CPP大作业报告

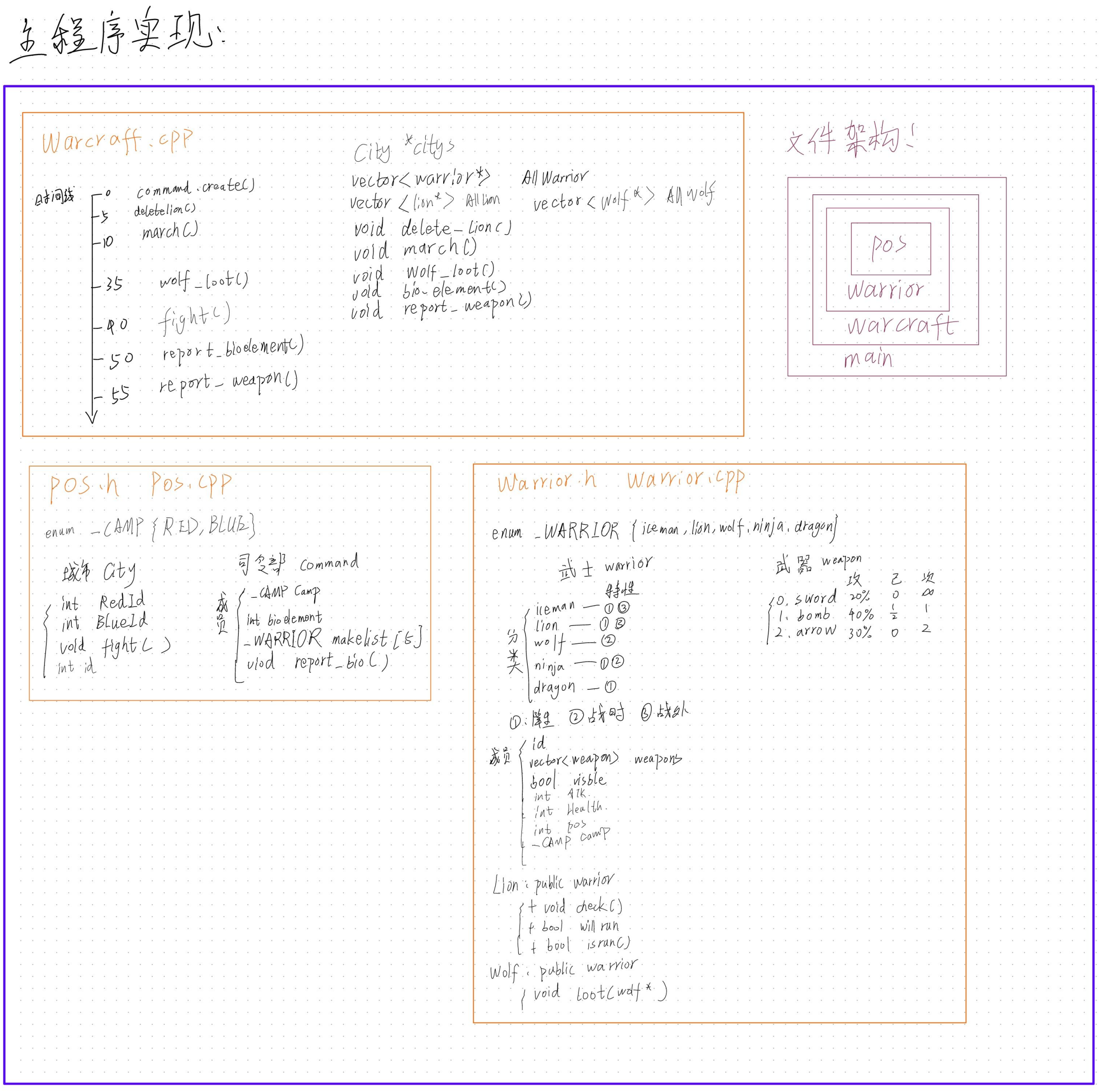
张景岳 202200130193 难度三

程序文件组织结构与前期设计

前期设计

在开始编码前我设计了整个项目的三个主要部分，Warcraft、Warrior与Pos。

如图，这是我开始编码前对项目的设计稿，包含了几个类的基本成员与整体运行逻辑。



按照时间轴在Warcraft部分里依次执行每个操作函数，来实现题目所描述的过程。

运行逻辑

因为有多组测试数据，所以在main函数里需要初始化整个游戏与运行游戏两个步骤，如下：

```cpp
int main()
{
    int t;
    std::cin >> t;
    while (t--)
    {
        init();
        game();
    }
    return 0;
}
```

之后在warcraft.cpp部分里实现这两步即可，初始化函数详见具体代码。

game()函数来模拟整个时间轴，依次执行对应的操作。

```cpp
void game()
{
    while (!isGameEnd)
    {
        if (checktime(0))
            break;
        create();
        if (checktime(5))
            break;
        delete_lion();
        if (checktime(10))
            break;
        march();
        if (isGameEnd)
            break;
        if (checktime(35))
            break;
        wolf_loot();
        if (checktime(40))
            break;
        fight();
        if (checktime(50))
            break;
        rpt_bio();
        if (checktime(55))
            break;
        rpt_weapon();
        ++CurHour;
    }
}
```

详细代码解析

下面来详细说明每个函数或类的实现。

类实现

准备工作

先定义好一些枚举类型与全局变量，方便后面函数的实现。

```
enum _WARRIOR
{
    dragon,
    ninja,
    iceman,
    lion,
    wolf
};
const char WarriorName[5][10] = {
        "dragon",
        "ninja",
        "iceman",
        "lion",
        "wolf"};

enum _WEAPON
{
    sword,
    bomb,
    arrow
};
const char WeaponName[3][10] = {
        "sword",
        "bomb",
        "arrow"};
enum _CAMP
{
    RED,
    BLUE
