程序设计与算法
大作业报告

魔兽世界

姓名：赵一鸣
学号：1900012759
院系：信息科学技术学院电子系
电话：15236520530
邮箱：1900012759@pku.edu.cn

一、 需求分析

1.1 功能需求

通过面向对象的 c++程序设计实现 OpenJudge 上魔兽世界的三道题目。即按照题目描述，在设计规则下，给定基本信息作为输入，要求程序按照顺序给出完全正确的输出，包括一定时间内游戏世界里发生的所有事件。

1.2 性能需求

内存限制：题目要求内存限制在 65536kb 之内。

时间限制：前两道题要求在 1000ms 内结束，终极版要求 6000ms 内结束。

准确性要求：要求输出与标准输出完全一致，不能有任何偏差。

1.3 系统输入输出

输入：前两道题输入测试组数、初始生命元、武士生命值等，终极版的输入还包括武士攻击力，arrow 攻击力、城市个数、忠诚度变化值、测试时间等。

输出：要求程序按序输出规定时间内或从游戏开始到结束的所有事件。

二、 系统设计

2.1 模块设计

为了深入实践面向对象的程序设计思想，本程序是基于类的构造，指针、多态的灵活运用以及对象之间的相互作用来实现的。

下面主要对终极版程序中一些基本类的设计作简要介绍:

1. **City 类**：城市类。本程序中代表地点与位置的类，用来容纳武士以及产生各种事件。成员包括编号、旗帜、容纳武士指针的容器等；方法包括产生生命元、更换旗帜、调整攻击次序等。

```
class City {
public:
    City(int _id, int l = 0) :ID(_id), life_element(l), number_of_soliders(0), flag(nothing),
    last_two_wins(nothing, nothing), temp_life_elem(0) {}

    int get_ID()const { return ID; }
    int get_life()const { return life_element; }
    void generate_elem() { life_element += 10; }//产生生命元
    void set_attack_order();//设置攻击次序
    void change_wins(int c);//更改胜利情况
    void set_flags();//设置旗帜状况
    void reset();//重置城市
    vector<Solider*> get_solider() { return soliders; }

    //////对五种武士开放信息许可//////
    friend class Solider;
    friend class Dragon;
    friend class Ninja;
    friend class Iceman;
    friend class Lion;
    friend class Wolf;
    ////////////////////////////
protected:
    int ID;//城市编号
    Flag flag;//城市中的旗帜
    int life_element;//生命元
    int temp_life_elem;//生命元暂存
    int number_of_soliders;//武士总数
    vector<Solider*>soliders;//保存基类指针，存储当前城市的武士实时情况
    pair<Flag, Flag>last_two_wins;//记录前两次胜利情况
};
```

2. **Headquarters 类**：司令部类。由 City 类派生而来，代表双方阵营，用以产生武士，统计全局信息。在 City 类的基础上，成员增加了颜色阵营，储存本方所有武士指针的容器等；方法还包括产生武士，报告生命元情况等等。

```
class Headquarters :public City {
    //司令部类
public:
    Headquarters(Color col, int l, int _id) :City(_id, l), color(col), producing(true),
    number_of_soliders_ever(0), conquered(false) {
        memset(countmap, 0, sizeof(countmap));
        if (life_element < create_chain[color][0])producing = false;
    }
    ~Headquarters() {
        for_each(soliders_ever_created.begin(), soliders_ever_created.end(), deleter());//清除所有武士
    }
    //公有接口
    int get_number()const { return number_of_soliders; }
    Color get_color()const { return color; }
    bool isproducing()const { return producing; }
    bool isconquered()const { return conquered; }
    void create(Kind k);//产生特定种类的武士
    void create_next();//产生下一个武士
    void march_next();//驱动武士进军
    void check_state();//查看是否被占领
    void report_life();//报告生命元情况
    void award();//奖励武士

    //////对五种武士开放信息许可/////
    friend class Solider;
    friend class Dragon;
    friend class Ninja;
    friend class Iceman;
    friend class Lion;
    friend class Wolf;
    ////////////////////////////////

private:
    Color color;//保存阵营
    int countmap[5];//保存各种武士的数量
    bool producing;//是否正常运转
    int number_of_soliders_ever;//该司令部创造的所有武士数量
    bool conquered;//司令部是否被占领
    vector<Solider*>soliders_ever_created;//该司令部创造的所有武士
};
////////////////////////////////////////////////////////////////////////////////
```

