

面向对象程序设计实践（C++）

课程设计报告

——宠物小精灵对战系统



姓 名：崔 思 颖

学 号：2018211290

学 院：计算机学院

班 级：2018211307

目录

[任务分析 4](#_Toc50123914)

[问题描述 4](#_Toc50123915)

[功能要求 4](#_Toc50123916)

[任务分析 5](#_Toc50123917)

[模块设计 6](#_Toc50123918)

[设计思路 6](#_Toc50123919)

[模块图示 7](#_Toc50123920)

[详细设计 9](#_Toc50123921)

[宠物小精灵的加入 9](#_Toc50123922)

[用户注册与平台登录 14](#_Toc50123923)

[游戏对战的设计 23](#_Toc50123924)

[友元说明 28](#_Toc50123925)

[程序结果演示 29](#_Toc50123926)

[运行环境说明 38](#_Toc50123927)

[综合实验一 38](#_Toc50123928)

[综合实验二 39](#_Toc50123929)

[综合实验三 39](#_Toc50123930)

[小结 39](#_Toc50123931)

# 任务分析

## 问题描述

本实验要求用面向对象的设计方法来设计一款平台类对战游戏，包含宠物小精灵的加入、用户注册与平台登录、游戏对战的设计三个题目。

## 功能要求

**题目一：宠物小精灵的加入**

* 设计宠物小精灵的类，为简化游戏设计，精灵的属性包括种类（力量型：高攻击；肉盾型：高生命值； 防御型：高防御； 敏捷型：低攻击间隔，共 四种）、名字、等级、经验值、攻击力、防御力、生命值、攻击间隔等（以上属性必须，其他属性可自行添加）（基本要求：本游戏中只有上面的 4 种 类型。 进一步要求：上述 4 种类型可以进一步深入划分，比如力量型又可 以再细分为：沙瓦朗、火爆猴、腕力等）。
* 每个精灵初始等级为 1，满级15级，每当精灵升级的时候，宠物对应的属性值会有少量增加（主属性增加量相对较多）。
* 每个精灵有自己独特的攻击方式，如“闪电攻击”，“火焰攻击”等等，请设计一个精灵的基类，并将精灵的攻击方法设为虚方法以方便子类重写。
* 请写一个测试程序对设计的精灵类的相关属性和方法（包括攻击函数，升级 函数等）进行测试
* 题目主要考察点：类的继承，对象数据成员设计，成员函数设计

**题目二：用户注册与平台登录**

* 每个用户需要注册一个账号，用户名全局唯一，不能有任何两个用户名相同， 要考虑注册失败的场景时的反馈。
* 实现注册、登录、登出功能，均采用 C/S 模式，客户端和服务端用 socket 进行通信，服务端保存所有用户的信息，每个用户拥有：用户名、拥有的精灵，两个属性。 用户注册成功时，系统 自动随机分发三个 1 级精灵给用户。
* 用户可以查看所有成功注册用户拥有的精灵，也可以查看所有当前在线的用户。
* 题目主要考察点：socket 通信，交互场景反馈，用户信息存储方式，界面交互，其它合理的新颖设计。

**题目三：游戏对战的设计**

* 已经登录的在线用户可以和服务器进行虚拟决斗，决斗分两种：升级赛和决斗赛，两种比赛都能增长精灵宠物经验值。服务器上有一个虚拟精灵的列表， 用户可以挑选其中任意一个进行比赛（升级赛或者决斗赛）。另外决斗赛中用户胜出可以直接获得该战胜的精灵，失败则系统从用户的精灵中随机选三个（不够三个精灵的情况就选择他所有的精灵），然后由用户选一个送出。
* 升级赛 只是用户用来增加精灵经验值，规则开发者自定。
* 累积多少经验值升一级，规则开发者自定。
* 决斗赛的上述规则同升级赛，只是额外还可以赢得一个宠物或失去一个宠物。
* 用户如果没有精灵（比如总是失败，已经全部送出去），则系统会随机放给他一个初级精灵。
* 请让你的系统自动模拟每场比赛的每次出招。另外，为了增加不确定性，可以加入概率闪避攻击和暴击伤害机制。
* 比赛的过程和结果由系统根据上述规则自动模拟完成，要求结果具有一定的随机性。
* 用户增加新功能，可以查看某个用户的胜率。
* 用户增加新属性，为宠物个数徽章（金银铜）和高级宠物徽章（金银铜）， 分别根据拥有的宠物个数的多少和拥有高级宠物（15 级）个数的多少颁发。
* 题目主要考察点：客户端与服务器数据交互（可采用多进程或异步通信或其他方法均可），并发请求处理，类的方法设计，伤害计算方法设计，界面交互，其它合理的新颖设计。