};
const char CampName[2][5] = {
        "red",
        "blue"};


const _WARRIOR makelist[2][5] = {
        {iceman, lion, wolf, ninja, dragon},
        {lion, dragon, ninja, iceman, wolf}};
```

Command类

司令部类有私有成员camp记录阵营，bioelement记录当前命元，curid记录当前制造到了哪个武士，isstop记录是否停止制造。

```
class Command
{
private:
    _CAMP camp;
    int bioelement;
    int curid;
    bool isStop;

public:
    Command(){};
    Command(_CAMP cp, int bio) : camp(cp), bioelement(bio), curid(0) {}
    void report_bio();
    warrior *create();
    void init(_CAMP);
};
```

City类

```
class City
{
private:
    int RedID;
    int BlueID;
    int id;

public:
    void clearRED() { RedID = -1; }
    void clearBLUE() { BlueID = -1; }
    void add(_CAMP camp, int tid)
    {
        if (camp == RED)
            RedID = tid;
        else
            BlueID = tid;
    }
    int getid() { return id; }
    int redid() { return RedID; }
    int blueid() { return BlueID; }
    City(int i) : id(i), RedID(-1), BlueID(-1) {}
};
```

weapon类

提取了三种weapon的共同点，对敌攻击与对自己攻击，使用次数，id。对使用次数为无穷的武器，只需要设定使用次数为-1即可，之后判断能否可用全是使用numofuse==0，负数都可用。

```
class weapon
{
private:
    int ATKtoOther;
    int ATKtoSelf;
    int NumOfUse;
    int id;

public:
    weapon(_WEAPON);
    bool operator<(const weapon &b) const;
    const int getID() { return id; }
    const int getNum() { return NumOfUse; }
    const int getATK2o(const warrior &a);
    const int getATK2s(const warrior &a);
    void use() { --NumOfUse; }
};
```

