**C++面向对象课程设计说明文档**

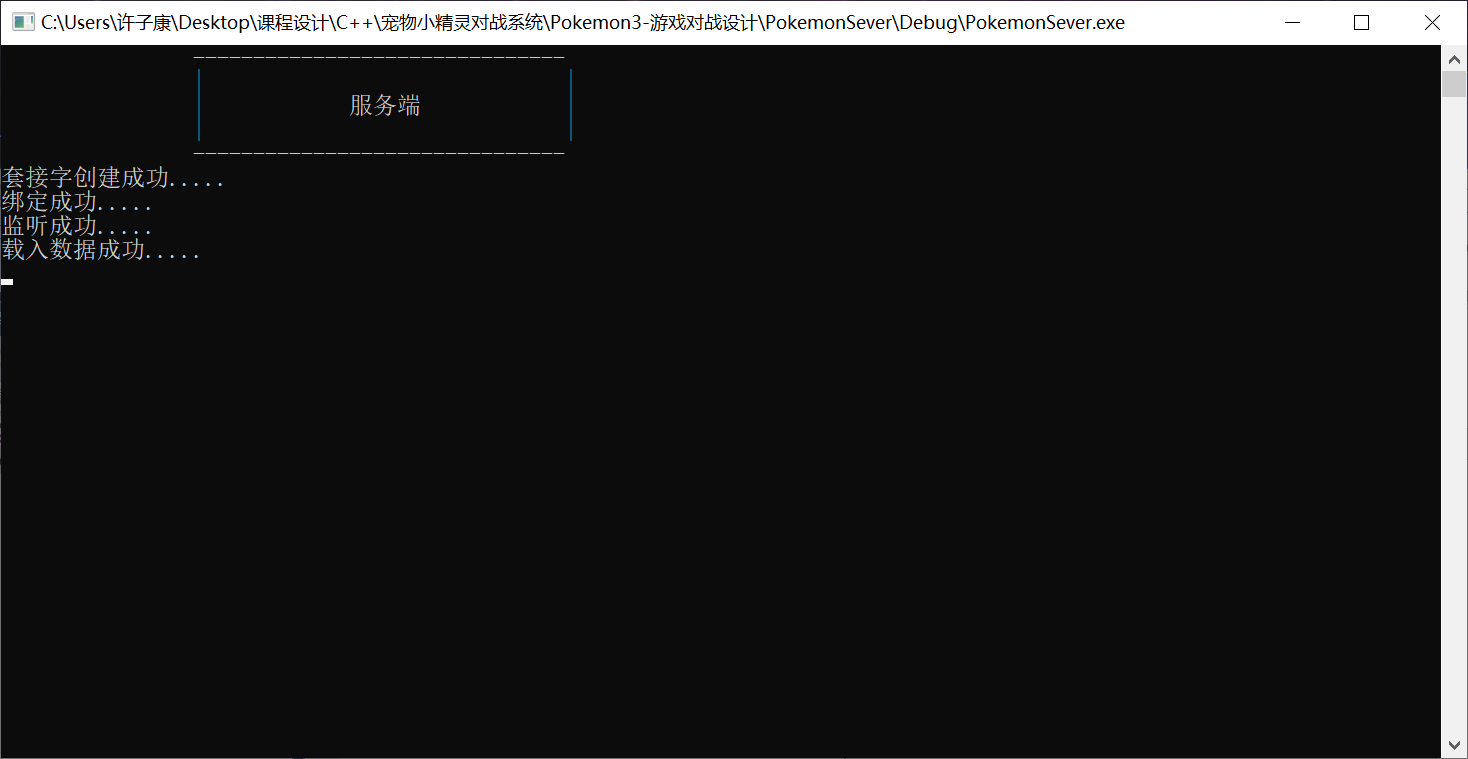
**宠物小精灵对战系统**

**一、 课程设计内容**

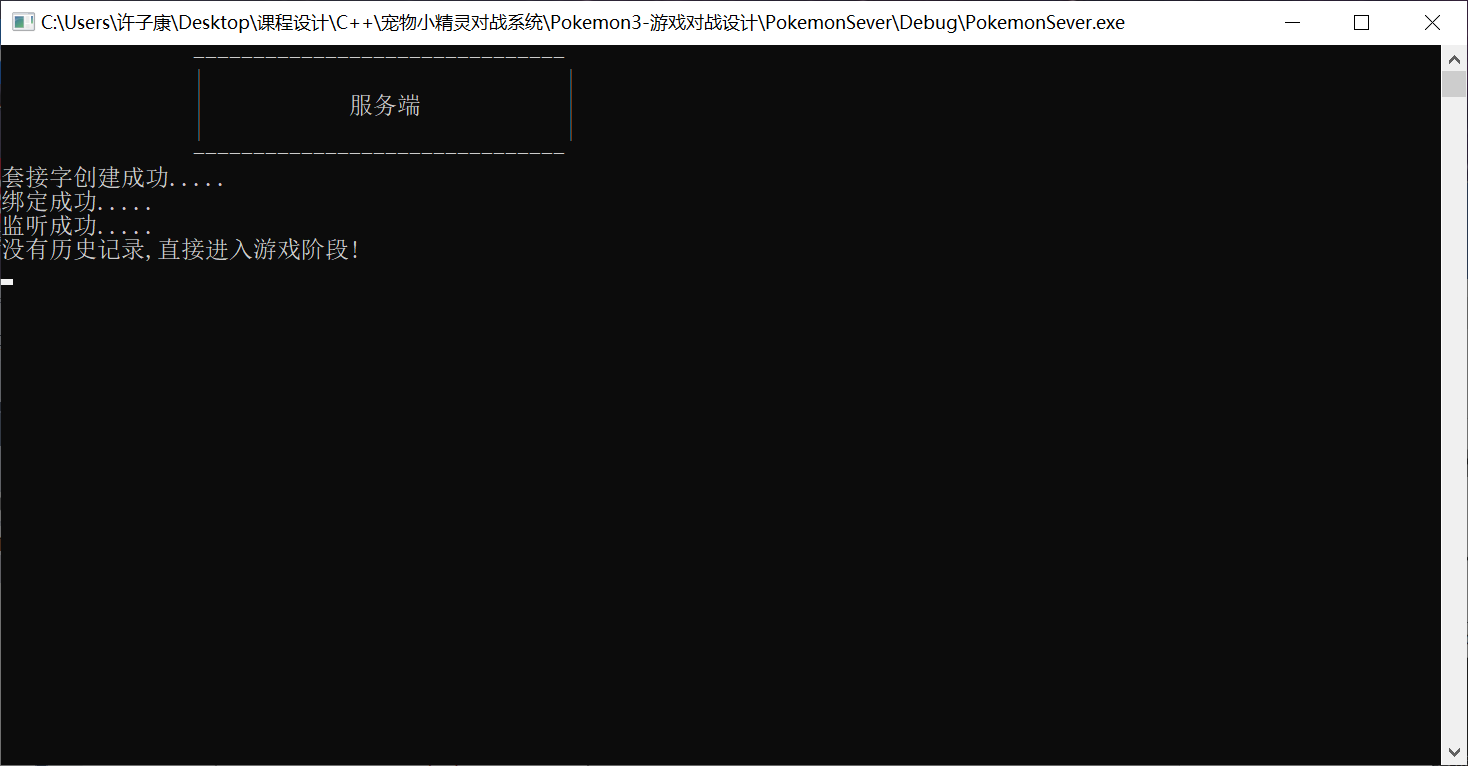
用面向对象的设计方法来设计一款平台类对战游戏。整个题目设计分为三部分，从宠物小精灵的加入到用户注册与平台登录，最后实现游戏对战的设计。

