、开始战斗、汇报战斗情况、清除死亡士兵等诸多有关士兵调用的关键函数。而在这些函数中，又大量的用到了Warrior类和特定士兵类的众多函数，实现了代码的关联。

main函数：main函数中只是简单的包含了项目的输入，创建指挥部，依次创建特定类型的士兵、时间的流逝，再特定的时间上，要求指挥部对士兵发起不同的指令，以及判断程序是否结束等相关的操作。

在难度三中：game.h文件只是包含了该项目中的所有的需要的头文件，让程序更加简单。

在图形化中：game.h文件中包含了大量有关图形化的相关操作，包括不同界面的跳转，图片的绘制，甚至是整个程序的进行过程也从main函数中，转移到game.h头文件中

# 2、程序完成的主要逻辑（难度三）

## 2.1士兵创生

士兵创建时，由main函数的相关函数确定应当生成的士兵种类，并且确定能否继续生成（如果本次无法生成士兵，那么以后就都不能创建士兵了），并且根据设定的相关生命值和攻击力创建对应的士兵，当士兵创生时，指挥部也会减少相关的生命元的数量。

士兵生成仅以红方为例：

```
if(red.getTotalLife()>=life[r-1] && RedContinue){
   cout<<setw(3)<<setfill('0')<<hour<<":"<<setw(2)<<setfill('0')<<minutes;
   red.createWarrior(r,life[r-1],atk[r-1]);
   r= nextR(r);
}else{
   RedContinue=false;
}
```

只有红色可以继续生成士兵且目前的生命元的数量大于产生士兵数，才可生成士兵且将士兵种类切换为下一个，如果不能生成的话，就将bool型的判断能否继续生成士兵的变量设置为false

以下为红方切换生成士兵的函数：（1，2，3，4，5依次代表Dragon Ninja Iceman Lion Wolf)

```
int nextR(int r){
    if(r==3){
        return 4;
    }else if(r==4){
        return 5;
    }else if(r==5){
        return 2;
    }else if(r==2){
        return 1;
    }else{
        return 3;
    }
}
```

蓝方也是同理，这里不再赘述

士兵创建的函数如下：（以产生Dragon为例）

```
Dragon *p = new Dragon(hp,atk); //创建武士
warrior.push_back(p);  //武士储存
dragon.push_back(p);
decreaseTotalLife(hp);  //扣除生命元
increaseDragon(); //数量增加
increaseWarrior();  //数量增加
p->setWeaponType(numberOfWarrior%3); //设置武器
p->setMorale(1.0*getTotalLife()/hp);  //设置士气值
p->setNum(getNumberOfWarrior());  //设置编号
p->setAtCity(0);  //设置城市位置
cout << " " << getName() << " dragon " << getNumberOfWarrior() << " born" << endl;
break;
```

**在针对于难度三的源代码解析中，后面如果不加特殊说明，均为指挥部Headquarter类的成员函数**

## 2.2狮子逃跑

因为规定输出结果是按照城市从左到右依次输出、红方比蓝方先输出，所以第一层for循环是遍历所有的城市，第二层for循环先遍历所有红方的狮子、再遍历所有蓝方的狮子，查看是否有在目标城市，如果有，检查忠诚度，如果为负，那么狮子逃跑，在指挥部中删除对应的狮子和士兵

核心代码如下：

```
if(p->getLoyalty()<=0){//如果忠诚度小于等于0
    cout<<setw(3)<<setfill('0')<<hour<<":"<<setw(2)<<setfill('0')<<minutes;
    cout<<" "<<getName()<<" lion "<<p->getNum()<<" ran away"<<endl;
    lion.erase(lion.begin()+i);//删除lion
    i--;//遍历目标递减
    numberOfWarrior--;//数量减少
    numberOfLion--;
    for(int j=0;j<warrior.size();j++){
        Lion* q=dynamic_cast<Lion*>(warrior[j]);
        if(q->getNum()==p->getNum()){ //在warrior中找到对应的lion
            warrior.erase(warrior.begin()+j);//删除warrior的lion
            break;
        }
    }
}
```