**软件设计要求**

* 如有必要的友元函数，要在报告（课程设计报告）和程序中说明每个友 元函数的不可替代性，为什么一定要用友元才能实现。
* 自己编写的代码，除主函数和必要的友元函数外，不允许出现任何一个非类成员函数。
* 任何不改变对象状态（不改写自身对象数据成员值）的成员函数均需显式标注 const。

**代码规范性要求**

* 代码需遵循课件提出的编码规范要求。
* 通过开发环境自动生成的界面类代码，全部数据成员和成员函数需在类 声明时加以注释，函数体内的必要步骤要加以注释。
* 其他全部类代码的数据成员和成员函数的声明和实现均需加以注释，成员函数的必要步骤要加以注释。

## 任务分析

本实验的最终成果是一个游戏，玩法方面，题目已经策划完善，其核心在于对战加收集，玩家的成长路线就是不断在升级赛与决斗赛之间切换战斗，逐渐对各种宠物精灵属性的运用得心应手，收获多枚勋章，成为最强训练家；界面交互方面，由题目要求我联想到了市面上已经存在的“赛尔号”、“拳皇”、“Q宠大乐斗”等回合制对战游戏，因此ui的设计思路可以借鉴，大致分为登录界面、大厅界面、战斗界面；底层实现方面，题目对编码要求很细致，也为我提供了很多思路，难点就在于我能否顺利将这一学期学到的面向对象程序设计理念运用在代码中了。