**二、.宠物小精灵对战系统使用说明**

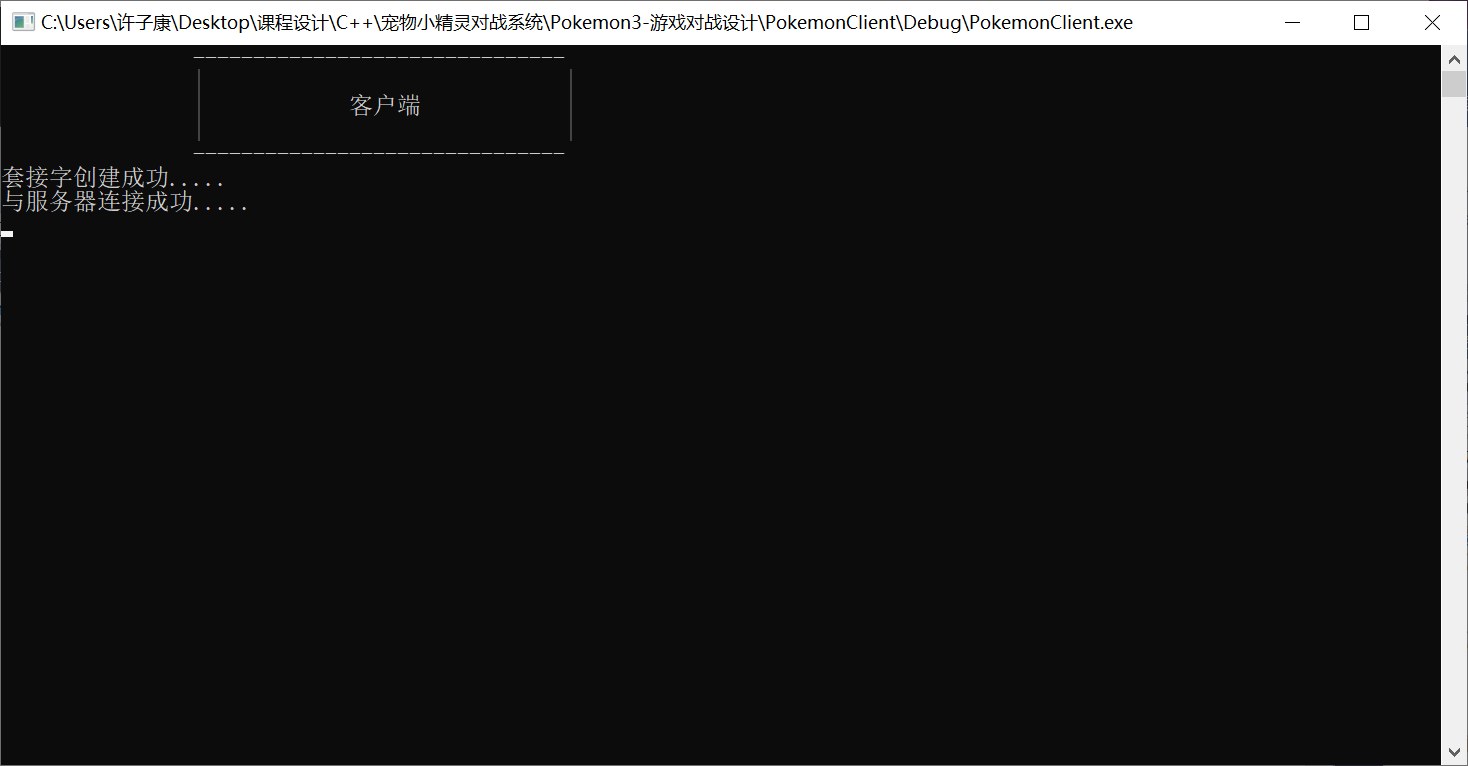
1. 首先运行宠物小精灵对战系统服务器，如图：



2. 如果本地有存储用户信息和宝可梦信息的资源文件，会显示“载入数据成功”，否则显示“没有历史记录,直接进入游戏阶段!”，如图：



3. 出现以上两种情况，表示服务器正常运行，此时可以打开客户端，如图：

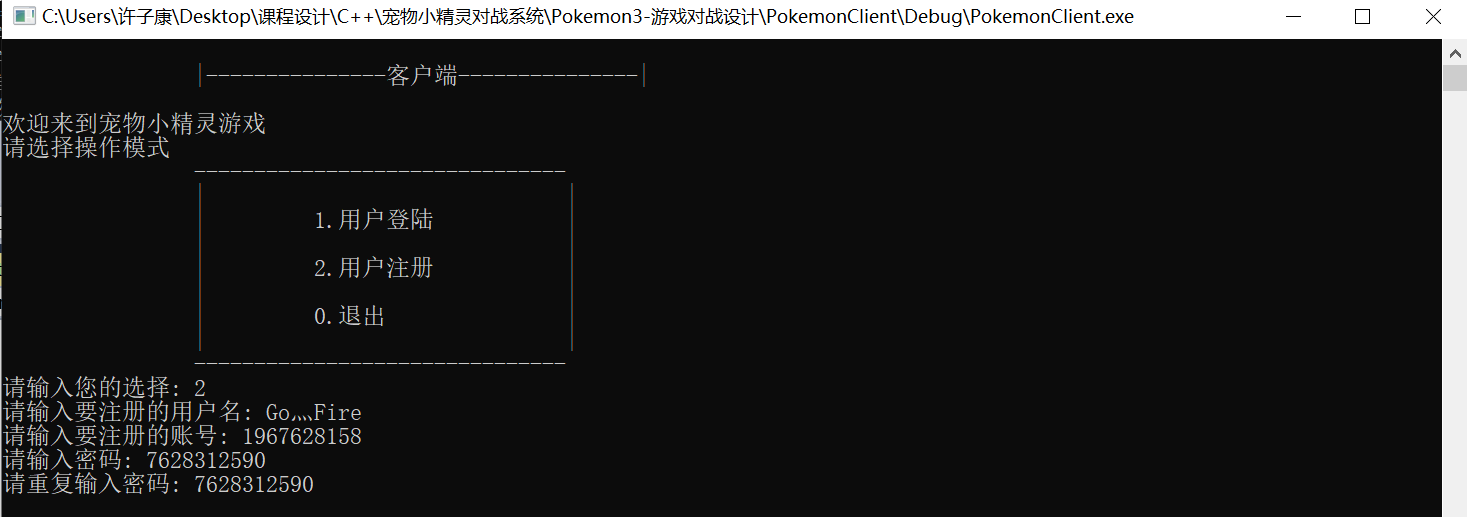


4. 在客户端设定自动跳转，如果与服务器连接成功，在等待0.5s后，自动进入下一个页面，如图：



5. 此时展现出三种选择，用户登录、用户注册和退出。

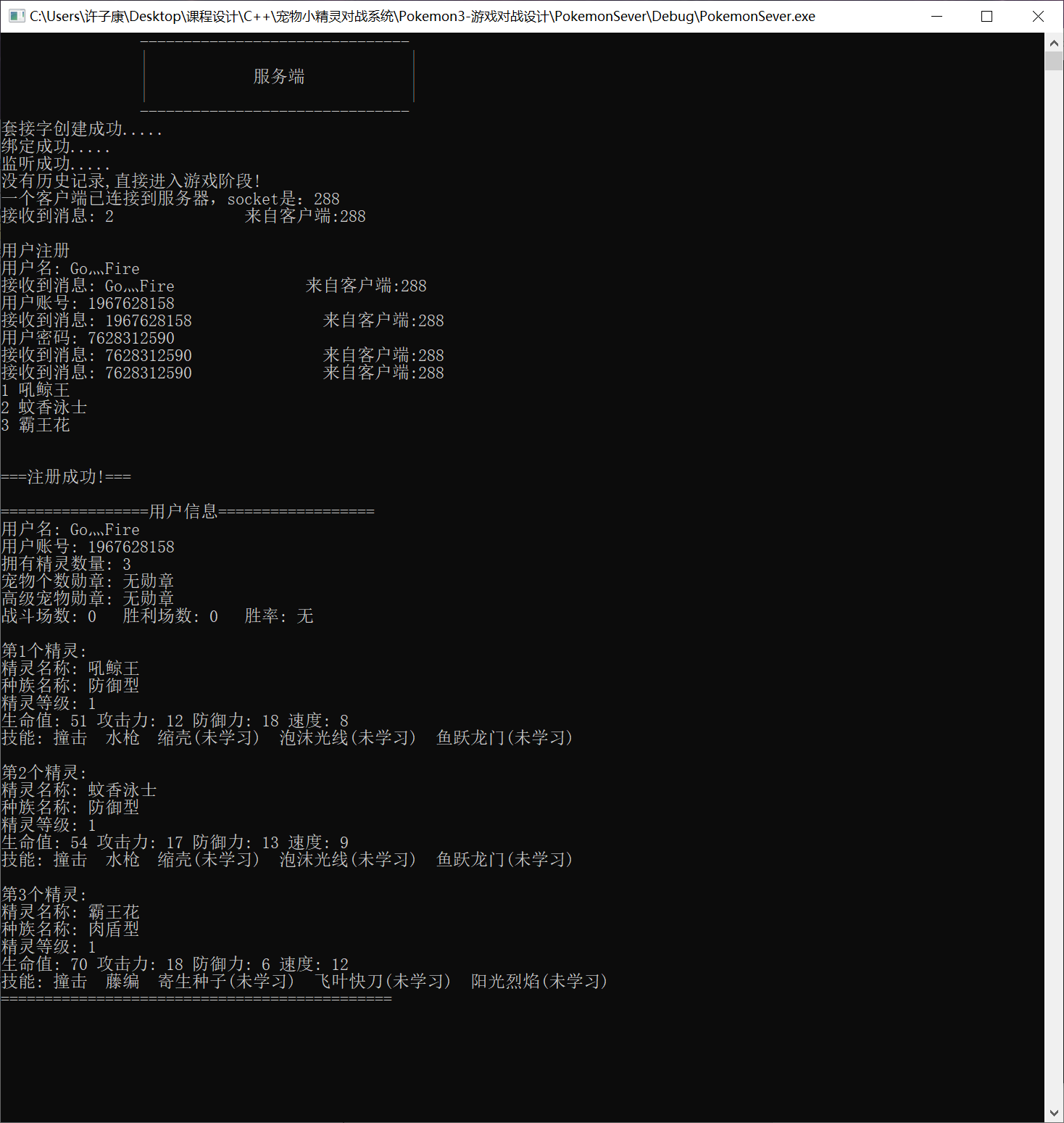
6. 首先进行用户注册，输入2，按下回车，进入用户注册界面，需要按要求输入要注册的账号用户名、账号id、密码并要求重复密码，如图：



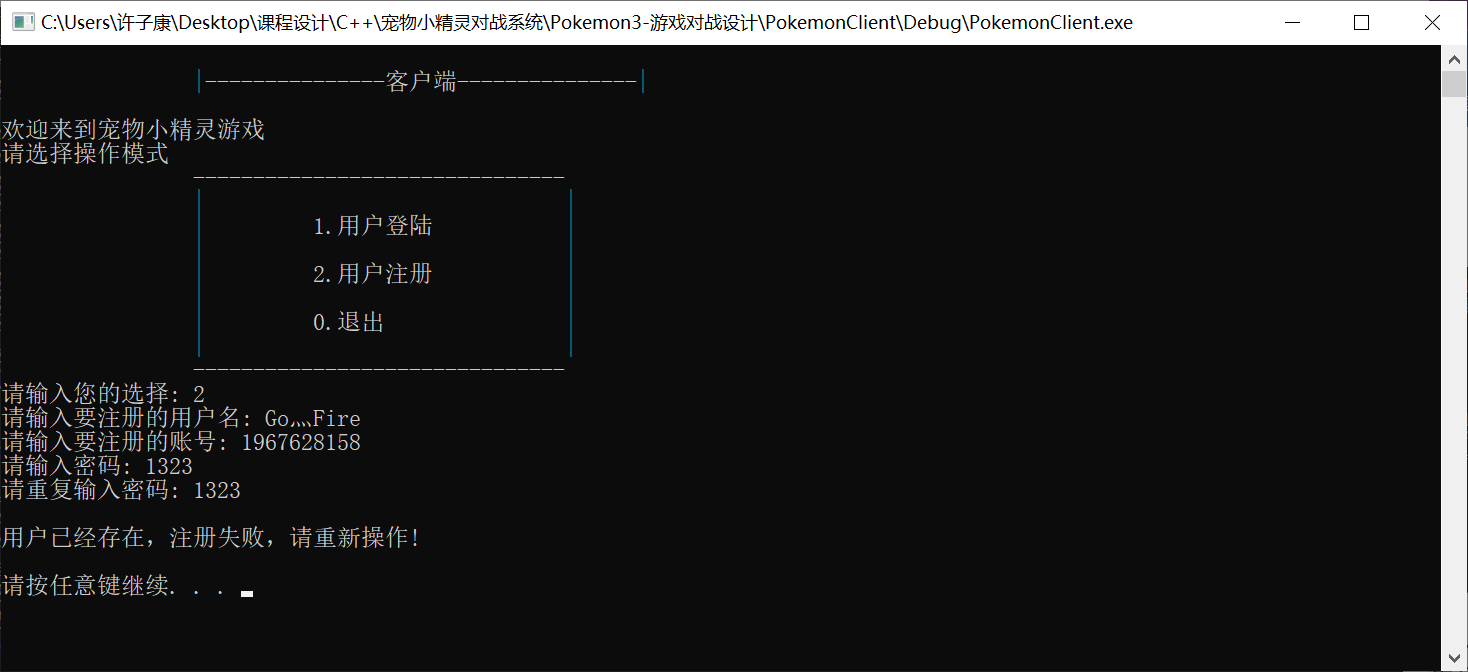
7. 如果注册成功，会显示注册成功的提示信息，并给新注册的用户随机分配三个一级精灵，如图：



