计算机学院 高级语言程序设计 课程实验报告

实验题目：魔兽世界(三)开战		学号：202200400053
日期：2024-6-14	班级： 2202	姓名： 王宇涵
Email：1941497679@qq.com		

实验目的：

提高面向对象思维的编程能力.

实验软件和硬件环境：

软件环境 ：Clion

硬件环境 ：Legion Y7000P

实验步骤与内容：

题目内容不过多赘述, 题目链接如下

http://cxsjsxmooc2.openjudge.cn/warcraft/3/

AC 截图


```
//
// Created by Lenovo on 2024-05-21.
//

#include "bits/stdc++.h"
using namespace std;
class warrior{
public:
    bool isChanged;//生命/ 武器状态是否改变
```

测试输入

1

20 1 10 400

20 20 30 10 20

5 5 5 5 5

测试输出

Case 1:

000:00 blue lion 1 born

Its loyalty is 10

000:10 blue lion 1 marched to city 1 with 10 elements and force 5

000:50 20 elements in red headquarter

000:50 10 elements in blue headquarter

000:55 blue lion 1 has 0 sword 1 bomb 0 arrow and 10 elements

001:05 blue lion 1 ran away

001:50 20 elements in red headquarter

001:50 10 elements in blue headquarter

002:50 20 elements in red headquarter

002:50 10 elements in blue headquarter

003:50 20 elements in red headquarter
003:50 10 elements in blue headquarter
004:50 20 elements in red headquarter
004:50 10 elements in blue headquarter
005:50 20 elements in red headquarter
005:50 10 elements in blue headquarter

解题思路：

主体分为**七大部分**:

00 分制造武士, 05 分 lion 逃跑, 10 分武士前进, 35 分 wolf 偷武器, 40 分武士战斗, 50 分司令部报告生命元数目, 55 分士兵报告武器情况.

我们设计全局变量 warrior * city2Warrior[25][2], 用于存储每个城市的武士信息, 设计司令部类, 武士类, 分别定义对应的操作.

司令部类定义如下

```
class headQuarter{
private:
    int color;//0代表红色，1代表蓝色
    int totalLifeValue; //总生命元
    int warriorsNum[6] = {0};//生成的武士对应的个数
    int makingSeq[6] = {0};
    int curMakingIndex;//当前制造的武士的索引
    int curId;//当前制造的武士的id
    int curHour;//当前小时
    int curMinutes;//当前分钟
    int initialCity;//初始城市, 0表示红色司令部，n+1表示蓝色司令部
    bool Stop;//是否停止制造武士
    bool Taken;//是否被占领
```

```
public:
    void makeWarrior();    //制造武士
    headQuarter(int color_);    //构造函数
    ~headQuarter();    //析构函数
    bool isStop() const;
    bool isTaken() const;
    void reportLifeValue();
    void addTime(int extra);
    void printEscapeLion(warrior* theWarrior);
    void printWeapon(warrior *pWarrior);
    void printWarriorBorn();//打印武士出生信息
    void printWarriorMove(warrior* theWarrior);//打印武士前进信息
    void printTaken(warrior* theWarrior);//打印被占领信息
    void printStealWeapon(wolf *pWolf, warrior *pWarrior, int weaponId, int weaponNum);
    void printDragonYell(warrior * theDragon);//dragon欢呼
    void printKill(warrior *winner, warrior *loser);
    void printBothAlive(warrior *pWarrior, warrior *pWarrior1);
    void printBothDie(warrior *pWarrior, warrior *pWarrior1);
    void setTaken(bool b);
};
```

我们主要利用司令部类来进行信息的打印，因此维护了当前的时间.

武士类定义如下

武士类采取了多态结构，含有基类武士 warrior 和派生类五种武士 dragon, lion, iceman, ninja, wolf，其中虚析构函数表明 warrior 为虚基类.

```
class warrior{
public:
    bool isChanged;//生命/ 武器状态是否改变
    int lifeValue;//生命值
    int attack;//攻击力
    int color;//颜色
    int id;//编号
    int weaponNum;//武器数量
    int tempWeaponNum;//临时武器数量
    string name;//名字
    int city;//所在城市
    int isUsedArrow = 0;//arrow使用过的次数
    map<int, int> weapons;//编号 -> 数量
    int weaponUseOrder[12];//武器使用顺序
    int curUseWeaponIndex = 0;//当前使用的武器的索引
    warrior(int id_, int lifeValue_, int attack_, int city_, int color_, string name);//构造函数
    virtual ~warrior(){}//析构函数
    void sortWeapons();//排序武器
    void attackEnemy(warrior * enemy);//攻击敌人
    void winWeapon(warrior *enemy);//赢得武器
};
```