**这里需要特别注意的就是，从遍历的列表中删除某一元素后，一方面需要将new出来的内存释放（vector自动释放），另外一方面，还需要将循环的结束的判断的变量的大小-1（如果不减1会数组会越界）、遍历的元素i-1（因为这里已经把第i-1个元素给删除了，原本的第i个元素称为了第i-1个，如果后面还要继续遍历的话，i++之后，会把现在所处的角为i的元素跳过了，这样遍历就少遍历了一个元素），后面也有相似的内容，也不再解释和介绍了**

**这里所有的从warrior*中获得的元素，为了安全起见采用了dynamic_cast转换成了所需类型，因为创建对象时，一方面这个元素本身就是子类的对象，所以通过多态的形式将子类的指针转换成基类的指针，并且存储在基类指针的数组中，另一方面，很容易发现，warrior这个类为一个抽象类，所以不可能直接创建出这个类的对象，所以这种转换一定是安全的。实际上，在后面的操作中，可以发现，我们并不需要进行这种转换，因为这里的所有用到的对象都是基类中声明出来的，而子类中特有的很多功能，在这里甚至是后面都没有体现到，所以说这里根本就不需要将基类的指针转换为子类的指针，后面也不再过多的解释**

## 2.3士兵前进

**所有遍历城市、士兵规则都是按照2.2中描述的方法进行，后面不再过多赘述，代码也过于冗长，也不在这里展示。**

前进的规则就是根据指挥部的颜色，判断是向右还是向左，这里的向左是指前往编号小的城市，如果到指挥部（到红方指挥部，这里设置为-1，到蓝方指挥部设置成n+1，n为城市数量，初始化的城市都是0，根据指挥部的颜色设置前往的第一个城市是1还是n），就程序结束

核心代码如下：

```
if(p->getAtCity()==0){//如果是0，设置初始位置
    if(this->name=="red"){
        p->setAtCity(1);
    }else if(this->name=="blue"){
        p->setAtCity(n);
    }
}else if(this->name=="red" && p->getAtCity()==n){//如果到蓝色指挥中心，设置成n+1
    p->nextCity();
}else if(this->name=="blue" && p->getAtCity()==1){//如果到红色指挥中心，设置成-1
    p->beforeCity();
    p->beforeCity();
}else{
```

```
    if(this->name=="red"){
        p->nextCity();
    }else if(this->name=="blue"){
        p->beforeCity();
    }
}
```

这里只是个人的设置问题，创建士兵时，所在城市都是0，然后根据指挥部的类型确定下一个城市在哪，以及是向左还是向右

```
for(int j=0;j<lion.size();j++){
    Lion* q=lion[j];
    q->setLoyalty(q->getLoyalty()-k);//lion的忠诚度扣除
}
for(int j=0;j<iceman.size();j++){
    Iceman* q=iceman[j];
    q->setHP(q->getHP()-int(q->getHP()*0.1));//iceman的生命值扣除
}
```

后面还需要对Lion的忠诚度、Iceman的血量进行扣除

## 2.4抢夺武器

因为考虑到这里武器并没有进行排序，所以不能够直接获取第一个武器，需要调用获取编号最小的武器的函数获取编号最小的武器，然后抢夺过来后，先分两种情况，一种情况就是编号为0和1的武器，这两种较为简单，只需要比较跟剩余可装备武器的数量，如果剩余可装备数量多，则全部装进去，否则，补满武器即可。如果编号是2的武器，那么需要将剩余可装备武器数量与没用过的和所有该类型的武器总和进行比较，如果说剩余可装备的武器数量多，那么全部装进去即可，如果可装备武器数量比没用过的都少，那么就之装备没用过的，且装满为止，如果在这两者之间，就要把没用过的全部装进去，用过的装满。