8. 此时服务器也会显示相应的信息，如图：

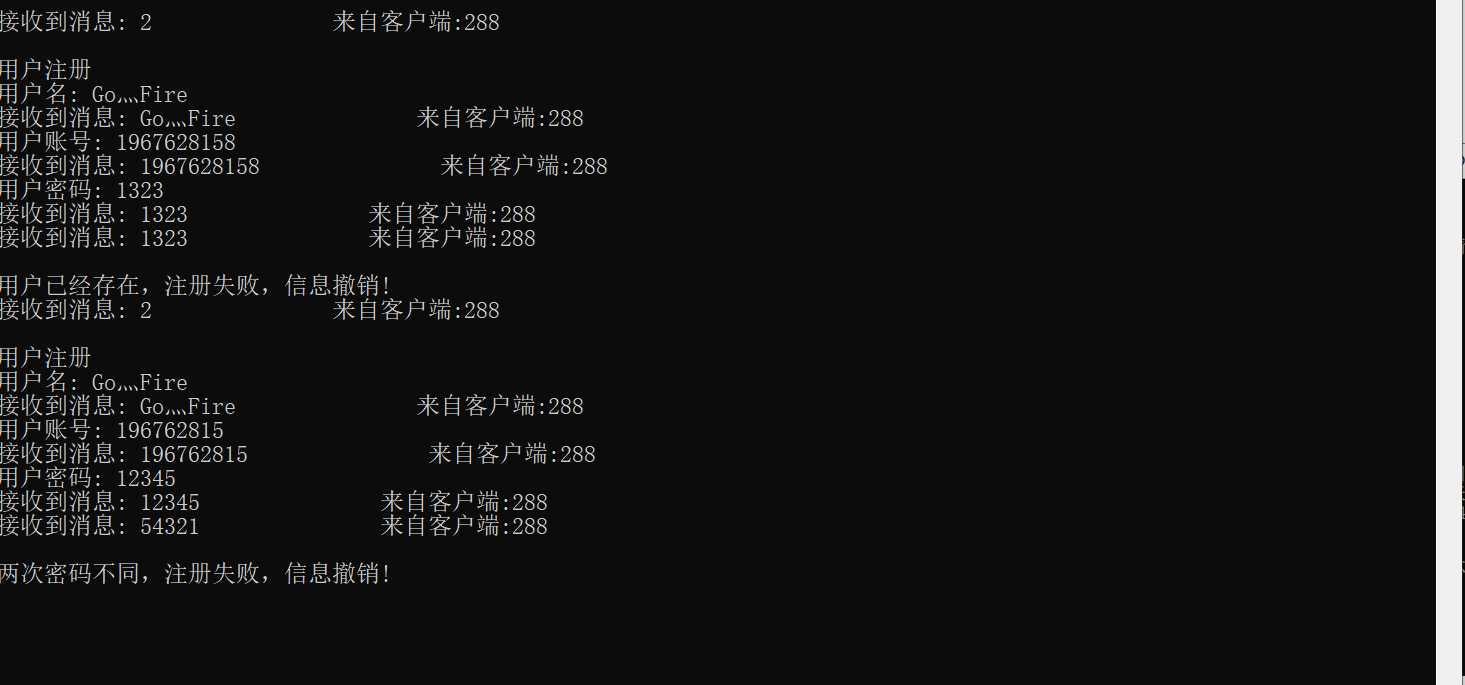


9. 如果在用户注册的过程中，出现已存在的用户账号id或者两次输入的密码不相同，会提示注册失败的信息，如图：

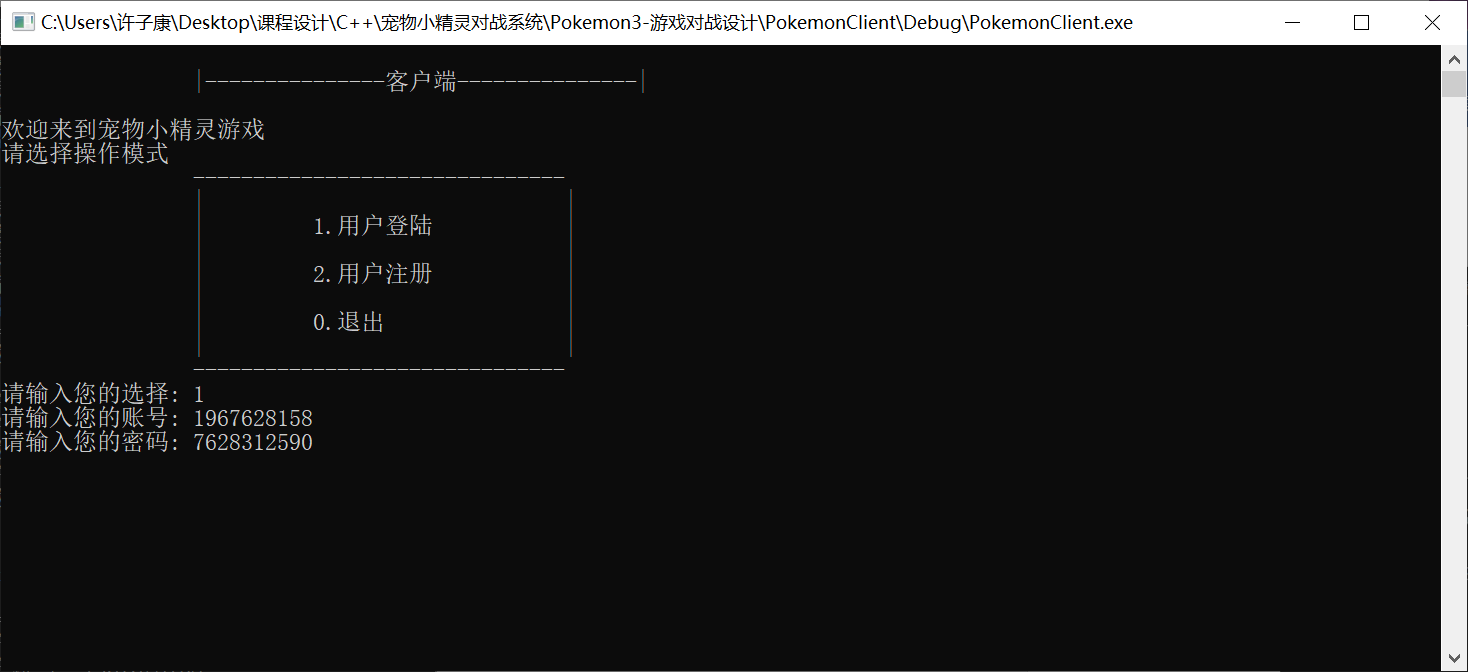




10. 同时服务器也会显示相对应的信息，如图：



11. 首先执行两次注册操作，注册用户“Go灬Fire”和“xzk”，然后进入用户登录界面，按要求输入用户id和密码，如图：

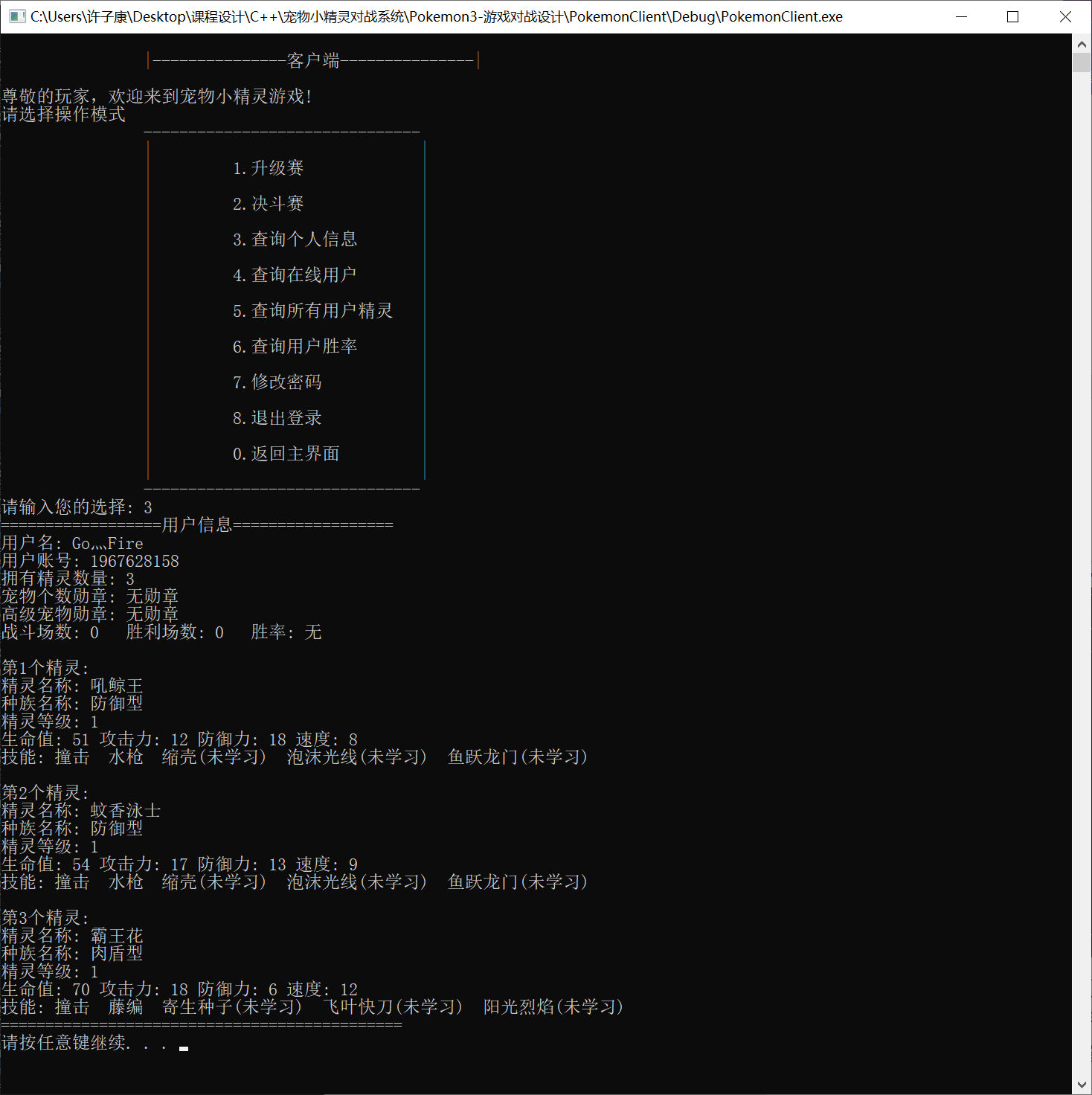


12. 按下Enter键，进入用户操作界面，如图：