3. **Armament 类**：武器类。实现基本的武器功能。成员包括攻击力、使用次数、持有者指针等；主要方法是 used，即武器被使用。本程序中，Armament 类派生出 sword、bomb、arrow 三种各有特点的武器类。

```
//////////////////////////////设计武器//////////////////////////////////
class Armament {
public:
    Arm get_kind()const { return kind; }
    virtual void used() = 0;//使用武器时调用
    //////对五种武士开放信息许可/////
    friend class Solider;
    friend class Dragon;
    friend class Ninja;
    friend class Iceman;
    friend class Lion;
    friend class Wolf;
    ////////////////////////////
protected:
    Armament(Solider* sol, int d = 0) :holder(sol), damage(d), available(true), usage(0) {}
    Armament(Armament& a) {}
    Arm kind;
    int damage;
    int usage;//使用次数
    bool available;//武器是否可用
    Solider* holder;//武器持有者指针
};
```

4. **Solider 类**：武士类。实现本程序主要功能的核心类，上与 City 类相交互，产生位置变化；下与 Armament 类相关联，持有并使用武器。成员包括种类、生命值、攻击力、司令部指针、位置指针、保存武器指针的容器等；方法包括攻击、死亡、进军等等。Solider 类派生出 dragon、ninja、iceman、wolf、lion 五种各有特点的武士类。

```
//////////////////////////////设计武士////////////////////////////////
class Solider {
    //武士类
public:
    //公有接口
    int get_id()const { return id; }
    int get_health()const { return health; }
    int get_damage()const { return damage; }
    Kind get_kind()const { return kind; }
    bool isfirsthand() { return first_hand; }
    bool Action() { return action; }
    Color get_color() { return color; }
    State get_state() { return state; }
    void catch_elem();//取得城市中的生命元
    void attacked(int d) { health -= d; };//受击
    virtual void march_on();//前进(利用多态处理iceman的特殊情况)
    virtual void attack(Solider* object);//攻击(利用多态处理dragon的攻击)
    virtual void fight_back(Solider* object);//反击(利用多态处理ninja的反击)
    void reset();//回合结束，状态重置
    void earn_life();//获取城市中的生命元
    void shoot();//射箭
    bool explode();//释放炸弹
    void report_weapon();//报告武器信息
    virtual void escape() {}
    virtual void yell(bool easy = false) {}
    virtual void catch_weapon(Solider* object) {}//缴获武器

    //开放信息许可
    friend class City;
    friend class Headquarters;
    friend class Dragon;
    friend class Ninja;
    friend class Iceman;
    friend class Lion;
    friend class Wolf;
    friend class Sword;
    friend class Bomb;
    friend class Arrow;
    ~Solider() {
        for_each(arms.begin(), arms.end(), deleter());//清空武器库
    }
protected:
    //受保护的构造函数，无法直接调用，防止基类对象实例化
    Solider(int _id, int _health, int _damage, Headquarters* h) :id(_id),
    health(_health), damage(_damage), camp(h), wins(0), state(none),
    first_hand(false), color(h->color), action(false) { position = camp; }

    Solider(const Solider& s) :id(s.id), health(s.health), damage(s.damage) {}
    int id;//编号
    int wins;//胜利次数
    Color color;
    Kind kind;//种类
    State state;//本回合战斗状态（战后统计）
    int health;//生命值
    int damage;//攻击力
    bool first_hand;//是否先手攻击
    bool action;//是否已经进军
    vector<Armament*>arms;//武器库
    Headquarters* camp;//表示所属司令部
    City* position;//表示所在位置
};
////////////////////////////////////////////////////////////////////////
```

2.2 内部接口

本程序采用 City（Headquarters）  Solider  Armament 三级调用的整体架构。通过城市或司令部作为总指挥控制其中的武士，武士操纵武器来完成各项活动与动作，几乎所有控制操作都是基于指针的调用完成的。