# 模块设计

## 设计思路

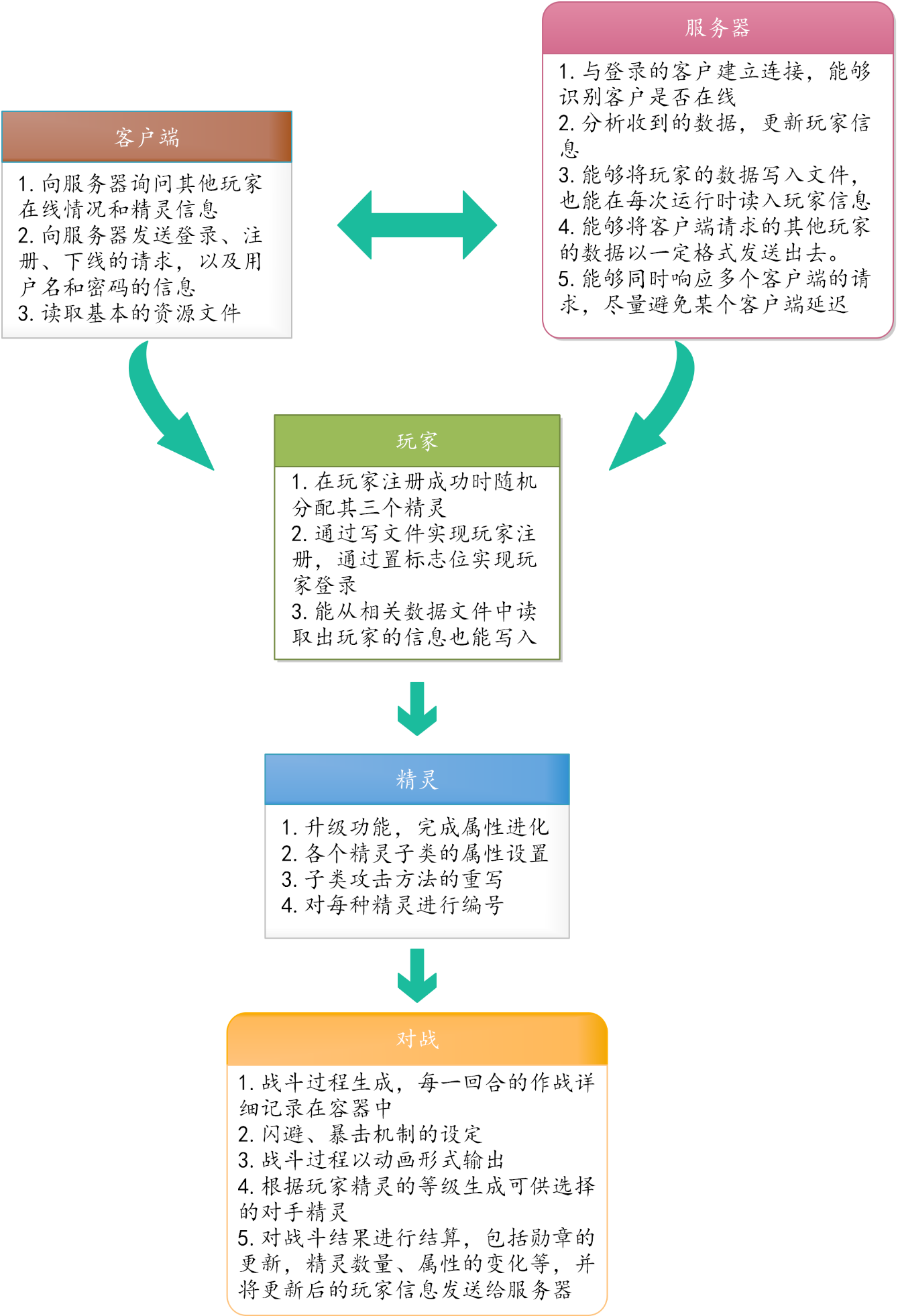
* **代码逻辑**

我将底层逻辑实现划分为四个模块。首先把socket通信功能独立出来，将其划分为客户端和服务器两个模块，通信的内容即为玩家的数据，因此第三个模块为玩家模块。每个玩家拥有的精灵属于游戏的核心设计部分，第四个模块即为精灵模块，包含对精灵的多种操作。最后一个模块为对战模块，这个模块需要设计对战机制和伤害公式，传入双方精灵即可生成对战过程。

* **用户界面**

用户打开游戏的第一步操作应是与服务器建立连接，因此第一个界面为登录或注册界面，系统读取用户在界面中输入的信息，将其发送给服务器，服务器返回玩家信息。若登录成功，即可进入游戏的主要界面——大厅界面。在大厅界面有多个选项可供选择，包括查看精灵背包、查看其他用户信息、进行比赛、登出。若用户选择战斗，则切换至战斗界面，用户可以选择自己的出战精灵和对手精灵。确定交战精灵后，系统播放对战动画，战斗结束进行结算，之后再次返回大厅界面。