13. 用户可以执行下面9种操作，进行升级赛、决斗赛、查询个人信息、查询在线用户、查询所有用户精灵、查询用户胜率、修改密码、退出登录和返回主界面。

14. 进行查询个人信息操作，如图：



15. 进行查询在线用户操作，如图：



16. 进行查询所有用户精灵操作，如图：



17. 进行查询所有用户胜率操作，如图：



18. 进行修改密码操作，如图：



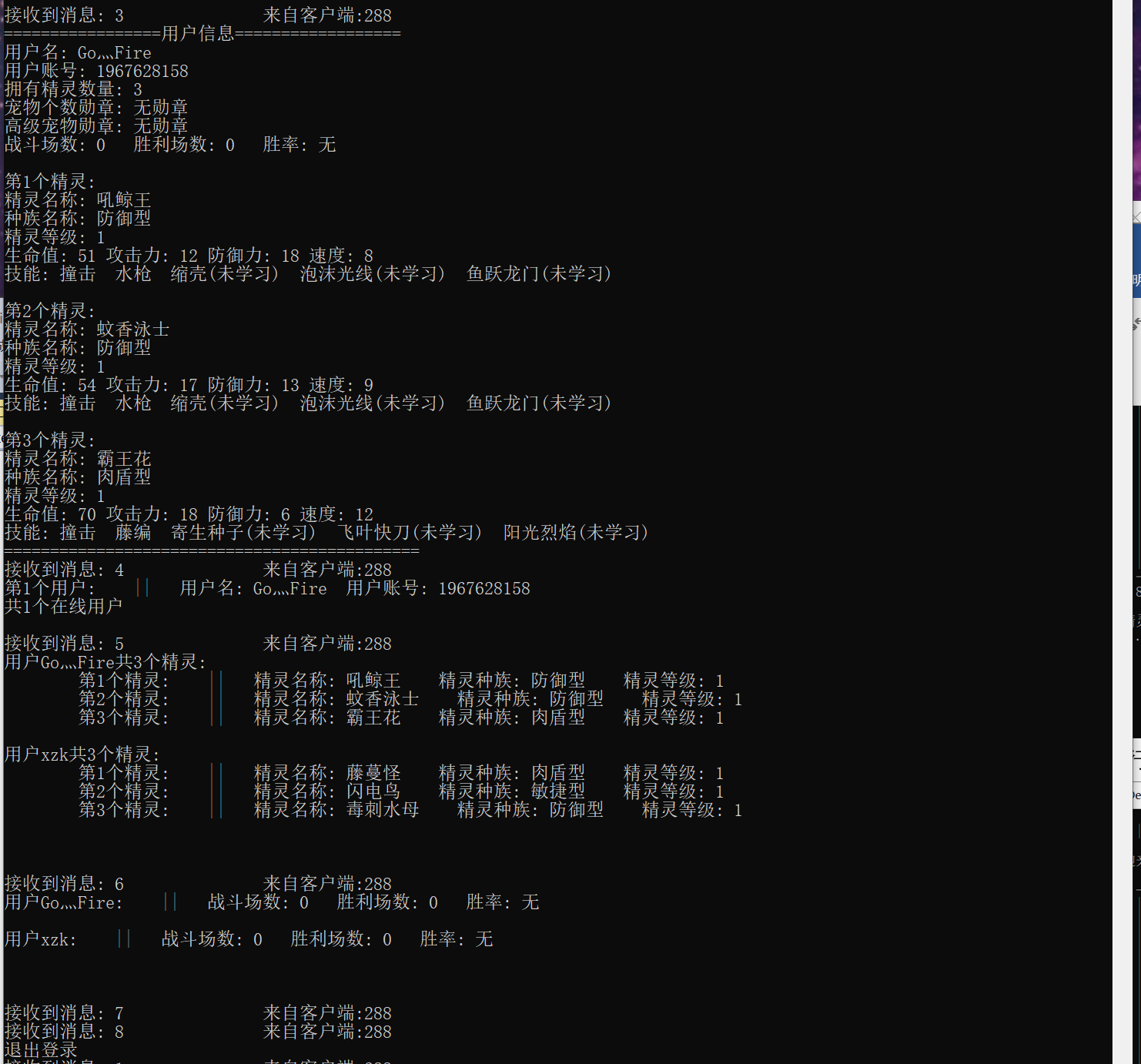
19. 进行退出登录操作，如图：



20. 进行返回主界面操作，如图：

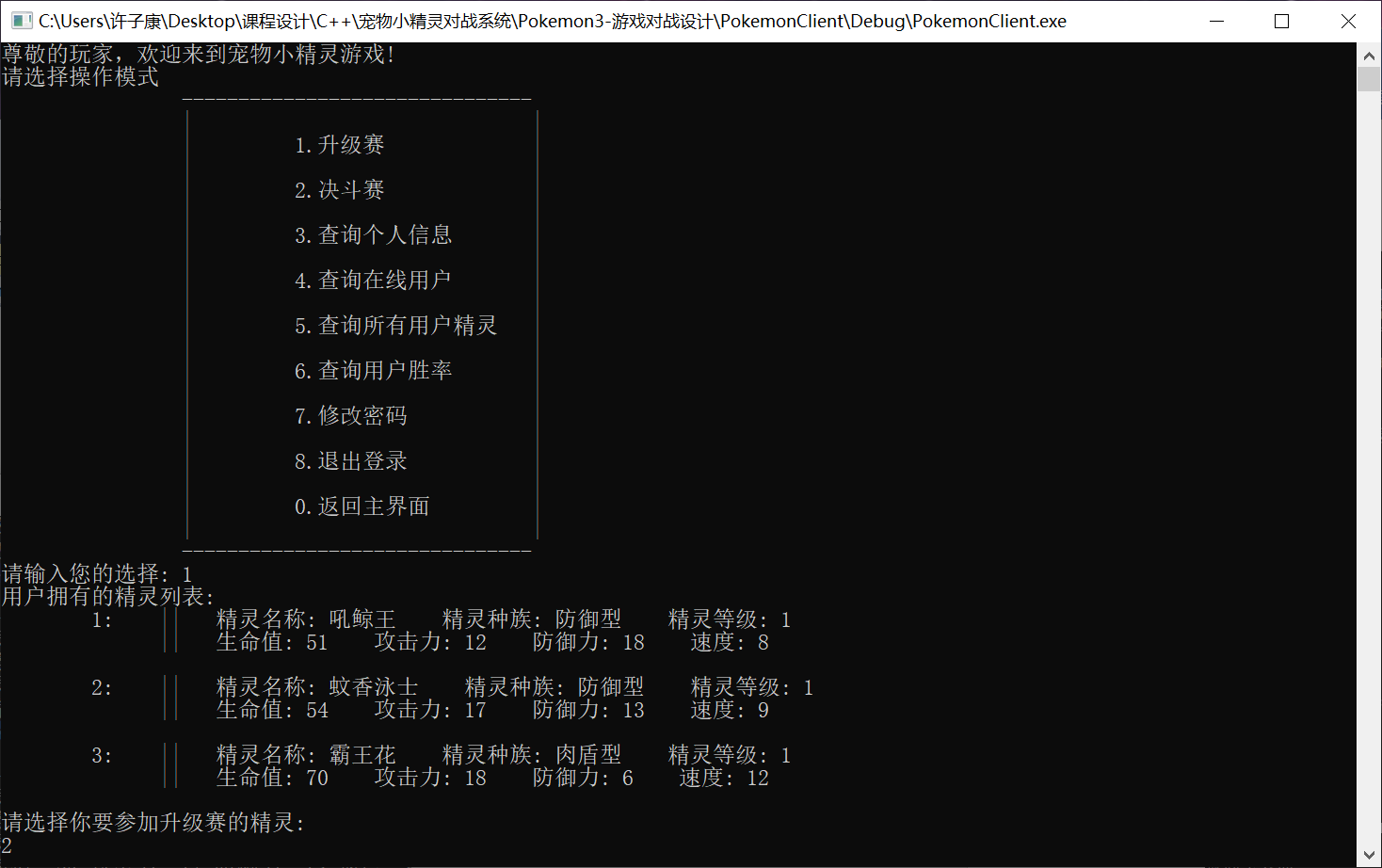


21. 对应的服务器显示信息如图：

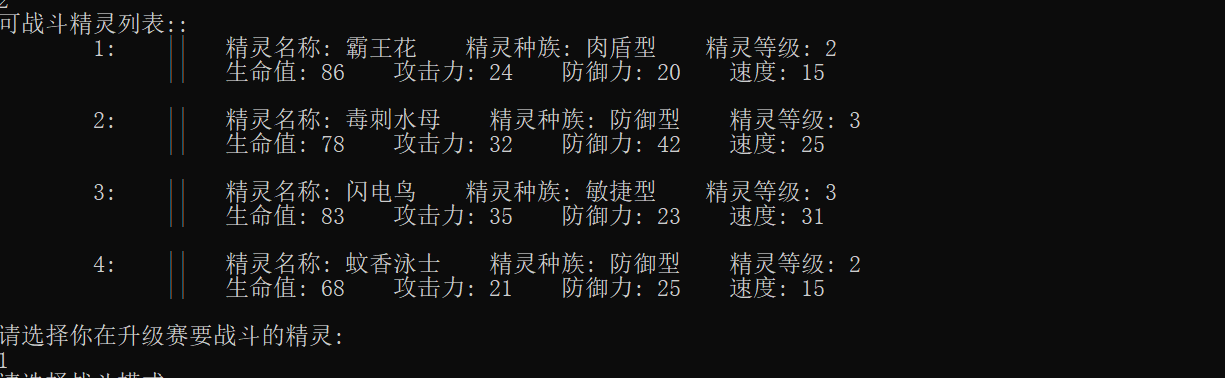