warrior类

最主要的类，几乎所有工作都是在这个类中实现的。有weapons表示当前对象的所有武器。

```

class warrior
{
private:
    int id;
    std::vector<weapon> weapons;
    bool visble;
    _WARRIOR type;
    int ATK;
    int Health;
    _CAMP camp;
    int pos;
    int curweaponID;
    void useweapon(warrior &b);
    void beAtk(int);

public:
    warrior(_WARRIOR ttype, int curid, _CAMP tcamp);
    int getATK() const { return ATK; }
    virtual ~warrior(){};
    void march();
    void fight(warrior &b);
    const bool &vis() const { return visble; }
    _CAMP getcamp() { return camp; }
    const int getid() const { return id; }
    const int getpos() const { return pos; }
    void report_march();
    const _WARRIOR &gettype() const { return type; }
    weapon &firstweapon() { return weapons[0]; }
    const int weaponNum() const { return weapons.size(); }
    void report_weapon();
    void addWeapon(const weapon &w) { weapons.push_back(w); }
    void sortWeapon();
    weapon belooted();
    bool emptyWeapon();
    bool sumAtk();
    void died();
};
```

```
class Lion : public warrior
{
private:
    bool WillRun;
    int loyalty;
```

```cpp

public:
    Lion(_WARRIOR ttype, int curid, _CAMP tcamp, int loy);
    bool isrun();
    void check();
};

class Wolf : public warrior
{
public:
    Wolf(_WARRIOR ttype, int curid, _CAMP tcamp) : warrior(ttype, curid, tcamp){};
    void loot(warrior *b);
};
```

函数实现

main()

```cpp
int main()
{
    int t;
    std::cin >> t;
    while (t--)
    {
        init();
        game();
    }
    return 0;
}
```

首先输入有t组数据，那么对每组数据需要初始化整个游戏系统，再进行游戏。

init()

```cpp
void init()
{
    citys.clear();
    for (auto x: AllWarrior)
        delete x;
    AllWarrior.clear();
    AllLion.clear();
    isGameEnd = false;
    std::cin >> M >> N >> K >> T;
    for (int i = 0; i < 5; ++i)
        std::cin >> InitHealth[i];
    for (int i = 0; i < 5; ++i)
        std::cin >> InitATK[i];
    CmdRed.init(RED);
    CmdBlue.init(BLUE);
    CurHour = 0;
    citys.push_back(City(0));
    for (int i = 1; i <= N; ++i)
        citys.push_back(City(i));
    citys.push_back(City(N + 1));
    printf("Case %d:\n", ++cnt);
}
```

init()函数就是初始化一些数组、读取所需的数据。

game()

```
void game()
{
    while (!isGameEnd)
    {
        if (checktime(0))
            break;
        create();
        if (checktime(5))
            break;
        delete_lion();
        if (checktime(10))
            break;
        march();
        if (isGameEnd)
            break;
        if (checktime(35))
            break;
        wolf_loot();
        if (checktime(40))
            break;
        fight();
        if (checktime(50))
            break;
        rpt_bio();
        if (checktime(55))
            break;
        rpt_weapon();
        ++CurHour;
    }
}
```

game()函数是模拟时间线，依次进行游戏的每个步骤。

期中checktime是检测是否到达游戏结束时间。

```
inline bool checktime(int minu)
{
    int sum = CurHour * 60 + minu;
    if (sum <= T)
        return false;
    isGameEnd = true;
    return true;
}
```

creat()