制造武士

和前两个实验逻辑相同，主要区别在于一旦不能制造当前武士便停止制造，而不是继续找下一个能制造的武士.

```cpp
void headQuarter::makeWarrior() {
    if (Stop) {//如果已经停止制造武士
        return;
    }
    int lifeValue = warriorLife[makingSeq[curMakingIndex]];//当前制造的武士的生命值
    if (totalLifeValue < lifeValue) {//如果生命元不够
        Stop = true;
        return;
    }
    totalLifeValue -= lifeValue;//生命元减少
    warriorsNum[makingSeq[curMakingIndex]]++;//当前武士的数量增加
    curId++;//当前武士的id
    printWarriorBorn(lifeValue);//打印武士出生信息
    if (makingSeq[curMakingIndex] == 1) {//如果是dragon
        city2Warrior[initialCity][color] = new(dragon)( id_: curId, lifeValue_: lifeValue, attack_: warriorAttack[1], remainingLife
    } else if (makingSeq[curMakingIndex] == 2) {//如果是ninja
        city2Warrior[initialCity][color] = new(ninja)( id_: curId, lifeValue_: lifeValue, attack_: warriorAttack[2], city_: initia
    } else if (makingSeq[curMakingIndex] == 3) {//如果是iceman
        city2Warrior[initialCity][color] = new(iceman)( id_: curId, lifeValue_: lifeValue, attack_: warriorAttack[3], city_: initi
    } else if (makingSeq[curMakingIndex] == 4) {//如果是lion
        lion* curLion = new(lion)( id_: curId, lifeValue_: lifeValue, remainingLifeValue: totalLifeValue, attack_: warriorAttack[4],
        city2Warrior[initialCity][color] = curLion;
        curLion->printLoyalty();
    } else if (makingSeq[curMakingIndex] == 5) {//如果是wolf
        city2Warrior[initialCity][color] = new(wolf)( id_: curId, lifeValue_: lifeValue, attack_: warriorAttack[5], city_: initialC
    }

    //更新当前制造的Index, 映射到1~5
    curMakingIndex = (curMakingIndex + 1) % 6;
    if (curMakingIndex == 0)
        curMakingIndex = 1;
```

Lion 逃跑

从左到右遍历所有城市，找到 lion 武士，如果他的忠诚值<=0，则 delete 对应的指针并置为 NULL，视为逃跑

```cpp
void lionEscape(headQuarter&redHeadQuarter, headQuarter&blueHeadQuarter) {
    for (int i = 0; i <= n + 1; i++) {
        //如果是lion, 判断忠诚度
        if (city2Warrior[i][0] != NULL && city2Warrior[i][0]->name == "lion" ){
            lion *curLion = dynamic_cast<lion *>(city2Warrior[i][0]);
            if (curLion->getLoyalty() <= 0) {
                redHeadQuarter.printEscapeLion( theWarrior: city2Warrior[i][0]);
                delete city2Warrior[i][0];
                city2Warrior[i][0] = NULL;
            }
        }
        //如果是lion, 判断忠诚度
        if (city2Warrior[i][1] != NULL && city2Warrior[i][1]->name == "lion") {
            lion *curLion = dynamic_cast<lion *>(city2Warrior[i][1]);
            if (curLion->getLoyalty() <= 0) {
                blueHeadQuarter.printEscapeLion( theWarrior: city2Warrior[i][1]);
                delete city2Warrior[i][1];
                city2Warrior[i][1] = NULL;
            }
        }
    }
    redHeadQuarter.addTime( extra: 5);
    blueHeadQuarter.addTime( extra: 5);
}
```