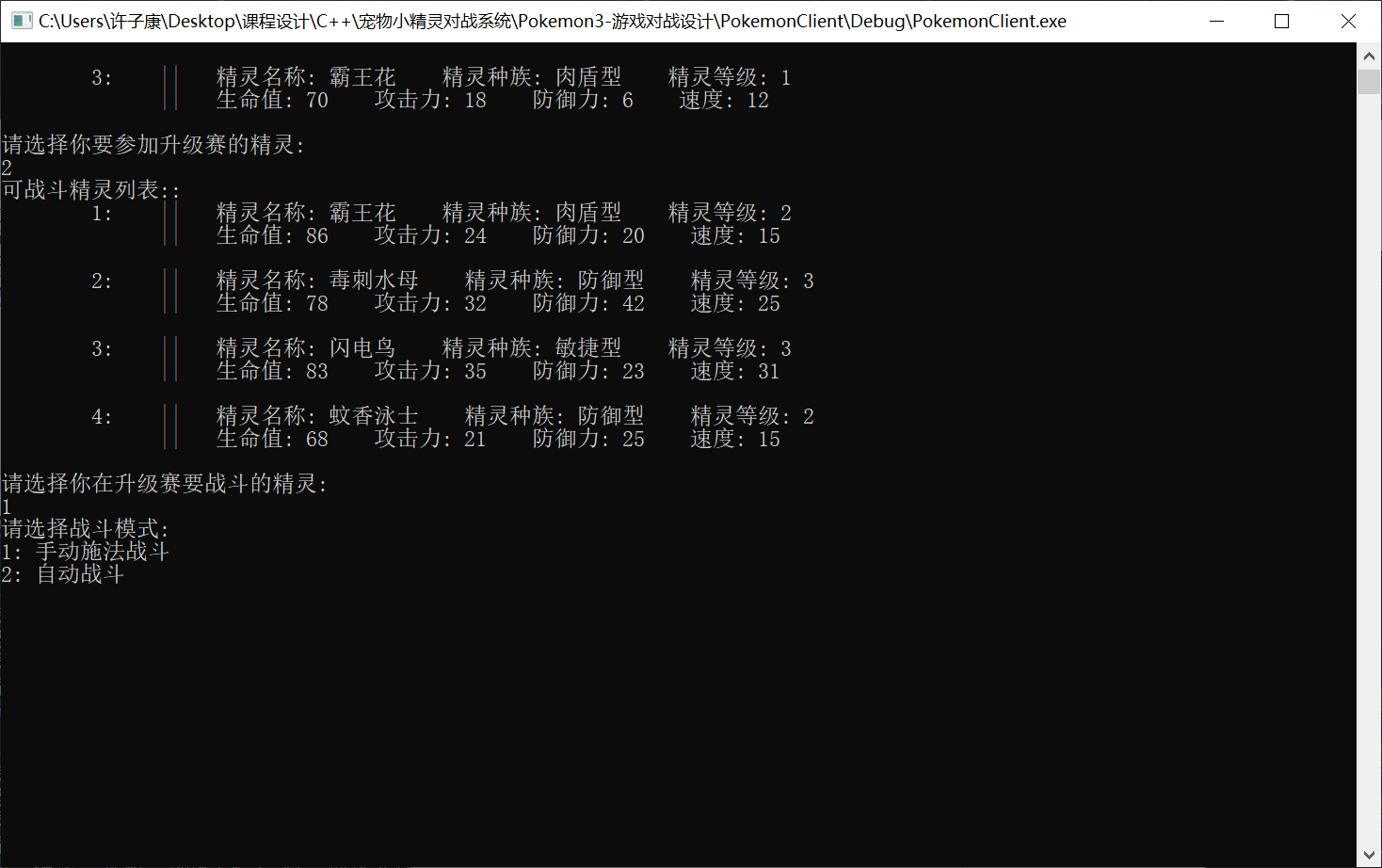
22. 进行升级赛，首先选择你要参加战斗的精灵，如图：



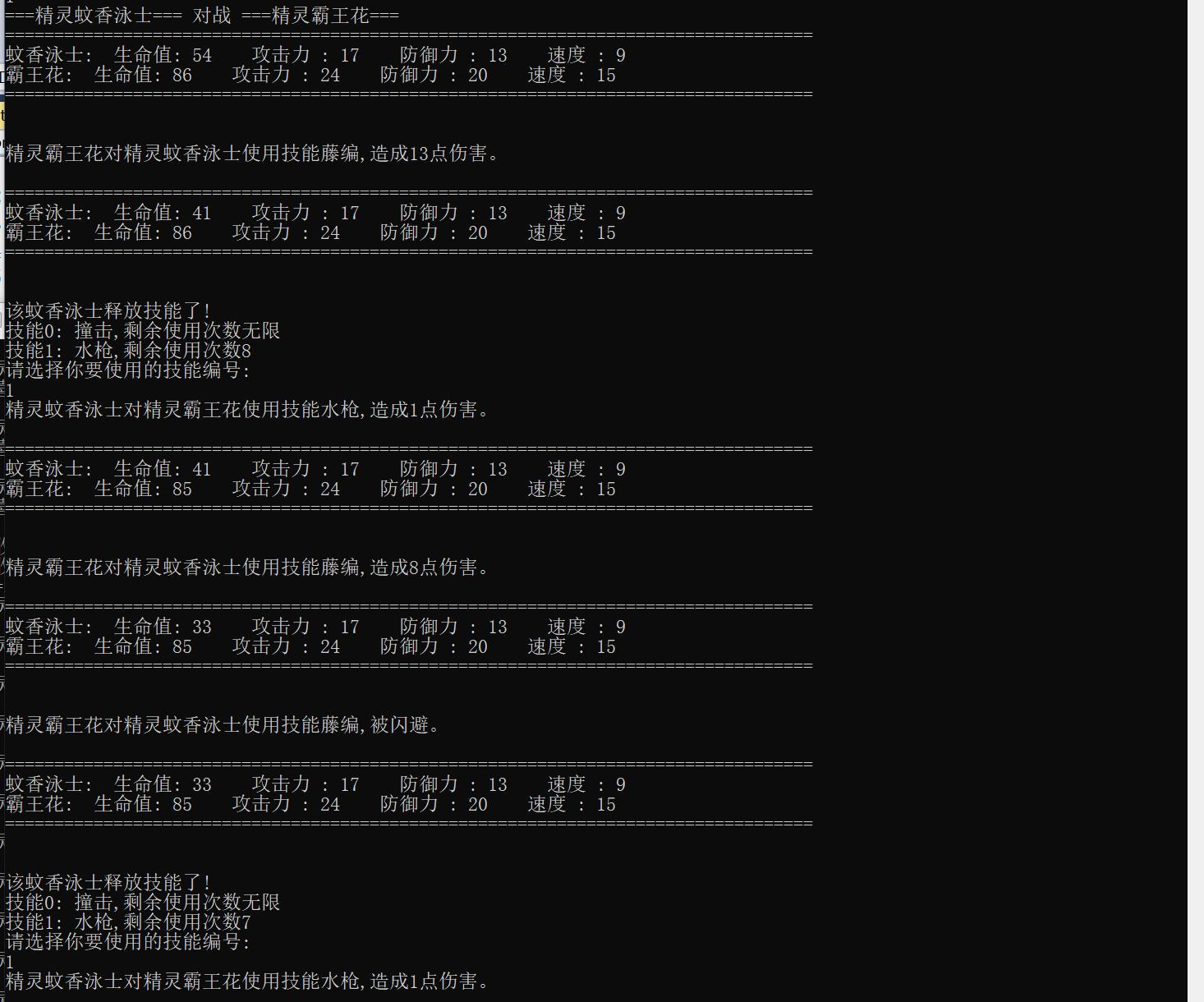
23. 之后选择你要对战的精灵，如图：

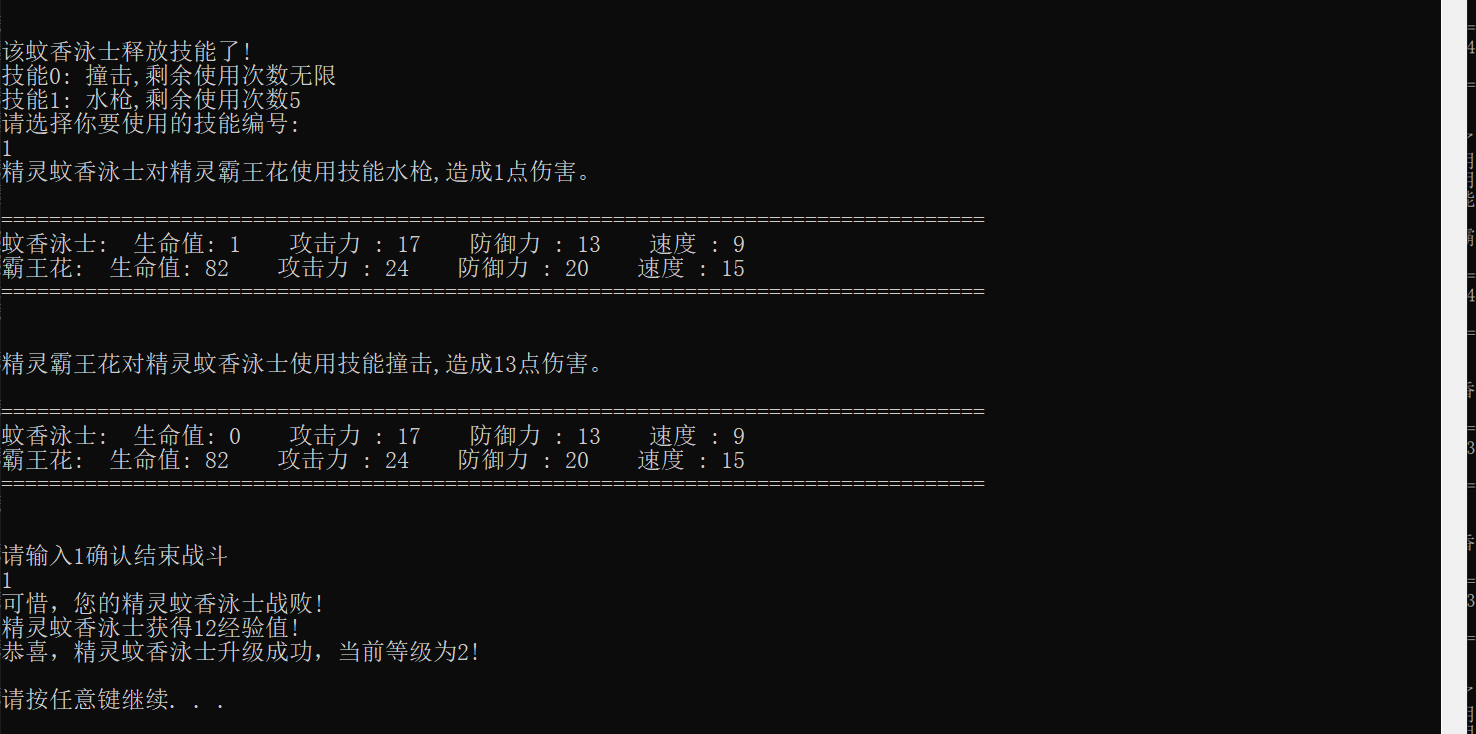


24. 之后可以选择战斗模式，手动施法战斗和自动战斗，如图：



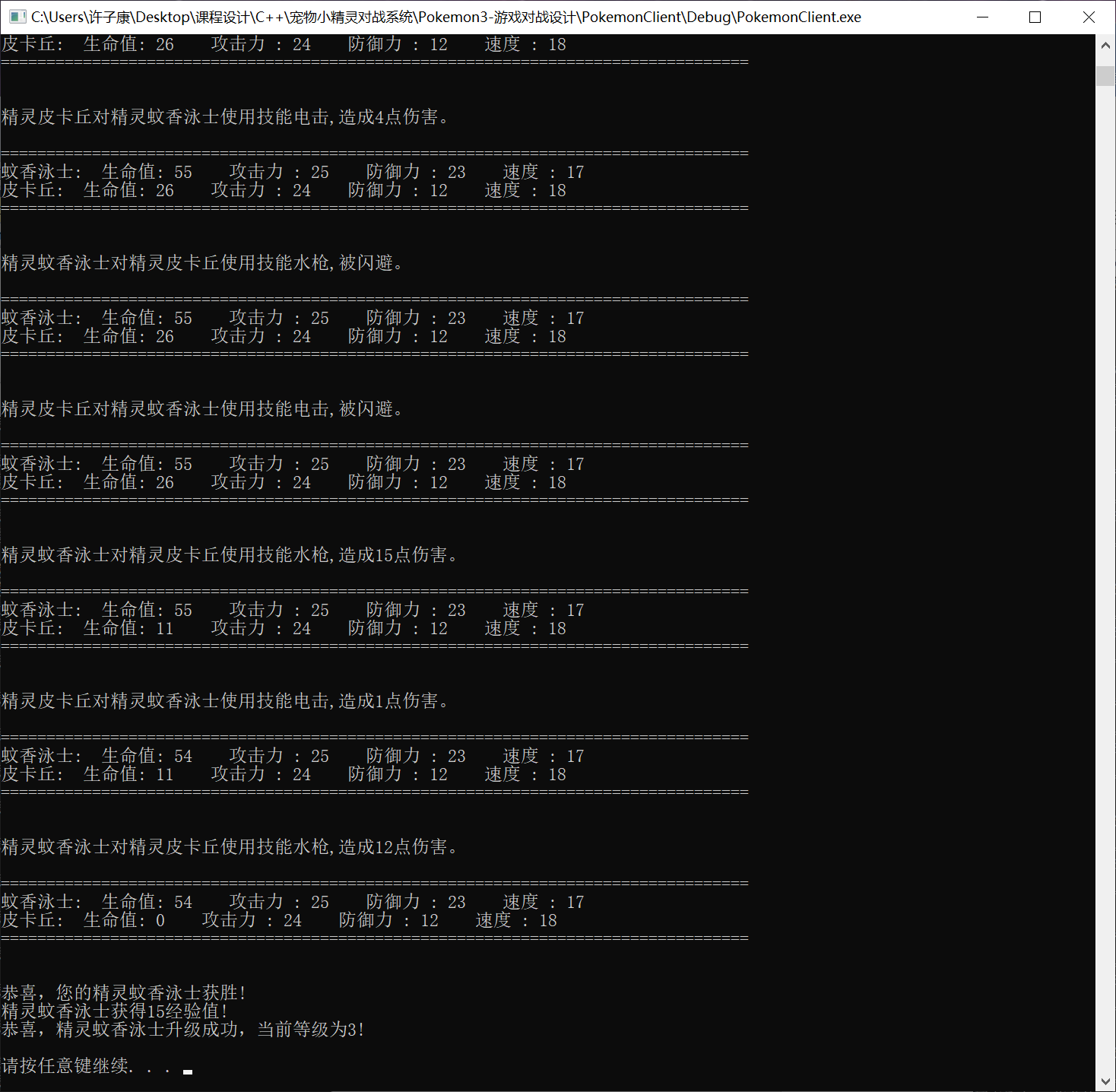
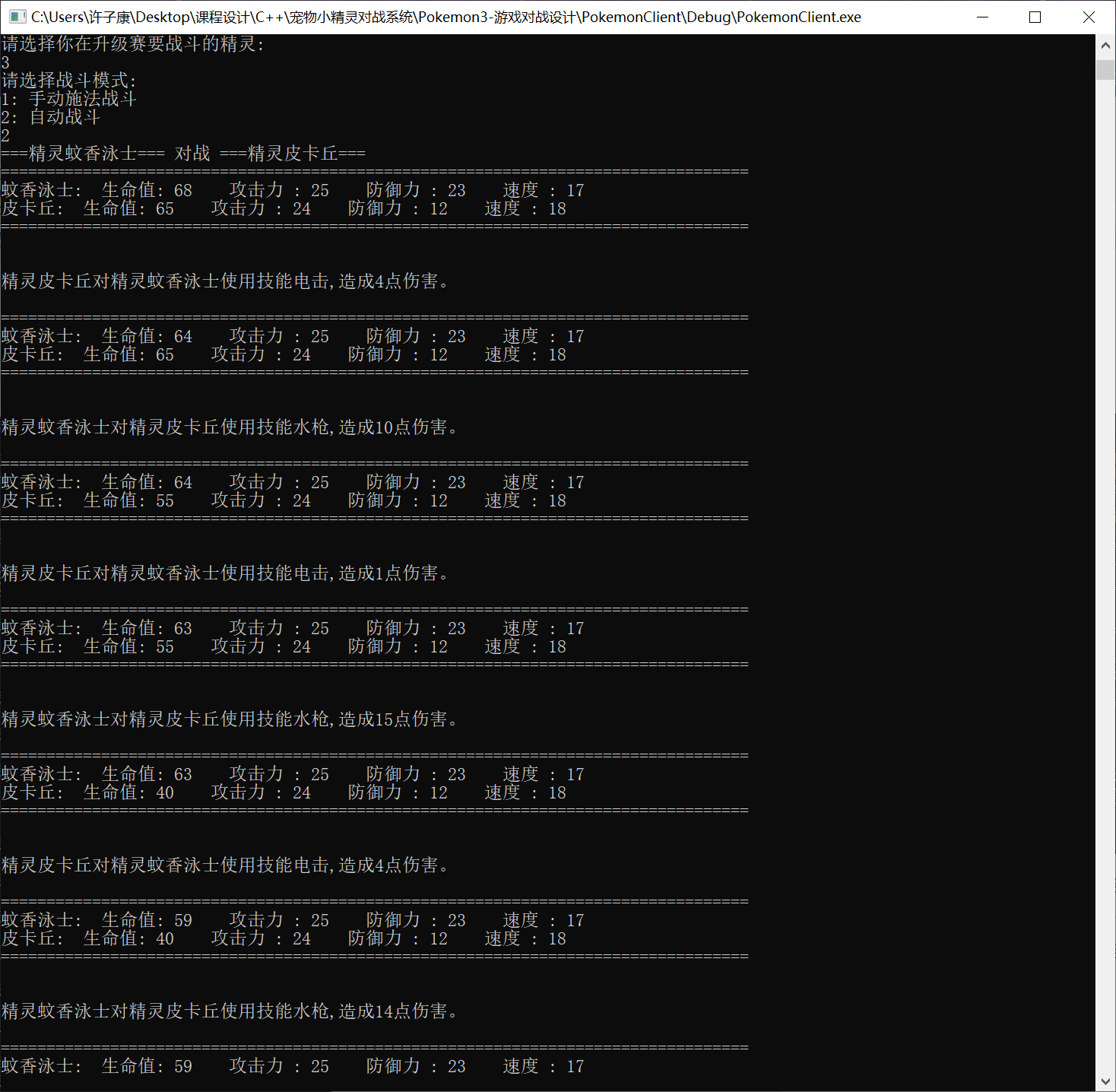
25. 手动施法战斗，每次需要选择使用哪个技能，直到一方精灵失败，如图：