creat函数就是红蓝双方司令部制造武士，首先调用Command类的creat()函数，记录红蓝司令部创建的武士的内存地址，如果创建成功（nw!=nullptr）则把这个指针扔到Allwarrior数组里，这个数组记录了整局游戏里的所有武士。另一方面，如果这个武士是lion，则扔到AllLion数组里，这个数组记录所有的lion，方便之后对lion进行的一些操作。

```
void create()
{
    warrior *nw = CmdRed.create();
    if (nw != nullptr)
    {
        Allwarrior.push_back(nw);
        if (Allwarrior[Allwarrior.size() - 1]->gettype() == lion)
            AllLion.push_back(dynamic_cast<Lion *>
(Allwarrior[Allwarrior.size() - 1]));
    }
    nw = CmdBlue.create();
    if (nw != nullptr)
    {
        Allwarrior.push_back(nw);
        if (Allwarrior[Allwarrior.size() - 1]->gettype() == lion)
            AllLion.push_back(dynamic_cast<Lion *>
(Allwarrior[Allwarrior.size() - 1]));
    }
    sortwarrior();
}

```

Command类里的create函数，负责制造自己阵营的武士并输出，如果命元不足以制造武士，则制造过程停止。

```
warrior *Command::create()
{
    if (isStop)
        return nullptr;
    curid++;
    _WARRIOR wartype = makelist[camp][CurHour % 5];
    if (bioelement - InitHealth[wartype] < 0)
    {
        isStop = true;
        return nullptr;
    }
    bioelement -= InitHealth[wartype];
    warrior *pt;
    if (wartype == lion)
        pt = new Lion(wartype, curid, camp, bioelement);
    else if (wartype == wolf)
        pt = new Wolf(wartype, curid, camp);
    else
        pt = new warrior(wartype, curid, camp);
    printf("%03d:00 %s %s %d born\n", CurHour, CampName[camp],
WarriorName[pt->gettype()], curid);
    if (wartype == lion)
```

```
        printf("Its loyalty is %d\n", bioelement);
    return pt;
}
```

warrior的构造函数，负责武士的初始化与初始武器的获取。

```
warrior::warrior(_WARRIOR ttype, int curid, _CAMP tcamp) : type(ttype),
id(curid), visble(true), camp(tcamp)
{
    pos = (camp == RED ? 0 : N + 1);
    ATK = InitATK[type];
    Health = InitHealth[type];
    if (type == dragon || type == lion)
        weapons.push_back(weapon(_WEAPON(id % 3)));
    if (type == ninja)
    {
        weapons.push_back(weapon(_WEAPON(id % 3)));
        weapons.push_back(weapon(_WEAPON((id + 1) % 3)));
    }
    if (type == iceman)
        weapons.push_back(weapon(_WEAPON(id % 3)));
}
```

sortwarrior函数负责给武士排序，方便之后输出武士对应的事件。排序顺序是死亡的武士排最后面，否则按照位置排，位置相同则按阵营排。排序完成后，把所有死亡的武士都删除。

```
inline void sortwarrior()
{
    std::sort(AllWarrior.begin(), AllWarrior.end(), cmp);
    while (AllWarrior.size() > 0 && !AllWarrior[AllWarrior.size() - 1]-
>vis())
        AllWarrior.pop_back();
}
```

```
bool cmp(warrior *a, warrior *b)
{
    if (b->vis() == 0)
        return true;
    if (a->vis() == 0)
        return false;
    if (a->getpos() != b->getpos())
        return a->getpos() < b->getpos();
    if (a->getcamp() == RED)
        return true;
    return false;
}
```

delete_lion()

期中，vis()函数返回的是这个对象是否可访问，也即这个武士是否还存活，死亡的武士不考虑。

```
void delete_lion()
{
    for (auto x: AllLion)
        if (x->vis())
            if (x->isrun())
            {
                printf("%03d:05 %s lion %d ran away\n", CurHour, (x->getcamp() == RED ? "red" : "blue"), x->getid());
                x->died();
            }
}
```

willrun是lion类里的一个成员，记录这个对象是否要逃跑，在每次行进之后更新。

```
bool Lion::isrun()
{
    return willRun;
}
```