## 模块图示



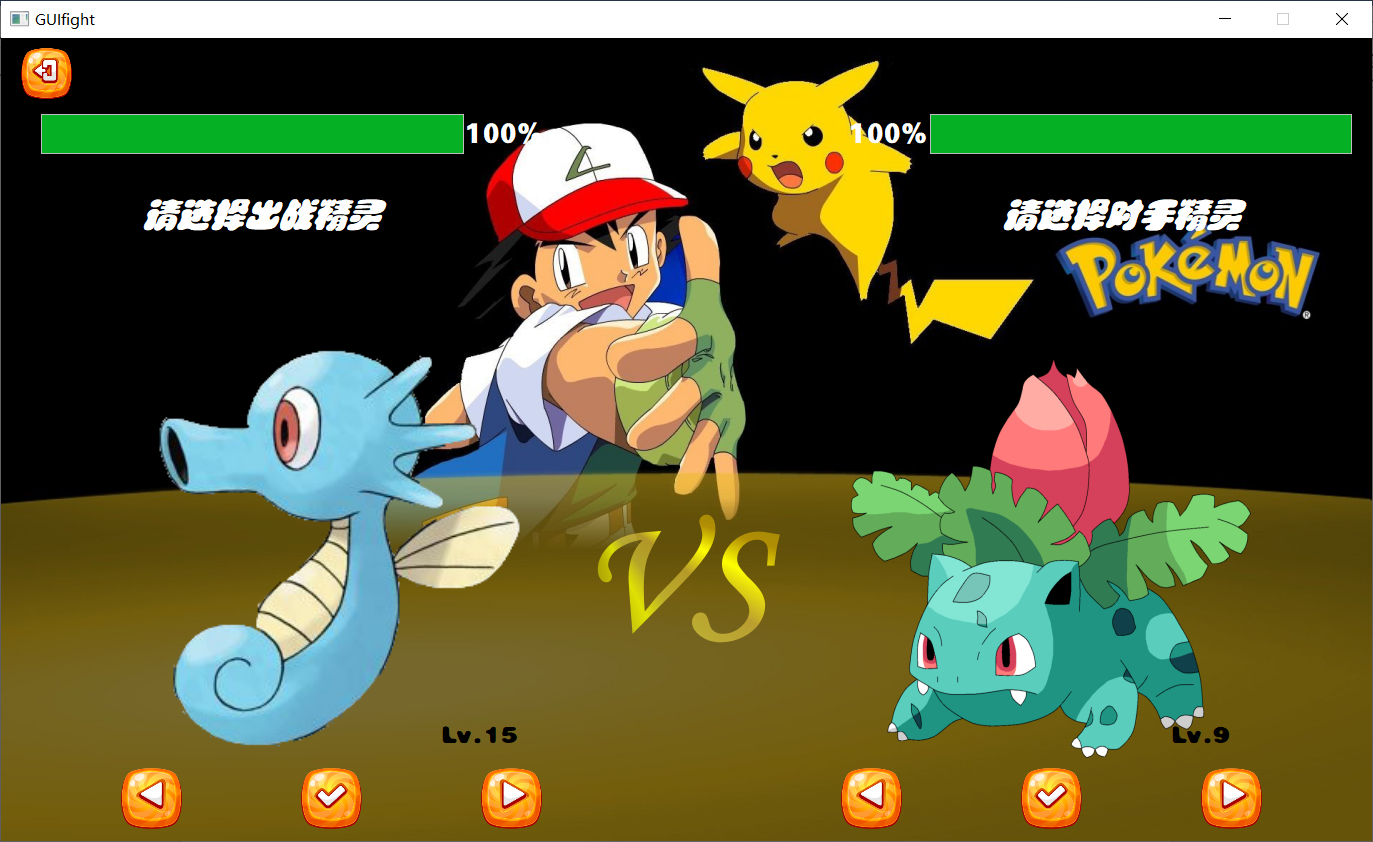
**登录注册界面**



**大厅界面**



**对战界面**



# 详细设计

开发与测试环境：Windows 10 VS2017+Qt Designer。

## 宠物小精灵的加入

系统总共内置了16种宠物精灵，力量型、肉盾型、防御型、敏捷型各四种，每种精灵有自己独特的属性值和攻击招式。

* **宠物小精灵类的设计**

精灵基类：

class Pokemon {

protected:

enum Type { ATK, HP, DF, ATKT };//精灵种类：力量型、肉盾型、防御型、敏捷型

int pokSeq;//精灵编号

Type type;//种类

string name;//名字

string lettername;//拼音名字

int lv = 1;//等级

float atk;//攻击力

float df;//防御力

float atkt;//攻击间隔

float hp;//生命值

public:

float exp = 0.0;//经验值

void initPok(int Lv, float Exp)//根据精灵等级和经验值初始化精灵

{

if (this != NULL)

{

while (lv < Lv) this->upgrade(0);

exp = Exp;

}

}

float getDf() { return df; }//返回防御力值

float getHp() { return hp; }//返回生命上限

int getLv() { return lv; }//返回等级

float getAtkt() { return atkt; }//返回攻击间隔

string getLetterName() { return lettername; };//返回拼音名字

string getName() { return name; };//返回名字

int getSeq() { return pokSeq; }//返回精灵编号

void printProperty(Pokemon pok, stringstream &out);//打印精灵属性

void upgrade(float newExp);//升一级

Pokemon\* newPok(int Pok);//根据编号返回一个创建的精灵指针

virtual ATTACK attack(Pokemon& pok, int choice);//攻击方法（虚方法）

void deletePok(Pokemon\* pok) { delete pok; }//删除已经创建的精灵

};