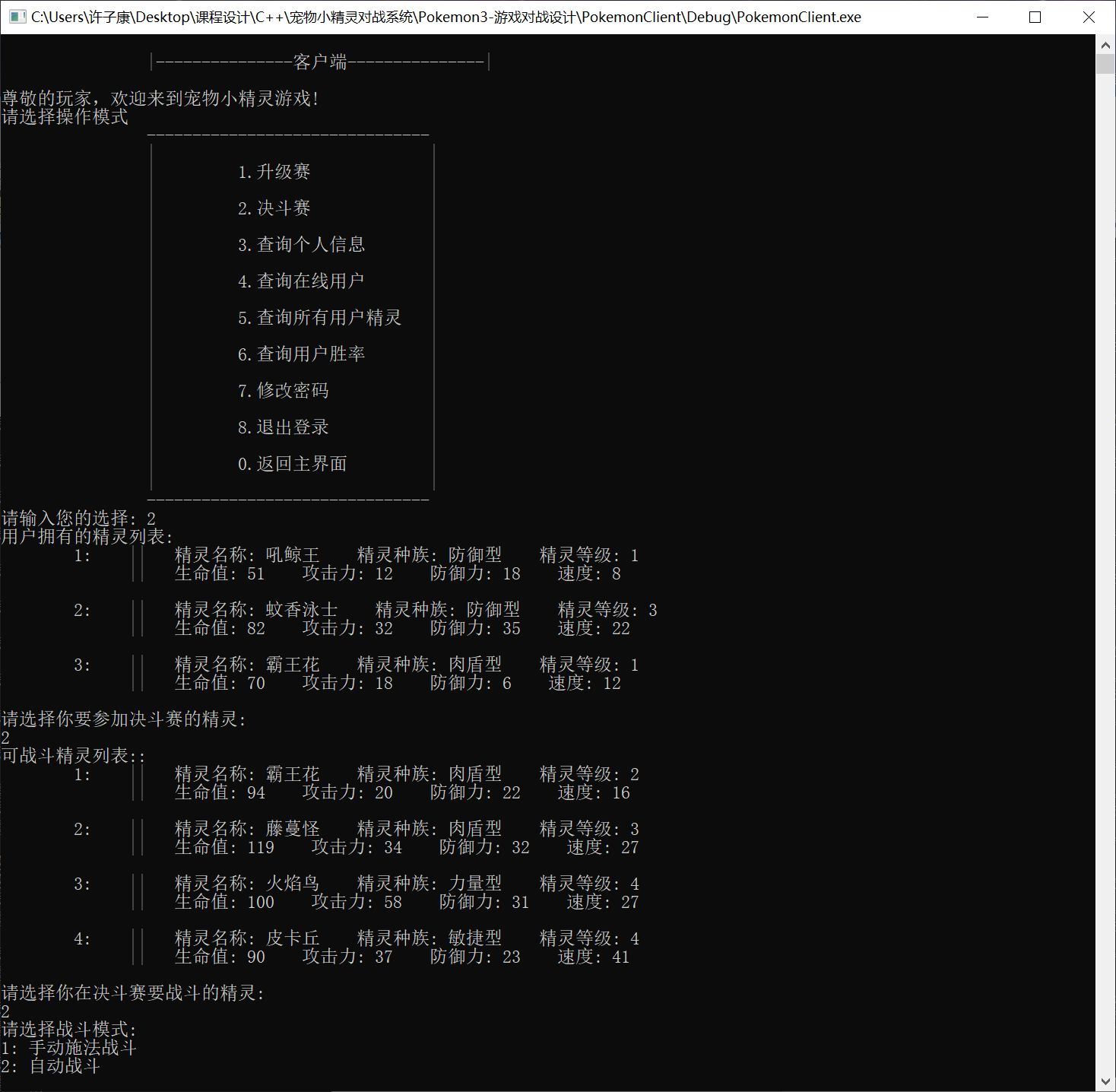


26. 战斗结束后，会显示精灵获取的经验值，如果精灵获取经验值之后升级，也会显示出相应信息。

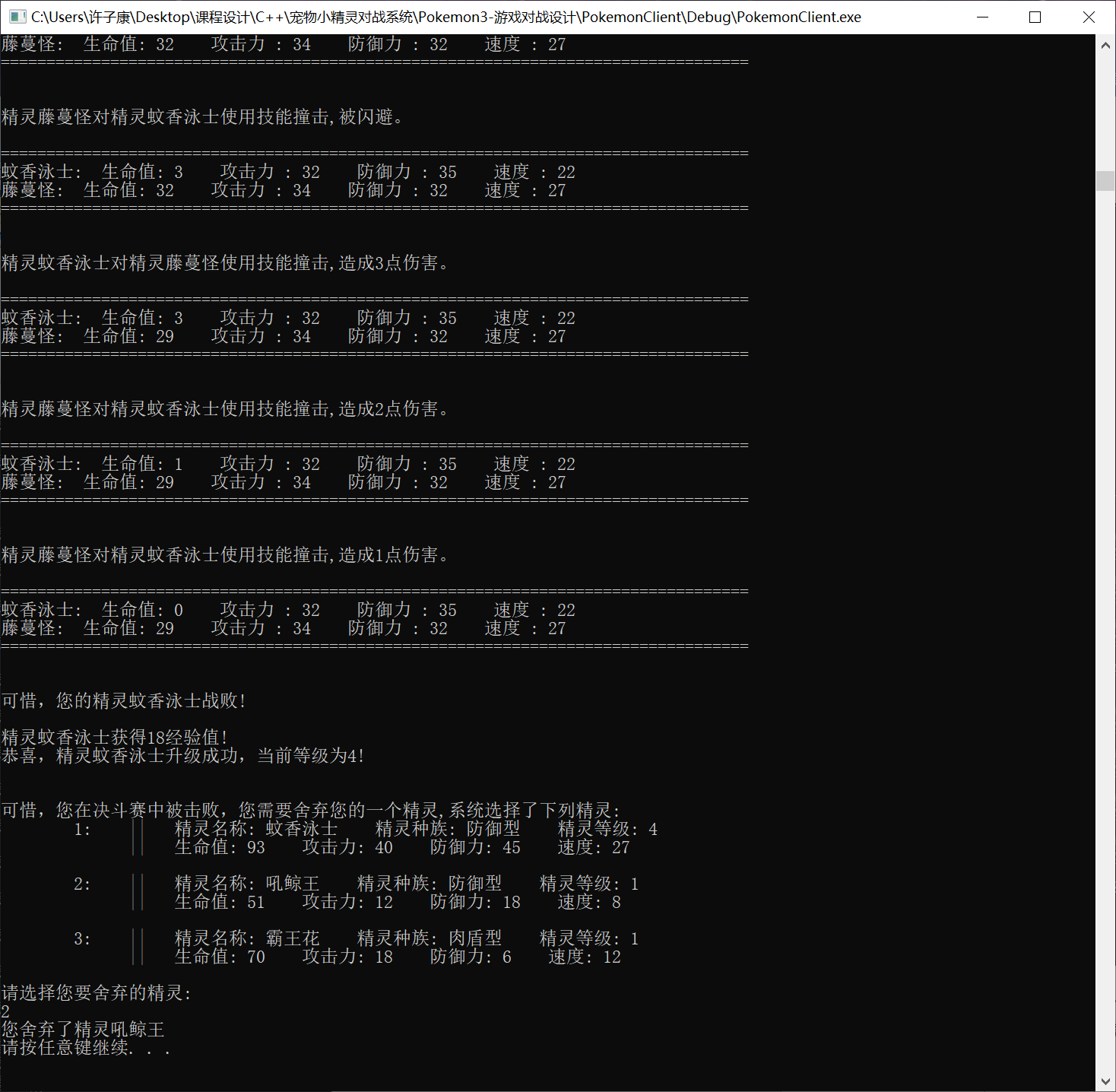
27. 选择自动战斗模式，两方精灵自动释放技能，如图：



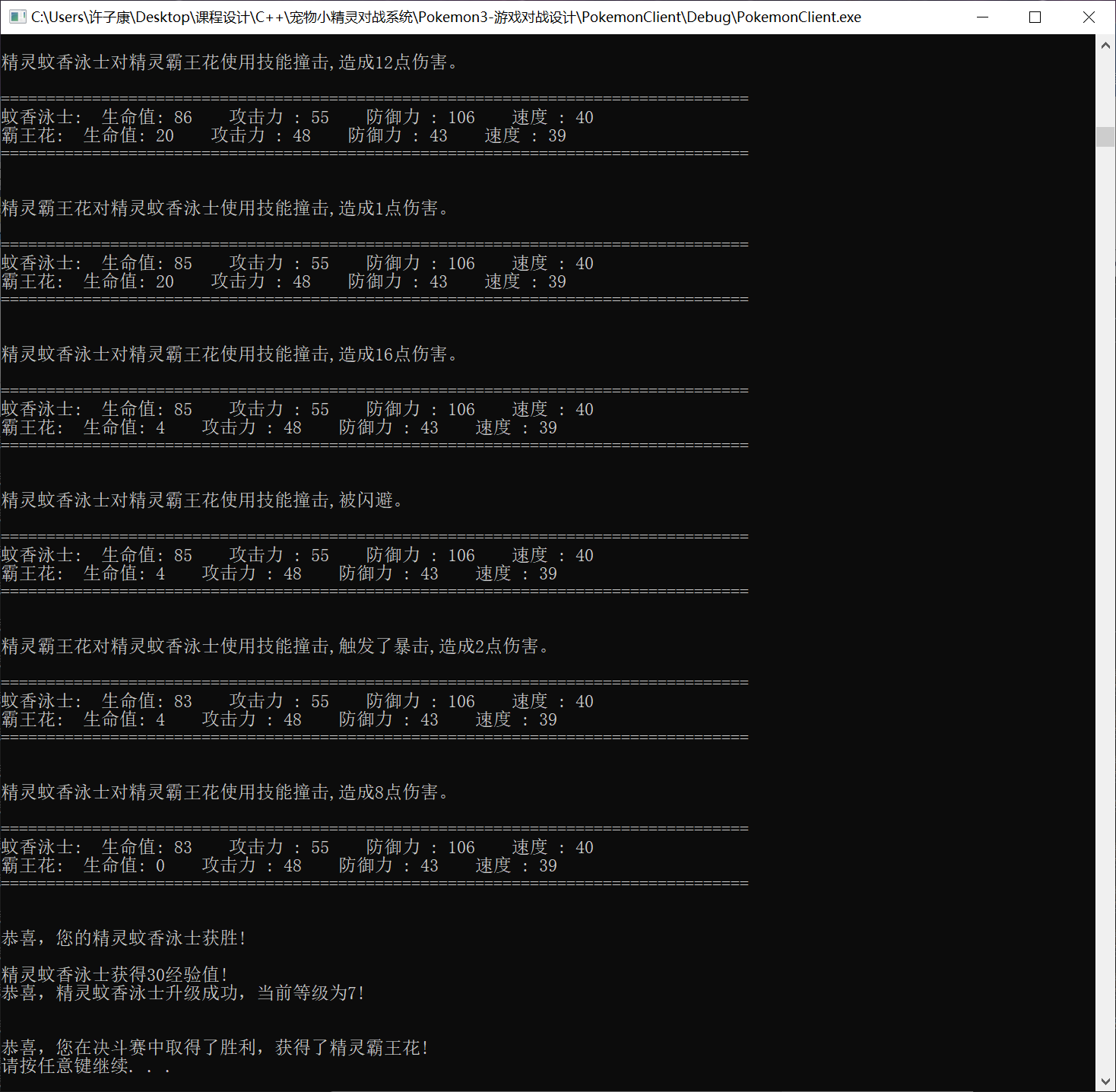
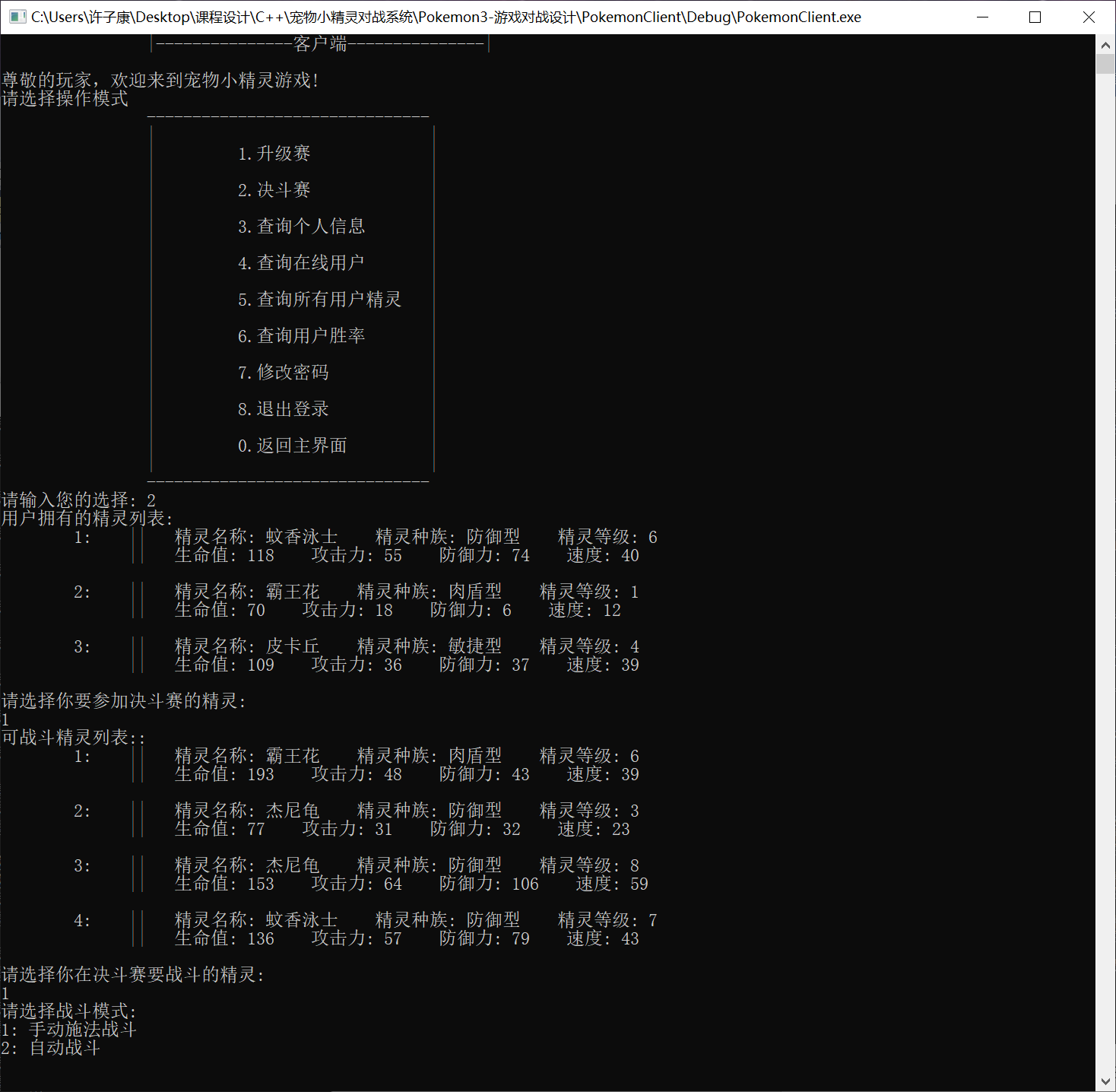
28. 进行决斗赛，前面流程和升级赛一致，选择双方精灵后，进行战斗，如图：