died函数完成武士死亡之后的一些操作。

```
void warrior::died()
{
    visble = false;
    if (camp == RED)
        citys[pos].clearRED();
    else
        citys[pos].clearBLUE();
}
```

march()

march函数，进行行进工作。调用每个活着的武士的march函数让他们行进，然后排序，再按顺序输出每个事件，再对lion做处理。

```
void march()
{
    for (auto x: AllWarrior)
        if (x->vis())
            x->march();
    sortwarrior();
    for (auto x: AllWarrior)
        if (x->vis())
            x->report_march();
    for (auto x: AllLion)
        if (x->vis())
            x->check();
}
```

```
void warrior::march()
{
    if (type == iceman)
        Health -= Health / 10;
    if (Health <= 0)
    {
        visble = false;
        return;
    }
    if (camp == RED)
    {
        if (citys[pos].redid() == id)
            citys[pos].clearRED();
        pos++;
        citys[pos].add(camp, id);
    } else
    {
        if (citys[pos].blueid() == id)
            citys[pos].clearBLUE();
        pos--;
        citys[pos].add(camp, id);
    }
}
```

如果有武士到达了敌方司令部，则游戏结束。

```
void warrior::report_march()
{
    if ((camp == RED && pos == N + 1) || (camp == BLUE && pos == 0))//到达敌方司令部
    {
        isGameEnd = true;
        printf("%03d:10 %s %s %d reached %s headquarter with %d elements and force %d\n",
               CurHour, CampName[camp], WarriorName[type], id, (camp == RED ? "blue" : "red"), Health, ATK);
        printf("%03d:10 %s headquarter was taken\n", CurHour, (camp == RED ? "blue" : "red"));
    } else
        printf("%03d:10 %s %s %d marched to city %d with %d elements and force %d\n",
               CurHour, CampName[camp], WarriorName[type], id, pos, Health, ATK);
}
```

```
void Lion::check()
{
    loyalty -= K;
    if (loyalty <= 0)
        willRun = true;
}
```

wolf_loot()

进行wolf的抢夺武器环节，对于每个活着的wolf对象，检索他当前所在城市的敌人，如果有则进行Loot。

```
void wolf_loot()
```

```
{
    for (auto x: AllWarrior)
        if (x->vis() && x->gettype() == wolf)
        {
            Wolf *pw = dynamic_cast<Wolf *>(x);
            warrior *po = nullptr;
            if (pw->getcamp() == BLUE)
            {
                if (citys[pw->getpos()].redid() == -1)//当前所在城市没有敌人
                    continue;
                for (auto y: AllWarrior)
                    if (y->vis() && y->getcamp() == RED && y->getid() == citys[pw->getpos()].redid())
                        po = y;
            } else
            {
                if (citys[pw->getpos()].blueid() == -1)
                    continue;
                for (auto y: AllWarrior)
                    if (y->vis() && y->getcamp() == BLUE && y->getid() == citys[pw->getpos()].blueid())
                        po = y;
            }
            if (po == nullptr)
                continue;
            pw->loot(po);
        }
}
```

首先当对方也是wolf时，或者对方无武器时，不进行抢夺，否则先让对方把武器排序，再抢夺所有跟第一把武器id相同的武器。（如果装得下）

```
void Wolf::loot(warrior *b)
{
    if (b->gettype() == wolf || b->emptyWeapon())
        return;
    b->sortWeapon();
    int Fid = b->firstweapon().getID();
    int lootnum = 0;
    while (weaponNum() < 10 && !b->emptyWeapon() && b->firstweapon().getID() == Fid)
    {
        ++lootnum;
        addWeapon(b->belooted());
    }
    sortWeapon();
    printf("%03d:35 %s wolf %d took %d %s from %s %s %d in city %d\n",
            CurHour, CampName[getcamp()], getid(), lootnum, WeaponName[Fid], CampName[b->getcamp()], WarriorName[b->gettype()], b->getid(), getpos());
}
```