武士前进

这部分逻辑可能出现问题的地方在于题目要求我们从左到右输出结果，且以战士到达的目的地为准，优先输出红色，因此我们从左往右遍历城市的时候，得优先处理下一个城市即将到来的蓝武士(如果有的话)，因此我们采用 bool 数组进行判断是否已经遍历过当前武士

```cpp
//全局函数
void warriorMove(headQuarter&redHeadQuarter, headQuarter&blueHeadQuarter) {
    //初始化tmp和st
    //备份city2Warrior
    bool st[25][2] = {[0][0]: false};//标志信息有没有处理过
    //如果下一个城市的对方的武士要来到当前城市，则先处理对方武士的信息
    for (int i = 0; i <= n + 1; i++) {
        if (i <= n) {
            blueWarriorMove( city: i + 1, &: redHeadQuarter, &: blueHeadQuarter, st);
        }
        redWarriorMove( city: i, &: redHeadQuarter, &: blueHeadQuarter, st);
        blueWarriorMove( city: i, &: redHeadQuarter, &: blueHeadQuarter, st);
    }
    //此时统计各个武士应该在的位置
    for (int i = n + 1; i >= 0; i--) {
        if (city2Warrior[i][0] != NULL) {
            city2Warrior[city2Warrior[i][0]->city][0] = city2Warrior[i][0];
            city2Warrior[i][0] = NULL;
        }
    }
    for (int i = 0 ; i <= n + 1; i++) {
        if (city2Warrior[i][1] != NULL) {
            city2Warrior[city2Warrior[i][1]->city][1] = city2Warrior[i][1];
            city2Warrior[i][1] = NULL;
        }
    }

    redHeadQuarter.addTime( extra: 25);
    blueHeadQuarter.addTime( extra: 25);
}
```

其中以红武士为例，写出一次移动逻辑，此时只改变红武士的 city 值，不改变 city2Warrior 指针数组

```
//处理红武士的移动
void redWarriorMove(int city, headQuarter&redHeadQuarter, headQuarter&blueHeadQuarter,  bool st[25][2]) {
    if (city2Warrior[city][0] == NULL || st[city][0]) {
        return;
    }
    st[city][0] = true;
    city2Warrior[city][0]->city ++;
    if (city2Warrior[city][0]->name == "lion") {//如果是lion
        lion *curLion = dynamic_cast<lion *>(city2Warrior[city][0]);
        curLion->loyaltyDecrease(k);
    }else if(city2Warrior[city][0]->name == "iceman") {//如果是iceman
        iceman *curIceman = dynamic_cast<iceman *>(city2Warrior[city][0]);
        curIceman->lifeValueDecrease();
    }
    if (city2Warrior[city][0]->city == n + 1) {//如果到达蓝色司令部
        redHeadQuarter.printTaken( theWarrior: city2Warrior[city][0]);
        blueHeadQuarter.setTaken( b: true);
    }else {//如果没有到达蓝色司令部
        redHeadQuarter.printWarriorMove( theWarrior: city2Warrior[city][0]);
    }
}
```

- 注意如果是 lion 或者 iceman，其状态会进行相应改变

Wolf 偷武器

我们将偷武器的操作封装在武士 wolf 类中，并通过动态分配指针，将 warrior*指针改为子指针 wolf*指针，进行函数调用

```
void wolfStealWeapon(headQuarter &redHeadQuarter, headQuarter& blueHeadQuarter) {
    for (int i = 0; i <= n; i++) {
        if (city2Warrior[i][0] != NULL && city2Warrior[i][1] != NULL) {
            if (city2Warrior[i][0]->name == "wolf" && city2Warrior[i][1]->name == "wolf") {
                continue;
            }else if(city2Warrior[i][0]->name == "wolf" && city2Warrior[i][1]->name != "wolf") {
                wolf *curWolf = dynamic_cast<wolf *>(city2Warrior[i][0]);
                auto tt : pair<int, int>  = curWolf->stealWeapon( enemy: city2Warrior[i][1]);
                if (tt.first != -1 && tt.second != -1) {
                    redHeadQuarter.printStealWeapon( pWolf: curWolf,  pWarrior: city2Warrior[i][1],  weaponId: tt.first,  weaponNur
                }
            }else if (city2Warrior[i][0]->name != "wolf" && city2Warrior[i][1]->name == "wolf") {
                wolf *curWolf = dynamic_cast<wolf *>(city2Warrior[i][1]);
                auto tt : pair<int, int>  = curWolf->stealWeapon( enemy: city2Warrior[i][0]);
                if (tt.first != -1 && tt.second != -1) {
                    blueHeadQuarter.printStealWeapon( pWolf: curWolf,  pWarrior: city2Warrior[i][0],  weaponId: tt.first,  weaponNu
                }
            }else {
                continue;
            }
        }
    }
    redHeadQuarter.addTime( extra: 5);
    blueHeadQuarter.addTime( extra: 5);
}
```

偷武器的逻辑如下

```cpp
pair<int, int> wolf::stealWeapon(warrior *enemy) {
    for (int i = 0; i < 3; i++) {
        if (enemy->weapons[i] > 0) {
            int takeNum = enemy->weapons[i];
            if (weaponNum + enemy->weapons[i] > 10) {//不能大于10件
                takeNum = 10 - weaponNum;
            }
            //考虑拿arrow的情况
            if (i == 2) {
                int goodArrow = enemy->weapons[i] - enemy->isUsedArrow;
                if (goodArrow >= takeNum) {
                    isUsedArrow = 0;
                }else {
                    isUsedArrow = takeNum - goodArrow;
                }
            }
            weapons[i] += takeNum;
            weaponNum += takeNum;
            enemy->weapons[i] -= takeNum;
            enemy->weaponNum -= takeNum;
            //返回偷的武器编号和数量
            return make_pair( &: i,  &: takeNum);
            break;
        }
    }
}
```