在调用函数的时候，保证每次只装备和删除一个武器，即删除第一次出现的目标武器（武器类型和武器剩余可使用次数都符合要求）

武器类型不是2的如下：

```
int number=0;
if(q->weaponNumber()<=0) //获取武器的数量
    break; //如果对面没有武器，就直接放弃
int type=q->minWeapon(); //获取对面编号最小的武器
if(type==3) //没有3这个武器，进一步保证安全性
    continue;//其实break是一样的，因为一个城市最多只能有俩士兵
if(type==0 || type==1){ //如果不是2号武器
    int get=q->numberOfThisWeapon(type); //对面的武器减少
    int beforeHave=p->weaponNumber(); //先获取我方拥有的武器数量
    if(10-beforeHave>=get){  //如果还能装备的武器数量比抢来的多
         number=get; //就直接抢完所有的
    if(type==0){ //如果是0
         for(int w=0;w<get;w++){
            q->deleteAWeapon(type);
            p->addWeapon(0,99999); //可以无限次使用
        }
```

```
    }else if(type==1){
        for(int w=0;w<get;w++) {
            q->deleteAWeapon(type);
            p->addWeapon(1, 1);  //只能使用一次
        }
    }
}else{  //如果能装备的数量比抢到的要少
    number=10-beforeHave;  //看能抢到多少
    if(type==0){
        for(int w=beforeHave;w<10;w++) {
            q->deleteAWeapon(type);
            p->addWeapon(0, 99999);
        }
    }else if(type==1){
        for(int w=beforeHave;w<10;w++){
            q->deleteAWeapon(type);
            p->addWeapon(1,1);
        }
    }
}
```

武器类型是2的如下：

```
int newArrow=q->newArrow(); //获取没有用过的数量
int oldArrow=q->oldArrow(); //获取用过一次的数量
int beforeHave=p->weaponNumber(); //看看还能装备多少
if(10-beforeHave>=newArrow+oldArrow){ //如果还能装备的比新的加旧的还多，就直接装备
    number=newArrow+oldArrow;
    for(int w=0;w<newArrow+oldArrow;w++){
        int remain=q->deleteArrow(); //删除对方的武器，并且获得剩余的使用次数
        p->addWeapon(2,remain);
    }
}else if(10-beforeHave<=newArrow){ //如果剩余装备的比新的还少，只用新的装满就可以
    number=10-beforeHave;
    for(int w=0;w<10-beforeHave;w++){
        q->deleteNewArrow();
        p->addWeapon(2,2);
    }
}else if(10-beforeHave>newArrow && 10-beforeHave<newArrow+oldArrow){ //如果在这两者之间
    number=10-beforeHave;
    for(int w=0;w<newArrow;i++){ //先让新的装满
        q->deleteNewArrow();
        p->addWeapon(2,2);
    }
    for(int w=0;w<10-p->weaponNumber();w++){ //剩下的用旧的装
        q->deleteOldArrow();
        p->addWeapon(2,1);
    }
}
```

## 2.5整理武器

整理武器就是进行两次排序，第一次排序是按照武器的编号进行冒泡排序，需要注意的是，武器编号的vector和剩余使用次数的vector需要同步进行移动；第二次排序是按照武器剩余使用次数进行冒泡排序，就是只需要比较编号是2的（这个时候，编号为2的武器一定是在数组的最后）武器，按照剩余次数进行排序即可，需要注意的点仍然同上。

代码如下：（这里的整理武器的代码是在warrior类中的，是可以直接操控武器数组的）

```
void Warrior::adjustWeapon() {
    for(int i=0;i<weapon.size();i++){
        for(int j=0;j<weapon.size()-1;j++){
            if(weapon[j]>weapon[j+1]){
                int w=weapon[j+1];
                int u=remain[j+1];
                weapon[j+1]=weapon[j];
                remain[j+1]=remain[j];
                weapon[j]=w;
                remain[j]=u;
            }
        }
    }
    int m=numberOfArrow();
    int n=weapon.size();
    for(int i=n-m;i<n;i++){
        for(int j=n-m;i<n-1;i++){
            if(remain[j]>remain[j+1]){
                int w=weapon[j+1];
                int u=remain[j+1];
                weapon[j+1]=weapon[j];
                remain[j+1]=remain[j];
                weapon[j]=w;
                remain[j]=u;
            }
        }
    }
}
```