sortweapon函数给武器排序，武器排序顺序跟武士相似，都是不可用的排后面然后删除掉，

```
void warrior::sortWeapon()
{
    curweaponID = 0;
    std::sort(weapons.begin(), weapons.end());
    while (weapons.size() > 0 && weapons[weapons.size() - 1].getNum() == 0)
        weapons.pop_back();
}
```

```
bool weapon::operator<(const weapon &b) const
{
    if (NumOfUse == 0)
        return false;
    if (b.NumOfUse == 0)
        return true;
    if (id != b.id)
        return id < b.id;
    return NumOfUse > b.NumOfUse;
}
```

```
weapon warrior::belooted()
{
    weapon looted = weapons[0];
    weapons.erase(weapons.begin());
    return looted;
}
```

```
void addweapon(const weapon &w) { weapons.push_back(w); }
```

fight

fight函数进行战斗过程，对每个城市，如果红蓝双方武士都在这个城市，那么双方进行对战。

```
void fight()
{
    for (auto x: citys)
    {
        if (x.redid() == -1 || x.blueid() == -1)
            continue;
        warrior *redw = nullptr, *bluew = nullptr;
        for (auto w: AllWarrior)
            if (w->vis())
            {
                if (w->getcamp() == RED && w->getid() == x.redid())
                    redw = w;
                else if (w->getcamp() == BLUE && w->getid() == x.blueid())
                    bluew = w;
            }
        if (redw == nullptr || bluew == nullptr)
            continue;
        if (x.getid() % 2 == 1)
            redw->fight(*bluew);
        else
            bluew->fight(*redw);
    }
}
```

武士类中的fight函数：进行武士间的对战工作。

先介绍辅助函数，定义了结局枚举类型，有五种战斗结束方式，con战斗未结束，继续战斗；die有一方死亡，结束；zeroWeapon，双方都没有武器了（这个实际上跟zeroAtk重复了，所以后面都是按zeroAtk来处理）；zeroATK，双方攻击力都为0；alldie，全死了。

```cpp
enum ending
{
    con,
    die,
    zeroWeapon,
    zeroATK,
    alldie
};

inline ending isend(warrior &a, warrior &b)
{
    ending flag = con;
    if (a.vis() ^ b.vis())
    {
        flag = die;
        return flag;
    }
    if (!a.vis() && !b.vis())
    {
        flag = alldie;
        return flag;
    }
    if (a.emptyWeapon() && b.emptyWeapon())
    {
        flag = zeroWeapon;
        return flag;
    }
    if (!a.sumAtk() && !b.sumAtk())
    {
        flag = zeroATK;
        return flag;
    }
    return flag;
}
```

下面介绍另一组辅助函数，使用武器和受到攻击。

使用武器就是依次使用每个武器，curweaponID记录当前使用到了那个武器，如果当前武器不合法，则检查下个武器是否可用。当使用完或检查完最后一个武器时，返回来从头开始使用每个武器。如果检查了两圈都没可用的，则所有武器都不可用，直接返回，否则敌方受到攻击，己方也收到武器对自己的伤害（有的武器为0）

```cpp
void warrior::beAtk(int num)
{
    Health -= num;
    if (Health <= 0)
```

```cpp
        visble = false;
}

void warrior::useweapon(warrior &b)
{
    if (emptyWeapon())
        return;
    while (curweaponID < weapons.size() && weapons[curweaponID].getNum() == 0)
        ++curweaponID;
    if (curweaponID == weapons.size())
    {
        curweaponID = 0;
        while (curweaponID < weapons.size() && weapons[curweaponID].getNum() == 0)
            ++curweaponID;
        if (curweaponID == weapons.size())
            return;
    }
    this->beAtk(weapons[curweaponID].getATK2s(*this));
    b.beAtk(weapons[curweaponID].getATK2o(*this));
    weapons[curweaponID].use();
    ++curweaponID;
}
```