武士战斗

战斗过程比较复杂，我们使用 res 来保存结果，函数 isgoingOn()来确定 res 的值判断当前战斗是否结束，其中 1 代表红色胜，2 代表蓝色胜利，0 代表平局，-1 代表继续

```cpp
int isGoingOn(warrior *redWarrior, warrior *blueWarrior) {
    if (redWarrior->lifeValue > 0 && blueWarrior->lifeValue <= 0) {
        return 1;
    } else if (redWarrior->lifeValue <= 0 && blueWarrior->lifeValue > 0) {
        return 2;
    } else if (redWarrior->lifeValue <= 0 && blueWarrior->lifeValue <= 0) {
        return 0;
    } else if (!redWarrior->isChanged && !blueWarrior->isChanged) {
        return 0;
    }
    return -1;
}
```

首先是战斗前的准备，武士要进行武器的整理

isChanged 表示武士的武器/生命值是否改变，从而判断是否平局

```
void warriorFight(headQuarter& redHeadQuarter, headQuarter& blueHeadQuarter) {
    for (int i = 0 ; i <= n + 1; i++) {
        if (city2Warrior[i][0] != NULL && city2Warrior[i][1] != NULL) {
            int res = -1; // 0表示平局, 1表示红色胜, 2表示蓝色胜
            warrior *redWarrior = city2Warrior[i][0];
            warrior *blueWarrior = city2Warrior[i][1];
            redWarrior->isChanged = true;
            blueWarrior->isChanged = true;
            redWarrior->sortWeapons();
            blueWarrior->sortWeapons();
```

开始战斗, 分为奇数偶数来判断谁先手谁后手

```
if (i % 2 == 1) {//如果是奇数, 红色先攻击
    while (res == -1) {
        redWarrior->attackEnemy( enemy: blueWarrior);
        res = isGoingOn(redWarrior, blueWarrior);
        if (res != -1) {
            break;
        }
        blueWarrior->attackEnemy( enemy: redWarrior);
        res = isGoingOn(redWarrior, blueWarrior);
    }

}
if (i % 2 == 0) { // 如果是偶数, 则蓝色先攻击
    while (res == -1) {
        redWarrior->isChanged = true;
        blueWarrior->isChanged = true;
        blueWarrior->attackEnemy( enemy: redWarrior);
        res = isGoingOn(redWarrior, blueWarrior);
        if (res != -1) {
            break;
        }
        redWarrior->attackEnemy( enemy: blueWarrior);
        res = isGoingOn(redWarrior, blueWarrior);
    }
}
```

其中进行一次攻击的逻辑如下

```cpp
void warrior::attackEnemy(warrior *enemy) {
    isChanged = false;
    //如果没有武器
    if (weaponNum == 0) {
        return;
    }
    if (curUseWeaponIndex == tempWeaponNum) {
        sortWeapons();
    }
    if (weaponUseOrder[curUseWeaponIndex] == 0) {
        curUseWeaponIndex++;
        //使用sword
        int swordAttack = attack * 2 / 10;
        if (swordAttack == 0 && weapons[1] == 0 && weapons[2] == 0) {
            isChanged = false;
        }else {
            isChanged = true;
            enemy->lifeValue -= swordAttack;
        }
    } else if (weaponUseOrder[curUseWeaponIndex] == 1) {
        curUseWeaponIndex ++;
        //使用bomb
        int bombAttack = attack * 4 / 10;
        isChanged = true;
        enemy->lifeValue -= bombAttack;
        if (name != "ninja") {
            lifeValue -= bombAttack * 5 / 10;
        }
        weaponNum--;
        weapons[1] --;
    } else if (weaponUseOrder[curUseWeaponIndex] == 2) {
        curUseWeaponIndex++;
        //使用arrow
```

```
            //使用arrow
            int arrowAttack = attack * 3 / 10;
            isChanged = true;
            enemy->lifeValue -= arrowAttack;
            if (isUsedArrow == weapons[2]) { //使用次数==武器数量
                weapons[2]--;
                weaponNum--;
                isUsedArrow--;
            } else {
                isUsedArrow++;
            }
        }
```