精灵子类：（以妙蛙种子为例）

class miaoWaZhongZi:public Pokemon

{

public:

miaoWaZhongZi()

{

pokSeq = 1;

type = ATK;

name = "妙蛙种子";

lettername = "miaowazhongzi";

hp = 45;

atk = 49;

df = 49;

atkt = 22;

}

virtual ATTACK attack(Pokemon& pok, int choice) override; //重写攻击方法函数

};

* **升级函数**

void Pokemon::upgrade(float newExp)//升一级属性值少量增加，主属性增加量相对较多

{

if (lv < 15) //15级为最高级别

{

switch (type) //主属性提升1.5倍，其他属性提升1.2倍

{

case ATK:

atk \*= UPGTYPE;

hp \*= UPG;

df \*= UPG;

atkt /= UPG;

break;

case HP:

atk \*= UPG;

hp \*= UPGTYPE;

df \*= UPG;

atkt /= UPG;

break;

case DF:

atk \*= UPG;

hp \*= UPG;

df \*= UPGTYPE;

atkt /= UPG;

break;

case ATKT:

atk \*= UPG;

hp \*= UPG;

df \*= UPG;

atkt /= UPGTYPE;

break;

}

lv++;//升一级

exp = newExp; //经验值重置

}

}

* **攻击方法**

攻击结构体：

typedef struct attack

{

string pok="";//出招精灵

string name = "";//招式名称

int seq = 0;//招式序号

float damage = 0;//攻击伤害

bool learn = 0;//是否习得

float hpOppo = 0;//攻击后敌方剩余血量

bool dodge = 0;//是否被敌方闪避

bool strike = 0;//是否暴击

}ATTACK;

父类攻击方法函数：

ATTACK Pokemon::attack(Pokemon& pok, int choice)

{

ATTACK atkNull; //只进行了初始化的攻击

return atkNull;

}

子类攻击方法函数：（以妙蛙种子为例）

ATTACK miaoWaZhongZi::attack(Pokemon& pok,int choice) //pok为对手精灵，choice为选择的招式编号，若为0则是随机出招

{

ATTACK Atk;

int power=0, moves;

//根据等级来确定招式数

if (lv >= 12) moves = 3;

else if (lv >=4) moves = 2;

else moves = 1;

if(choice==0) choice = (rand() % moves) + 1;//随机选择招式

if (choice <= moves) Atk.learn = 1; //该招式已学会

Atk.seq = choice;

switch (choice)

{

case 1:

Atk.name = "藤鞭";//招式名称

power = 45; //招式威力

break;

case 2:

Atk.name = "飞叶快刀";

power = 55;

break;

case 3:

Atk.name = "种子炸弹";

power = 80;

break;

default: return Atk;

break;