总的fight函数，首先双方武器排序，然后双方回合制进攻使用武器，如果没结束就继续。

然后对每个结局做处理。

zeroATK时把双方武器都用完。

die时报告死亡情况，然后拿取武器。

alldie就报告双方死亡情况。

然后其他结局是双方都存活（包括zeroATK）那就报告存活情况。

```cpp
void warrior::fight(warrior &b)
{
    sortWeapon();
    b.sortWeapon();
    int time = 1;
    while (isend(*this, b) == con)
    {
        if (time % 2)
            useweapon(b);
        else
            b.useweapon(*this);
        ++time;
    }
    ending end = isend(*this, b);
    if (end == zeroATK)
    {
        for (auto &w: weapons)
            while (w.getNum() > 0)
                w.use();
        for (auto &w: b.weapons)
```

```cpp
            while (w.getNum() > 0)
                w.use();
    }
    if (end == die)
    {
        warrior &winner = (this->vis() ? *this : b);
        warrior &died = (this->vis() ? b : *this);
        printf("%03d:40 %s %s %d killed %s %s %d in city %d remaining %d elements\n",
               CurHour, CampName[winner.getcamp()], WarriorName[winner.gettype()], winner.getid(),
               CampName[died.getcamp()], WarriorName[died.gettype()], died.getid(),
               pos, winner.Health);
        if (winner.type == dragon)
            printf("%03d:40 %s dragon %d yelled in city %d\n",
                   CurHour, CampName[winner.camp], winner.id, pos);
        died.sortWeapon();
        while (winner.weapons.size() <= 10 && !died.emptyWeapon())
            winner.addWeapon(died.belooted());
    } else if (end == alldie)
    {
        warrior &redw = (this->camp == RED ? *this : b);
        warrior &bluew = (this->camp == BLUE ? *this : b);
        printf("%03d:40 both red %s %d and blue %s %d died in city %d\n",
               CurHour, WarriorName[redw.type], redw.id, WarriorName[bluew.type], bluew.id, pos);
    } else
    {
        warrior &redw = (this->camp == RED ? *this : b);
        warrior &bluew = (this->camp == BLUE ? *this : b);
        printf("%03d:40 both red %s %d and blue %s %d were alive in city %d\n",
               CurHour, WarriorName[redw.type], redw.id, WarriorName[bluew.type], bluew.id, pos);
        if (type == dragon && b.type == dragon)
        {
            if (camp == RED)
            {
                printf("%03d:40 %s dragon %d yelled in city %d\n",
                       CurHour, CampName[camp], id, pos);
                printf("%03d:40 %s dragon %d yelled in city %d\n",
                       CurHour, CampName[b.camp], b.id, pos);
            } else
            {
                printf("%03d:40 %s dragon %d yelled in city %d\n",
                       CurHour, CampName[b.camp], b.id, pos);
                printf("%03d:40 %s dragon %d yelled in city %d\n",
                       CurHour, CampName[camp], id, pos);
            }
        } else
        {
            if (type == dragon)
                printf("%03d:40 %s dragon %d yelled in city %d\n",
                       CurHour, CampName[camp], id, pos);
            if (b.type == dragon)
```

```
                printf("%03d:40 %s dragon %d yelled in city %d\n",
                       CurHour, CampName[b.camp], b.id, pos);
        }
    }
}
```

两个rpt

report命元跟武器。

```
inline void rpt_bio()
{
    CmdRed.report_bio();
    CmdBlue.report_bio();
}
```

```
void Command::report_bio()
{
    printf("%03d:50 %d elements in %s headquarter\n", CurHour, bioelement, (camp == RED ? "red" : "blue"));
}