程序实现的接口包括 Headquarters 产生武士、驱动武士行军、奖励获胜武士等；City 设置武士攻击顺序、调整旗帜等；Solider 使用武器、拾取武器、丢弃武器等。

具体接口实现见源程序（魔兽 3：终极版.cpp）。

三、 系统实现

1. **设计思路**：本程序以面向对象的程序设计思想为基础，灵活使用指针，多态等语言技巧，加以整体结构的调整，利用对象的产生、消失以及对象间的相互作用来模拟、推演游戏进程。通过在正确的位置插入输出语句来产生事件输出，以最终满足题目要求。
2. **全局变量**：在本程序中使用了时间 pair，地图 vector，武器回收 vector 等实用的全局变量来进行游戏的宏观布局与调整以及内存的管理。此外还使用了大量有关颜色、旗帜、种类的枚举，这样做使程序结构更加清晰，大大增强了可读性。
3. **系统结构与主要函数**：根据题目特点，本程序在 start（）函数中使用时间线循环的方式运行，即在特定的时间点调用特定的功能函数，包括 Escape（），Attack（），March_on（）等。这种运行方式结构清晰，同时保证了在指定时间完成指定的动作，产生正确的事件输出。

这里功能函数即是特定时间段内所发生的一系列动作的集合，其通过调用各类的成员函数实现对象间的相互作用，从而产生对应的事件。将这样一系列动作封装在一些功能函数里可以使代码更加整洁规范。

4. **系统重置与内存管理**：由于题目设有多个测试样例，本程序在每一个 case 运行完后都会调用 RESET（）函数来重置全局变量，清理废弃数据等，以便给下一个 case 提供干净的运行空间。

   值得一提的是，在内存管理层面，本程序中每一个在堆中申请的对象都会在特定容器中（weapon_bin , soliders_ever_created 等）保留其指针，以便在程序结束时通过析构函数或者显式 delete 来清理内存占用。

四、 系统测试

本程序在 OpenJudge 上提交后，成功通过三个阶段的题目测试。其中在第三阶段（魔兽世界终极版）中通过了所有测试样例，消耗内存 2432kb，用时 286ms，整体完成了题目所有要求。

	题目ID	标题	通过率	通过人数	尝试人数
✔	A	魔兽世界之一：备战	97%	29	30
✔	B	魔兽世界之二：装备	100%	29	29
✔	C	魔兽世界终极版	90%	18	20

基本信息

#:	24145177
题目:	C
提交人:	1900012759赵一鸣
内存:	2432kB
时间:	286ms
语言:	G++
提交时间:	2020-05-15 11:51:50

五、 难点总结

由于整体架构复杂、可操作的对象与变量众多、实际情况复杂，本题目的难点在于对象的储存、管理与对象之间的相互作用的合理调整。

为了解决上述问题，本程序通过构造结构完整的基类与派生类，借用指针这一强大工具，结合多态的概念，大大降低了代码复杂度，提升了代码可读性。

```
virtual void escape() {}
virtual void yell(bool easy = false) {}
virtual void catch_weapon(Solider* object) {}//缴获武器
virtual void march_on();//前进(利用多态处理iceman的特殊情况)
virtual void attack(Solider* object);//攻击(利用多态处理dragon的攻击)
virtual void fight_back(Solider* object);//反击(利用多态处理ninja的反击)
```

如上图所示，利用各种虚函数，来完成不同种类武士具有不同特点的动作。例如除了基类 Solider 的 attack () 函数，本程序对 lion，dragon，wolf 的 attack () 函数进行了重写，以满足这三者在攻击时的特殊表现形式。

此外，各个对象保留的信息与全局信息联动、动态更新、互相交换，这既体现了模拟进程的设计思想，又便于后期调试。

通过设计程序完成魔兽世界题目，我对面向对象的编程思想有了更深的理解，也加强了指针运用层面的技巧，整体上增强了使用 c++语言编程的能力。

注：查阅源代码可参考 github 库 https://github.com/ZYM-PKU/OJ-World_of_Warcraft。