在前面也已经分析过了，如果设计武器类的话，可以对武器类重载"<"和">"运算符，从而使用sort函数对武器进行排序，运算符重载如下（虽然代码中并没有这么做）：

```
bool weapon :: operator < (weapon &p){
    if(this->type < p.type){
        return true;
    }else if(this->type == p.type){
        if(this->remain < p.remain){
            return true;
        }else{
            return false;
        }
    }else{
        return false;
    }
}
```

## 2.6开始战斗

同样是从城市开始遍历，如果双方都有士兵的话，开始战斗，根据城市的编号确定先后手，先获取战斗前的生命值，然后开始战斗，每次使用武器后，指针向后指一个，如果到了最后，武器重新指向第一个。进行一轮战斗后，重新确定生命值，如果生命值减少，则刷新；如果生命都没有减少，则记录起来，如果生命值不变的次数大于武器的个数，则表明双方都无法让对面受伤，平局；如果战斗中，出现一方的生命值小于等于0，则战斗停止。

以红方先攻为例，p为红方士兵，q为蓝方士兵，代码如下：

```
int RWeaponNumber = p->numberOfArrow() + p->numberOfBomb() + p->numberOfSword();//获取武器数量
int RNowWeapon = 0;
int BWeaponNumber = q->numberOfArrow() + q->numberOfBomb() + q->numberOfSword();
int BNowWeapon = 0;
int RReAtk=1; //获取是否具有反噬伤害
int BReAtk=1;
    if(p->getTypeName()==2){  //如果是Ninja无反噬
        RReAtk=0;
    }
    if(q->getTypeName()==2){
        BReAtk=0;
    }
int redHP=p->getHP();
int blueHP=q->getHP();
int same=0; //判断双方生命值不变的轮数，如果超过一定的轮数（最大的武器数量），认定为平
while (1) {//循环攻击
    if(RWeaponNumber>0)//红方武器数量不为0
        q->setHP(q->getHP() - p->fight(&RNowWeapon, &RWeaponNumber, RReAtk));//红方攻击
    if (q->getHP() <= 0)//蓝方是否死亡
        break;
    if (p->getHP() <= 0)//红方是否死亡
        break;
    if(BWeaponNumber>0)//蓝方武器数量不为0
       p->setHP(p->getHP() - q->fight(&BNowWeapon, &BWeaponNumber, BReAtk));//蓝方攻击
    if (p->getHP() <= 0)//红方是否死亡
        break;
    if (q->getHP() <= 0)//蓝方是否死亡
        break;
    RNowWeapon++;//下一个武器
    BNowWeapon++;//下一个武器
    if (RNowWeapon >= RWeaponNumber)//武器是否重置
        RNowWeapon = 0;
    if (BNowWeapon >= BWeaponNumber)//武器是否重置
        BNowWeapon = 0;
    if(redHP==p->getHP() && blueHP==q->getHP()){//如果双方生命值都不变
        same++;
    }else{
        redHP=p->getHP();
        blueHP=q->getHP();
        same=0;
     }
```

```
        if(same>RWeaponNumber&&same>BWeaponNumber){//如果经过了最大武器数量的轮数，生命值都不变，则表示攻击力过低，双方无法造成伤害平局
            break;
        }
    }
```

士兵攻击的代码如下（在士兵类中）

```
int Warrior::fight(int *n,int *number,int atk) {
    int type=Weapon[*n];
    if(type==0){
        return int(ATK*0.2);
    }else if(type==1){
        remain[*n]--;
        remain.erase(remain.begin()+*n);
        Weapon.erase(Weapon.begin()+*n);
        (*number)--;
        (*n)--;
        this->HP-=int(atk*int(ATK*0.4)*0.5); //反噬的生命值
        return int(ATK*0.4);
    }else if(type==2){
        remain[*n]--;
        if(remain[*n]==0){
            remain.erase(remain.begin()+*n);
            Weapon.erase(Weapon.begin()+*n);
            (*number)--;
            (*n)--;
        }
        return int(ATK*0.3);
    }
}
```