```

```
inline void rpt_weapon()
{
    for (auto x: AllWarrior)
        if (x->vis())
            x->report_weapon();
}
```

```
void warrior::report_weapon()
{
    int sum[3] = {0};
    sortWeapon();
    for (auto x: weapons)
        if (x.getNum() != 0)
            sum[x.getID()]++;
    printf("%03d:55 %s %s %d has %d sword %d bomb %d arrow and %d elements\n",
           CurHour, CampName[camp], WarriorName[type], id, sum[sword], sum[bomb], sum[arrow], Health);
}
```

注:

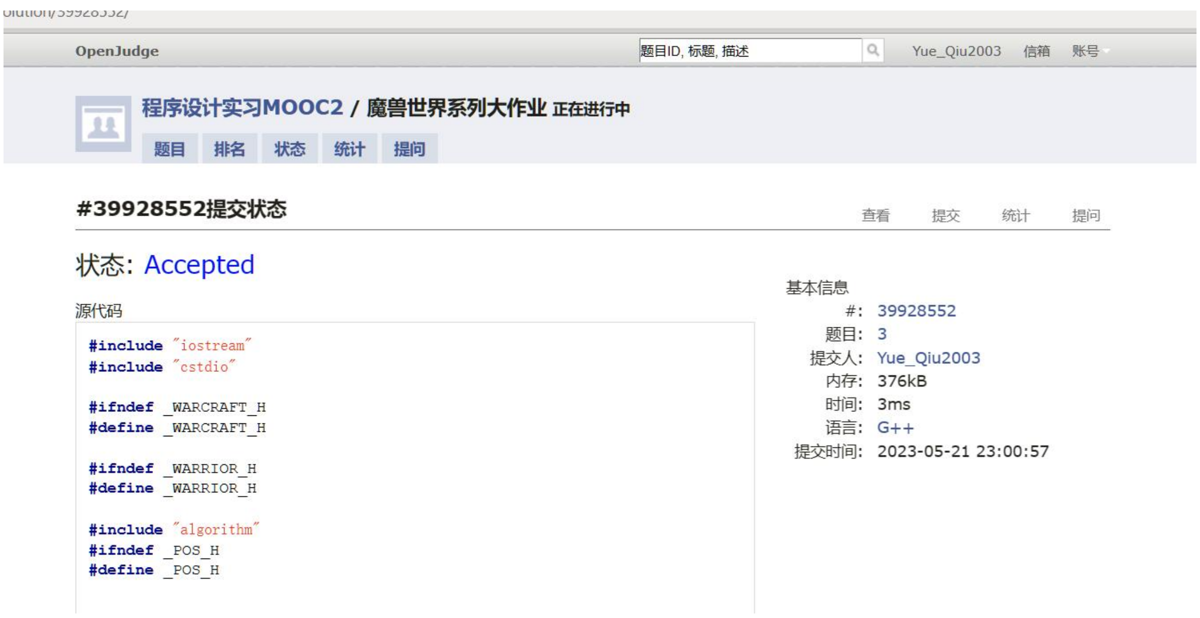
本项目在github开源：2003zjy/Warcraft: cpp大作业 (github.com)

所有修改过程可以在github的时间轴里查看。

编译项目所需的cmake文件附带在代码压缩包内，请使用支持c++17的编译器进行编译，本项目使用了c++17的部分特性。（工程环境为TMD-GCC 10.3.0）

另注：本项目代码在整理为一个文件后（手动替换头文件）已经在原题目oj（OpenJudge - 3:魔兽世界三(开战)）中通过测试（#:

39928552）



#39928552提交状态

状态: Accepted

源代码

```
#include "iostream"
#include "cstdio"

#ifndef _WARCRAFT_H
#define _WARCRAFT_H

#ifndef _WARRIOR_H
#define _WARRIOR_H

#include "algorithm"
#ifndef _POS_H
#define _POS_H
```

基本信息

#:	39928552
题目:	3
提交人:	Yue_Qiu2003
内存:	376kB
时间:	3ms
语言:	G++
提交时间:	2023-05-21 23:00:57