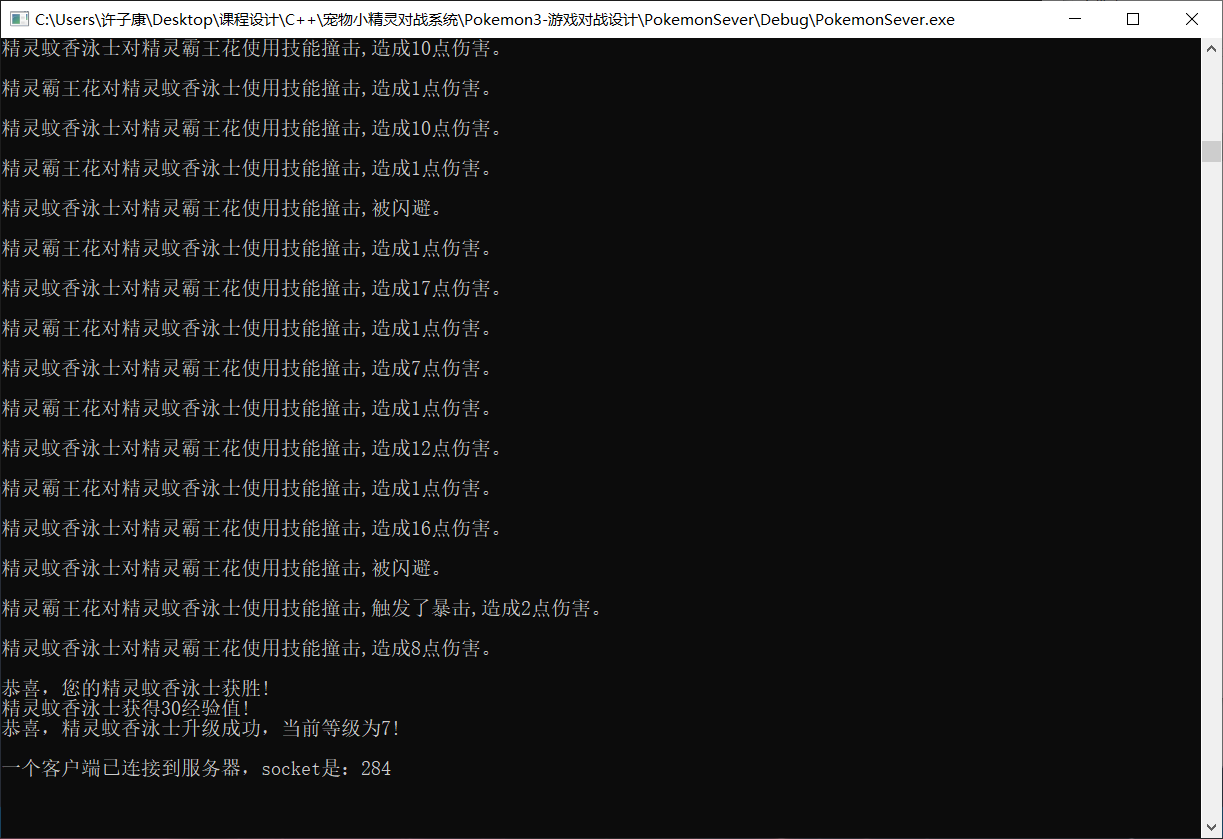
29. 在战斗之后，如果用户精灵失败，系统将随机选择用户的三只精灵，然后由用户选择一只精灵舍弃，如图：



30. 如果用户精灵获胜，则可以获得战败的精灵，如图：



31. 如果两个客户端同时进行操作，此时服务器会显示相应端口信息，如图：



32. 刚才第一个客户端的socket的id编号是288，新的是284，利用多线程实现了多用户同时操作的任务。

**三、代码内类和变量定义和实现**

1. 宝可梦四种种族：

//四种类型，力量型、肉盾型、防御型、敏捷型

#define PowerType 0

#define MeatShieldType 1

#define DefensiveType 2

#define AgileType 3

2. 宝可梦基类，PokemonBase：

class PokemonBase {

public:

PokemonBase(int Type);//构造函数

~PokemonBase() {};//析构函数

string raceName;//种族名称

int pokemonType;//精灵种类

int level;//等级

int expValue[16];//升到该级所需经验值

int baseAtk;//基础攻击力

int baseDef;//基础防御力

int baseHp;//基础生命值

int baseSp;//基础攻击速度

string skillName[5];//技能名称，普通攻击和四个特殊技能

//技能设定：普通攻击(1级)，特殊攻击技能1(1级)，增益技能(5级)，特殊攻击技能2(10级)，必杀技(15级)

int pp[5];//技能剩余使用次数

//攻击函数,true为击杀，false为未击杀

virtual bool attack(string& verStr, Pokemon &attacker, Pokemon &target, int skillId) = 0;

//闪避函数,true为闪避，false为未闪避

bool dodge(int atkSp, int tarSp);

};

其中攻击函数是纯虚函数，PokemonBase类为抽象类，不能实例对象，而且子类必须重写攻击函数。

3. 四种种族的信息类，四种宝可梦种族都继承了PokemonBase类，都重写了攻击函数：

class Power : public PokemonBase {//力量型种族

public:

Power();//构造函数

~Power() {};//析构函数

bool attack(string& verStr, Pokemon &attacker, Pokemon & target, int skillId);

};

class MeatShield : public PokemonBase {//肉盾型种族

public:

MeatShield();//构造函数

~MeatShield() {};//析构函数

bool attack(string& verStr, Pokemon &attacker, Pokemon & target, int skillId);

};

class Defensive : public PokemonBase {//防御型种族

public:

Defensive();//构造函数

~Defensive() {};//析构函数

bool attack(string& verStr, Pokemon &attacker, Pokemon & target, int skillId);

};

class Agile : public PokemonBase {//敏捷型种族

public:

Agile();//构造函数

~Agile() {};//析构函数

bool attack(string& verStr, Pokemon &attacker, Pokemon & target, int skillId);

};

4. 精灵宝可梦类，Pokemon：

class Pokemon {

public:

Pokemon();

Pokemon(int RaceId, string Name);//根据种族构造

Pokemon(int RaceId, string Name, int Level, int Exp, int Atk, int Def, int Hp, int Sp);//直接按照数据构造

Pokemon(const Pokemon& A);//赋值构造

~Pokemon() {};//析构函数

int \_raceId;//种类的编号

string \_name;//精灵名称

int \_level;//等级

int \_exp;//经验值

int \_atk, \_nowAtk;//攻击力

int \_def, \_nowDef;//防御力

int \_hp, \_nowHp;//血量

int \_sp, \_nowSp;//攻击速度

int \_nowPp[5];//技能剩余使用次数

static PokemonBase \*\_race[4];//四个精灵种类

//获取种族信息

string getRaceName() {//种类名称

return \_race[\_raceId]->raceName;

}

int getExpValue(int id) {

return \_race[\_raceId]->expValue[id];

}

string getSkillName(int id) {//技能名称

return \_race[\_raceId]->skillName[id];

}

int getPp(int id) {

return \_race[\_raceId]->pp[id];

}

//战斗信息

void recovery();//复原精灵信息

bool attack(string& verStr, Pokemon & target, bool autoFight = true, int oriSkillId = 0);//造成伤害，true为目标死亡，false未死亡

bool beAttacked(int val);//被伤害，true自己死亡，false未死亡

//暴击，true为暴击，false为未暴击

bool crit() {//力量型暴击率高于其他种族,暴击率和等级有关

if (\_raceId == PowerType) {

if (rand() % (23-\_level) == 0) return true;

return false;

}

if (rand() % (25-\_level) == 0) return true;

return false;

}

//升级

bool gainExp(string& verStr, int val);//升级函数，true升级，false未升级

bool gainExp(int val);//升级函数,true升级

//改变精灵当前信息

void changeAtk(int val);

void changeDef(int val);

void changeHp(int val);

void changeSp(int val);

//信息输出

void print();

};

static PokemonBase \*\_race[4]内包括每个种族精灵所固有的性质，包括种族名称、初始属性、升级后的增加属性、攻击方式、攻击命中率和暴击率。作为精灵类的静态成员变量，被所有精灵对象共享。

5. 用户类，Player:

class Player{//登陆、注册、登出、修改密码

public:

Player();

Player(string Name, string Id, string Key);

~Player() { };

bool loginStatus;//1为在线，0为不在线

string name;//用户名

string id;//登陆时账户

string key;//密码

int number;//精灵数量

int winNum, totNum;//比赛胜场数，比赛总场数，用于计算胜率

Pokemon pok[100];//所拥有的精灵

string numberMedal();//宠物个数徽章（金银铜）,根据拥有的宠物个数的多少颁发

string superPokemonMedal();//高级宠物勋章（金银铜），根据拥有的高级宠物（15级）个数颁发

//返回true表示密码修改成功

void changeKey(string newKey);

};

6. 闪避函数的实现，跟速度有关，带有一定的随机性，：

//闪避判断，跟两者速度有关，带有一定随机性，如果返回false表示未闪避，返回true表示闪避成功

bool PokemonBase::dodge(int atkSp, int tarSp) {

int sub = atkSp - tarSp;

if (sub > 30) return false;

else if (sub > 20) return rand() % 100 < 3;

else if (sub > 10)return rand() % 100 < 5;

else if (sub > 0) return rand() % 100 < 10;

else if (sub > -10) return rand() % 100 < 10;

else if (sub > -20) return rand() % 100 < 15;

else if (sub > -30) return rand() % 100 < 25;

else if (sub > -50) return rand() % 100 < 35;

else return rand() % 100 < 50;

}

7. 四个精灵种族，定义了技能名称和使用次数，重写了攻击函数

//==================四个精灵种类相关===================

Power::Power() :PokemonBase(PowerType) {//以小火龙为代表

raceName = "力量型";

skillName[0] = "撞击";//普通攻击

skillName[1] = "火焰旋涡";//伤害低级

skillName[2] = "愤怒";//强化攻击力

skillName[3] = "喷射火焰";//伤害中级

skillName[4] = "闪焰冲锋";//伤害高级

pp[0] = Inf;//普通攻击

pp[1] = 8;//特殊攻击技能

pp[2] = 2;//强化增益技能

pp[3] = 6;//特殊攻击技能

pp[4] = 4;//必杀技

}

//力量型精灵攻击

bool Power::attack(string& verStr, Pokemon & attacker, Pokemon & target, int skillId) {

cout << "精灵" << attacker.\_name << "对精灵" << target.\_name << "使用技能" << attacker.getSkillName(skillId) << ",";

verStr += "精灵" + attacker.\_name + "对精灵" + target.\_name + "使用技能" + attacker.getSkillName(skillId) + ",";

if (skillId == 0) {//撞击

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.1\*attacker.\_nowAtk - target.\_nowDef+getRand(0,attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 1) {//火焰旋涡