}

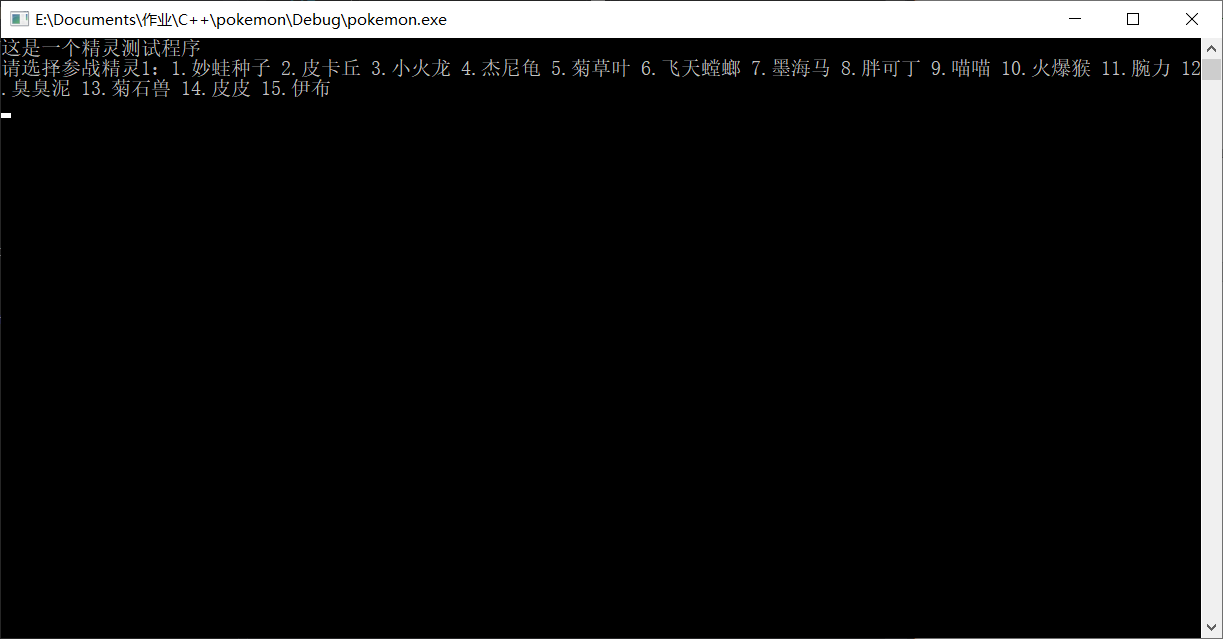
Atk.damage = ((lv\*(float)0.4 + 2)\*power\*atk / (pok.getDf() \* 50) + 2)\*

(217 + rand() / float(RAND\_MAX) \* 38) / 255; return Atk; //根据等级、招式、攻击力、对手防御值设计了特定公式，计算出本次攻击的伤害，并且加入一定随机性