这里根据武器的类型进行攻击，返回值是对对方造成的伤害，对自己的伤害在这个函数中可以直接扣除（因为是自己发动的攻击，可以直接设置自身的变量）

### 2.7武士欢呼

因为这个时候死亡的士兵并没有清除，并且只有经过战斗且存货的Dragon才会欢呼，所以只有类型是Dragon且生命值大于0，且存在敌方士兵（无论是否死亡）的才会欢呼。

核心代码如下：

```
for(int i=0;i<dragon.size();i++){
    Dragon* p=dragon[i];
    for(int j=0;j<headquarter.warrior.size();j++){
        Warrior* q=headquarter.warrior[j];
        if(p->getAtCity()==t+1 && q->getAtCity()==t+1 && p->getHP()>0 ){ //我方生命值大于0，地方生命值小于0
            cout<<setw(3)<<setfill('0')<<hour<<":"<<setw(2)<<setfill('0')<<minutes;
            cout<<" "<<this->getName()<<" "<<p->getType()<<" "<<p->getNum()<<" yelled in city "<<p->getAtCity()<<endl;
            break;
        }
    }
}
```

## 2.8收集武器

与Wolf的抢夺武器相似，不过这里**因为是经过战斗的，所以编号小的武器一定在前面**，编号为2的武器，用过的一定在后面，所以是**不需要进行排序的**，只需要不断获取武器，直到对方武器全部被收集或者我方武器满了为止，如果第一个武器的编号为0或1，那么直接收集，如果编号为2（表明武器只剩下2了，且后面一定都是2，且用过的在前面，没用过的在后面），则从最后一个收集，最后一个武器一定编号为2，且武器剩余使用次数一定是最多的。

代码如下：

```
if(q->getAtCity()==p->getAtCity()){
    if(q->getHP()<=0){
        int num=p->weaponNumber();
        while(num<=10 && q->weaponNumber()>0){//只要还能装并且对面还有武器
            if(q->firstWeapon()!=2 ){//只要第一个武器不是2
                p->addWeapon(q->firstWeapon(),q->firstRemain());
                q->deleteAWeapon(q->firstWeapon());//deleteAWeapon是删除第一个这样的武器
            }else{
                p->addWeapon(q->lastWeapon(),q->lastRemain());//如果是种类是2的话
                if(q->lastRemain()==2)//判断是否用过
                    q->deleteNewArrow();
                else
                    q->deleteOldArrow();
            }
            num++;
        }
    }
    break;
}
```

## 2.9清除士兵

遍历所有的士兵，如果生命值小于等于0，从数组中清除即可（vector自动删除，不需要delete，避免堆栈溢出）

代码如下：

```
for(int i=0;i<warrior.size();i++){
        if(warrior[i]->getHP()<=0){ //如果士兵生命值为负数，清理删除
            Warrior *p=warrior[i];
            warrior.erase(warrior.begin()+i);
            i--;
        }
    }
```

## 2.10汇报情况

无论是士兵汇报战斗情况、还是士兵汇报武器情况，都是先遍历城市，然后遍历指挥部的所有士兵，先遍历红色指挥部再遍历蓝色指挥部，这样可以按照要求输出。

仅以其中的某个功能的某一个士兵为例：

```
if(warrior[i]->getAtCity()==t+1){
    Warrior* p=warrior[i];
    cout<<setw(3)<<setfill('0')<<hour<<":"<<setw(2)<<setfill('0')<<minutes;
    cout<<" "<<this->getName()<<" "<<p->getType()<<" "<<p->getNum()<<" has "<<p-
>numberOfSword()<<" sword "<<p->numberOfBomb()<<" bomb "
    <<p->numberOfArrow()<<" arrow and "<<p->getHP()<<" elements"<<endl;
    break;
}
```