判断战斗结果并进行对应的操作

```
if (res == 0) {//平局
    //如果都活了
    if (redWarrior->lifeValue > 0 && blueWarrior->lifeValue > 0) {
        redHeadQuarter.printBothAlive( pWarrior: redWarrior,  pWarrior1: blueWarrior);
        if (redWarrior->name == "dragon") {
            redHeadQuarter.printDragonYell( theDragon: redWarrior);
        }
        if(blueWarrior->name == "dragon") {
            blueHeadQuarter.printDragonYell( theDragon: blueWarrior);
        }
    }else if (redWarrior->lifeValue <= 0 && blueWarrior->lifeValue <= 0) {//如果都死了
        redHeadQuarter.printBothDie( pWarrior: redWarrior,  pWarrior1: blueWarrior);
        delete city2Warrior[i][0];
        city2Warrior[i][0] = NULL;
        delete city2Warrior[i][1];
        city2Warrior[i][1] = NULL;
    }
} else if (res == 1) {//红色胜
    redHeadQuarter.printKill( winner: redWarrior,  loser: blueWarrior);
    if (redWarrior->name == "dragon") {
        redHeadQuarter.printDragonYell( theDragon: redWarrior);
    }
    redWarrior->winWeapon( enemy: blueWarrior);       //缴获武器
    delete city2Warrior[i][1];
    city2Warrior[i][1] = NULL;//蓝色武士死亡
```

```
    } else if (res == 2) {//蓝色胜
        redHeadQuarter.printKill( winner: blueWarrior,  loser: redWarrior);
        if (blueWarrior->name == "dragon") {
            blueHeadQuarter.printDragonYell( theDragon: blueWarrior);
        }
        blueWarrior->winWeapon( enemy: redWarrior);       //缴获武器
        delete city2Warrior[i][0];
        city2Warrior[i][0] = NULL;//红色武士死亡
    }
```

缴获武器的逻辑如下

```
void warrior::winWeapon(warrior *enemy) {
    if (weaponNum == 10) {
        return;
    }
    for (int i = 0; i < 3; i++) {
        if (enemy->weapons[i] > 0) {
            int takeNum = enemy->weapons[i];
            if (weaponNum + enemy->weapons[i] > 10) {//不能大于10件
                takeNum = 10 - weaponNum;
            }
            weapons[i] += takeNum;
            weaponNum += takeNum;
            if (i == 2) {
                int goodArrow = enemy->weapons[i] - enemy->isUsedArrow;
                if (goodArrow >= takeNum) {
                    isUsedArrow = 0;
                }else {
                    isUsedArrow = takeNum - goodArrow;
                }
            }
            if (weaponNum == 10) {
                break;
            }
        }
    }
}
```

报告生命元和武士报告武器的操作比较简单，不过多赘述

结论分析与体会：

本次实验相当具有挑战性，相比前两个实验对逻辑思维和严谨性提出了更高的要求. 我从拿到题目到 AC 整整用了快一周的时间才完整写完，其中遇到了很多问题，不断 debug 不断修复，也沮丧难过，也迷茫不知所措，但最终成功 AC，给我带来巨大的成就感和幸福感，让我明白认真努力会结出果实来的，同时也学到了很多调试技巧和心得经验.

最大的挑战应该来自于最后的 OJ 一直报错 MLE，给的测试样例全部正确，所以我很疑惑.
我一开始以为是进入死循环了，但是怎么找都找不到. 又觉得可能是因为指针没有 delete 掉，于是又进行了大改，发现还是没用. 又采取了删代码的策略，但是没发现规律. 后来又觉得是类写的太冗余了，占用了太大的空间，精简后发现还是不对. 我改了很多很多，最终都是没用的，一度想过放弃.
最后认真看了 warning 信息，发现一处函数在一种情况下没有返回值，抱着试试的态度，我加了返回值和报错信息，最终成功 AC，同时也学到了在系统没有接收到对应的值的时候直接会报错 MLE，而不是 RE，有种如释重负的感觉.

就实验过程中遇到的问题及解决处理方法，自拟 1－3 道问答题：

1. 如何比对大量的数据文件来 debug?

答：使用文件进行输入输出，同时通过一些工具来进行输出结果和正确答案的比对，比如网站 https://text-compare.com/.

2. OJ 系统爆 MLE 代表什么？

答：在考虑到内存大小的情况下，最大的可能是函数返回值异常!!

3. 无法直接使用 dynamic_cast 将 warrior*指针转化为具体的武士？

答 ：将 warrior 指定为虚基类，可以使用虚析构函数实现.