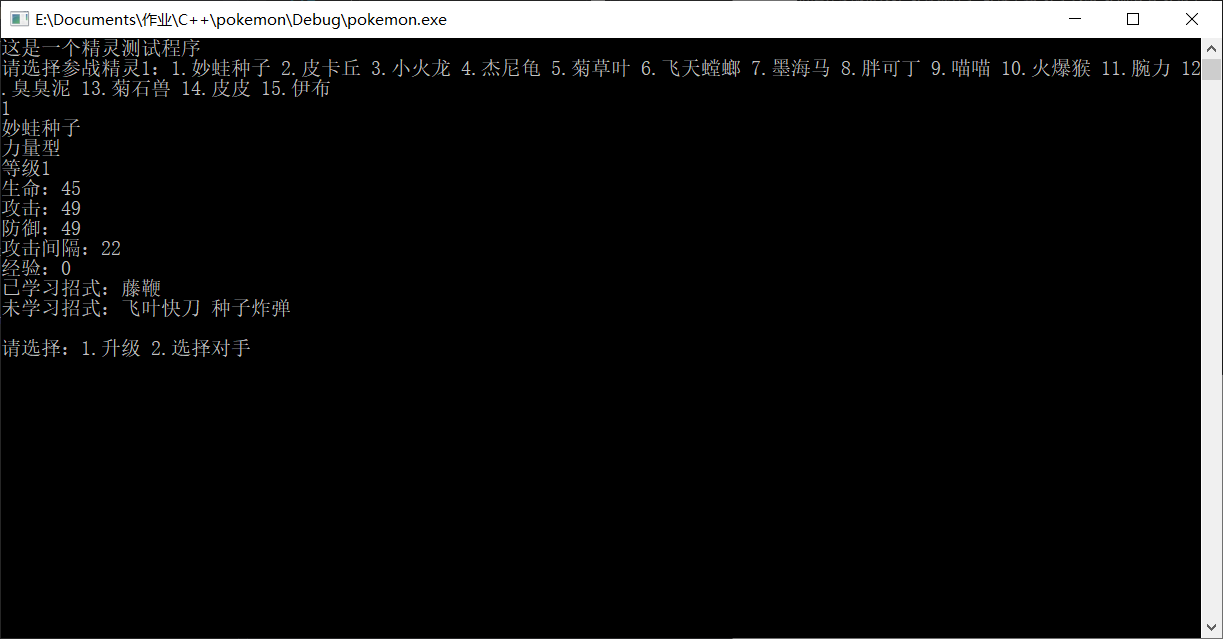
}

* **测试程序**

以妙蛙种子与皮卡丘为例进行测试。



由程序提示，输入妙蛙种子的编号1、回车。



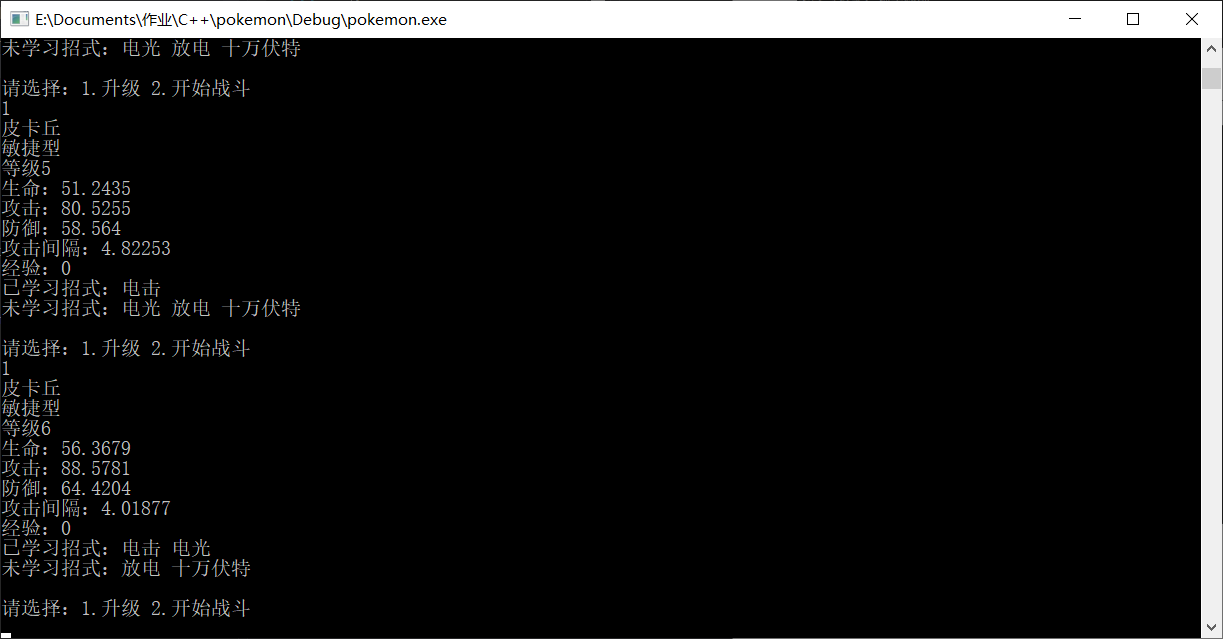
多次输入1、回车，将妙蛙种子升级至7级。



输入2，选择对手为皮卡丘。



将皮卡丘升至6级。



输入2，开始战斗，程序输出每回合作战状态，包括每次攻击的伤害。





## 用户注册与平台登录

一个用户即为一个player对象，服务器端将每个用户的信息写入文件playerPok.txt中，在连接时读入。用户登录成功后，服务器会发送其精灵信息给客户端。用户也可以发送查询消息，服务器将把所有玩家密码以外的信息，包括精灵、等级、胜率、在线情况，发送给客户端，从而可以查看所有成功注册用户拥有的精灵及胜率，也可以查看所有当前在线的用户。