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.2\*attacker.\_nowAtk - target.\_nowDef + getRand(0, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 2) {//愤怒

attacker.changeAtk(attacker.\_nowAtk/10);

cout << "提升了自身" << attacker.\_nowAtk / 10 << "点攻击力。\n" << endl;

verStr += "提升了自身" + to\_string(attacker.\_nowAtk / 10) + "点攻击力。\n"; verStr += "#";

return false;

}

else if (skillId == 3) {//喷射火焰

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.3\*attacker.\_nowAtk - target.\_nowDef + getRand(0, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 4) {//闪焰冲锋

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.6\*attacker.\_nowAtk - target.\_nowDef + getRand(0, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

return false;

}

MeatShield::MeatShield() : PokemonBase(MeatShieldType) {//以妙蛙种子为代表

raceName = "肉盾型";

skillName[0] = "撞击";//普通攻击

skillName[1] = "藤编";//低伤

skillName[2] = "寄生种子";//生命力回复

skillName[3] = "飞叶快刀";//中伤

skillName[4] = "阳光烈焰";//高伤

pp[0] = Inf;//普通攻击

pp[1] = 6;//特殊攻击技能

pp[2] = 6;//强化增益技能

pp[3] = 6;//特殊攻击技能

pp[4] = 2;//必杀技

}

//肉盾型精灵攻击

bool MeatShield::attack(string& verStr, Pokemon & attacker, Pokemon & target, int skillId) {

cout << "精灵" << attacker.\_name << "对精灵" << target.\_name << "使用技能" << attacker.getSkillName(skillId) << ",";

verStr += "精灵" + attacker.\_name + "对精灵" + target.\_name + "使用技能" + attacker.getSkillName(skillId) + ",";

if (skillId == 0) {//撞击

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 1) {//藤编

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.1\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 2) {//寄生种子

attacker.changeHp(attacker.\_nowHp / 6);

cout << "提升了自身" << attacker.\_nowHp / 6 << "点生命值。\n" << endl;

verStr += "提升了自身" + to\_string(attacker.\_nowHp / 6) + "点生命值。\n"; verStr += "#";

return false;

}

else if (skillId == 3) {//飞叶快刀

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.2\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 4) {//阳光烈焰

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.4\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

return false;

}

Defensive::Defensive() : PokemonBase(DefensiveType) {//以杰尼龟为代表

raceName = "防御型";

skillName[0] = "撞击";//普通攻击

skillName[1] = "水枪";//低伤

skillName[2] = "缩壳";//防御力提高

skillName[3] = "泡沫光线";//中伤

skillName[4] = "鱼跃龙门";//高伤

pp[0] = Inf;//普通攻击

pp[1] = 8;//特殊攻击技能

pp[2] = 4;//强化增益技能

pp[3] = 6;//特殊攻击技能

pp[4] = 2;//必杀技

}

bool Defensive::attack(string& verStr, Pokemon & attacker, Pokemon & target, int skillId) {

cout << "精灵" << attacker.\_name << "对精灵" << target.\_name << "使用技能" << attacker.getSkillName(skillId) << ",";

verStr += "精灵" + attacker.\_name + "对精灵" + target.\_name + "使用技能" + attacker.getSkillName(skillId) + ",";

if (skillId == 0) {//撞击

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 1) {//水枪

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.1\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 2) {//缩壳

attacker.changeDef(attacker.\_nowDef / 10);

cout << "提升了自身" << attacker.\_nowDef / 10 << "点防御力。\n" << endl;

verStr += "提升了自身" + to\_string(attacker.\_nowDef / 10) + "点防御力。\n"; verStr += "#";

return false;

}

else if (skillId == 3) {//泡沫光线

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.2\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 4) {//鱼跃龙门

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.4\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

return false;

}

Agile::Agile() : PokemonBase(AgileType) {//以皮卡丘为代表

raceName = "敏捷型";

skillName[0] = "撞击";//普通攻击

skillName[1] = "电击";//低伤

skillName[2] = "高速移动";//速度提高

skillName[3] = "十万伏特";//中伤

skillName[4] = "电磁炮";//高伤

pp[0] = Inf;//普通攻击

pp[1] = 7;//特殊攻击技能

pp[2] = 4;//强化增益技能

pp[3] = 6;//特殊攻击技能

pp[4] = 3;//必杀技

}

bool Agile::attack(string& verStr, Pokemon & attacker, Pokemon & target, int skillId) {

cout << "精灵" << attacker.\_name << "对精灵" << target.\_name << "使用技能" << attacker.getSkillName(skillId) << ",";

verStr += "精灵" + attacker.\_name + "对精灵" + target.\_name + "使用技能" + attacker.getSkillName(skillId) + ",";

if (skillId == 0) {//撞击

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 1) {//电击

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.1\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 2) {//高速移动

attacker.changeSp(attacker.\_nowSp / 10);

cout << "提升了自身" << attacker.\_nowSp / 10 << "点速度。\n" << endl;

verStr += "提升了自身" + to\_string(attacker.\_nowSp / 10) + "点速度。\n"; verStr += "#";

return false;

}

else if (skillId == 3) {//十万伏特

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.2\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

else if (skillId == 4) {//电磁炮

if (dodge(attacker.\_nowSp, target.\_nowSp)) {

cout << "被闪避。\n" << endl;

verStr += "被闪避。\n"; verStr += "#";

return false;

}

int val = (double)1.4\*attacker.\_nowAtk - target.\_nowDef + getRand(-5, attacker.\_level);

val = max(val, 1);

if (attacker.crit()) {

val \*= 2;

cout << "触发了暴击,";

verStr += "触发了暴击,";

}

cout << "造成" << val << "点伤害。\n" << endl;

verStr += "造成" + to\_string(val) + "点伤害。\n"; verStr += "#";

return target.beAttacked(val);

}

return false;

}

PokemonBase \*Pokemon::\_race[4] = { new Power(),new MeatShield(),new Defensive(),new Agile() };

8. Pokemon类的攻击函数，利用类中静态种族类，实现攻击：

//攻击敌方精灵,如果敌方精灵死亡，返回true，否则返回false

bool Pokemon::attack(string& verStr, Pokemon & target, bool autoFight, int oriSkillId) {

if (autoFight) {//自动攻击，优先选择技能，没有技能使用普通攻击

bool flag[5];

int tot = 0;

for (int i = 1; i <= 4; i++) {

if (\_level >= (i - 1) \* 5 && \_nowPp[i]) {

flag[i] = true;

tot += \_nowPp[i];

}

else flag[i] = false;

}

if (tot==0){//没有技能，使用普通攻击

return \_race[\_raceId]->attack(verStr, \*this, target, 0);

}

//有技能，优先使用技能

int num = rand() % tot + 1,skillId=0;

for (int i = 1; i <= 4; i++) {

if (flag[i]) {

if (num <= \_nowPp[i]) {

skillId = i;

\_nowPp[i]--;

break;

}

else {

num -= \_nowPp[i];

}

}

}

return \_race[\_raceId]->attack(verStr, \*this, target, skillId);

}

else {

/\*for (int i = 0; i <= 4; i++) {

if (\_level >= (i - 1) \* 5 && \_nowPp[i]) {

cout << "技能" << i << ": " << \_race[\_raceId]->skillName[i] << ",剩余使用次数";

if (i == 0) cout << "无限" << endl;

else cout << \_nowPp[i] << endl;

}

}

cout << "请输入使用的技能编号:";

int skillId; cin >> skillId;\*/

if (\_level >= (oriSkillId - 1) \* 5 && \_nowPp[oriSkillId]) {

if (oriSkillId >= 1) \_nowPp[oriSkillId]--;

return \_race[\_raceId]->attack(verStr, \*this, target, oriSkillId);

}

else {

cout << "输入无效，请重新操作" << endl;

return false;

}

}

return false;

}

9. 精灵宝可梦的获取经验值和升级函数：

//获得经验值，判断升级,true为升级，false未升级

bool Pokemon::gainExp(int val) {

val = max(val, 1);

cout << "精灵" << \_name << "获得" << val << "经验值!" << endl;

if (\_level == 15) {

cout << "精灵已满级，不能再升级!" << endl;

return false;

}

\_exp += val;

bool tag = false;

while (\_level < 15 && \_exp >= \_race[\_raceId]->expValue[\_level + 1]) {

tag = true;

\_level++;

\_atk += 5 + getRand(0, 4);

\_def += 4 + getRand(0, 6);

\_hp += 10 + getRand(0, 8);

\_sp += 4 + getRand(0, 4);

if (\_raceId == PowerType) \_atk += 5;

else if (\_raceId == MeatShieldType) \_hp += 10;

else if (\_raceId == DefensiveType) \_def += 5;

else if (\_raceId == AgileType) \_sp += 3;

}

if (tag) {

cout << "恭喜，精灵" << \_name << "升级成功，当前等级为" << \_level << "!" << endl << endl;

return true;

}

cout << endl;

return false;

}

10. 服务器TCPSocket通信，通过多线程持续接收请求：

// 1 初始化

WSADATA wsadata;

WSAStartup(MAKEWORD(2, 2), &wsadata); //make word,你把鼠标移到WSAStartup看看参数列表,是不是就是一个word啊

// 2 创建服务器的套接字

SOCKET serviceSocket;

serviceSocket = socket(AF\_INET, SOCK\_STREAM, 0); //socket(协议族,socket数据传输方式,某个协议) 我们默认为0,其实就是一个宏

if (SOCKET\_ERROR == serviceSocket) {

cout << "套接字创建失败....." << endl;

Sleep(1000);

return 0;

}

else {

cout << "套接字创建成功....." << endl;

}

// 3 绑定套接字 指定绑定的IP地址和端口号

sockaddr\_in socketAddr; //一个绑定地址:有IP地址,有端口号,有协议族

socketAddr.sin\_family = AF\_INET;

socketAddr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

socketAddr.sin\_port = htons(1234);

int bRes = bind(serviceSocket, (SOCKADDR\*)&socketAddr, sizeof(SOCKADDR)); //绑定注意的一点就是记得强制类型转换

if (SOCKET\_ERROR == bRes) {

cout << "绑定失败....." << endl;

cout << "正在退出服务端，请重新启动!" << endl;

Sleep(1000);

return 0;

}

else {

cout << "绑定成功....." << endl;

}

// 4 服务器监听

int lLen = listen(serviceSocket, 20); //监听的第二个参数就是:能存放多少个客户端请求,到并发编程的时候很有用哦

if (SOCKET\_ERROR == lLen) {

cout << "监听失败....." << endl;

cout << "正在退出服务端，请重新启动!" << endl;

Sleep(1000);

return 0;

}

else {

cout << "监听成功....." << endl;

}

//从playerInformationStore.txt里面读取信息，获取历史记录

GetInformationFromFile();

while (true) {//多线程，循环接收客户端连接请求并创建服务线程

SOCKET \*ClientSocket = new SOCKET;

ClientSocket = (SOCKET\*)malloc(sizeof(SOCKET));

//接收客户端连接请求

int SockAddrlen = sizeof(sockaddr);

\*ClientSocket = accept(serviceSocket, 0, 0);

cout << "一个客户端已连接到服务器，socket是：" << \*ClientSocket << endl;

CreateThread(NULL, 0, &ServerThread, ClientSocket, 0, NULL);

}

closesocket(serviceSocket);

// 8 终止

WSACleanup();

cout << "服务器停止" << endl;

11. 客户端TCPSocket通信，向服务器发出请求：

// 1 初始化

WSADATA wsadata;

WSAStartup(MAKEWORD(2, 2), &wsadata);

// 2 创建套接字

SOCKET clientSocket = {};

clientSocket = socket(PF\_INET, SOCK\_STREAM, 0);

if (SOCKET\_ERROR == clientSocket) {

cout << "套接字创建失败....." << endl;

cout << "正在退出客户端，请重新启动!" << endl;

Sleep(1000);

return 0;

}

else {

cout << "套接字创建成功....." << endl;

}

// 3 绑定套接字 指定绑定的IP地址和端口号

sockaddr\_in socketAddr;

socketAddr.sin\_family = PF\_INET;

socketAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

socketAddr.sin\_port = htons(1234);

int cRes = connect(clientSocket, (SOCKADDR\*)&socketAddr, sizeof(SOCKADDR));

if (SOCKET\_ERROR == cRes) {

cout << "与服务器连接失败....." << endl;

cout << "正在退出客户端，请重新启动!" << endl;

Sleep(1000);

return 0;

}

else {

cout << "与服务器连接成功....." << endl;

}