* **玩家类**

玩家类除用户名及密码设置为private，其余均为public，方便战斗后对玩家的数据进行更新：

class player

{

private:

string name,password;//用户名和密码

public:

int pokNum;//精灵数

int winnum=0;//赢的场次

int totalnum=0;//比赛总共次数

bool medal[7];//勋章

vector<Pokemon\*> poks;//拥有的精灵

int Login;//是否已登录

player(string Name) { name = Name; };//由用户名构造玩家对象

void assignPok();//随机分配三个一级精灵

bool login(string Name,player& p);//登录

void friend readPlayers(string s);//从字符流中读取一个玩家数据

void printPlayerInfo(int pokInfo,stringstream &out);//打印玩家信息，可以选择是否打印精灵详情

SOCKET connectServer();//建立连接，返回服务器socket

bool enter(SOCKET socketServer,int choice,string Password);//进入游戏，注册或登录

string getName() { return name; }

string getPassword() { return password; }

};

* **通信方法**

我采用了事件选择网络模型实现客户端与服务器的通信，服务器端将系统监听到的用户事件分为ACCEPT、READ、CLOSE三类。获取到ACCEPT事件则与新用户建立连接，加入到已有客户端列表；获取到READ事件则处理用户发来的玩家信息数据包，根据包头choice的值进行相应操作；获取CLOSE事件则清除相应下线客户端。

服务器连接：

bool server::connectClient()

{

//配置版本库

WORD wdVersion = MAKEWORD(2, 2);

WSADATA wdSockMsg;

int nRes = WSAStartup(wdVersion, &wdSockMsg);

if (0 != nRes)

{

switch (nRes)

{

case WSASYSNOTREADY://系统配置问题，重启电脑，检查ws2\_32库是否存在

cout << "重启电脑，或者检查网络库" << endl; break;

case WSAVERNOTSUPPORTED://版本库不支持

cout << "请更新网络库" << endl; break;

case WSAEINPROGRESS://运行期间出现阻塞

cout << "请重新启动" << endl; break;

case WSAEPROCLIM://Windows Socket限制了应用程序数量

cout << "请关掉一些不必要的软件" << endl; break;

}

return 0;

}

if (2 != HIBYTE(wdSockMsg.wVersion) || 2 != LOBYTE(wdSockMsg.wVersion))

{

cout << "版本出错" << endl;

//版本不对，关闭网络库

WSACleanup();

return 0;

}

SOCKET socketServer = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);//创建服务器socket

if (INVALID\_SOCKET == socketServer)

{

int a = WSAGetLastError();

cout << "创建服务器socket出错，错误码:" << a << endl;

//清理网络库

WSACleanup();

return 0;

}

//设置端口及地址

si.sin\_family = AF\_INET;

si.sin\_port = htons(12345);

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

//绑定

if (SOCKET\_ERROR == bind(socketServer, (const struct sockaddr \*)&si, sizeof(si)))

{

int a = WSAGetLastError();

cout << "绑定出错，错误码:" << a<<endl;

closesocket(socketServer);

WSACleanup();

return 0;

}

//监听

if (SOCKET\_ERROR == listen(socketServer, SOMAXCONN))

{

int a = WSAGetLastError();

cout << "监听出错，错误码:" << a << endl;

closesocket(socketServer);

WSACleanup();

return 0;

}

WSAEVENT eventServer = WSACreateEvent();//创建事件

if (WSA\_INVALID\_EVENT == eventServer)

{

int a = WSAGetLastError();

cout << "创建服务器事件出错，错误码:" << a << endl;

closesocket(socketServer);

WSACleanup();

return 0;

}

//服务器先连接上

if (SOCKET\_ERROR == WSAEventSelect(socketServer, eventServer, FD\_ACCEPT))

{

int a = WSAGetLastError();

cout << "服务器socket加入出错，错误码:" << a << endl;

WSACloseEvent(eventServer);//释放句柄

closesocket(socketServer);

WSACleanup();

return 0;

}

//服务器加入事件、socket列表

esSet.eventall[esSet.count] = eventServer;

esSet.sockall[esSet.count] = socketServer;

esSet.count++;

ifstream fs("playerPok.txt", ifstream::in);

while (!fs.eof())

readPlayer(fs);//读取文件中玩家信息

fs.close();

while (1)

{

for (int i = 0; i < esSet.count; i++)

{

//等待事件

DWORD nRes = WSAWaitForMultipleEvents(1, &esSet.eventall[i], FALSE, 0, FALSE);

if (WSA\_WAIT\_FAILED == nRes)

{

int a = WSAGetLastError();

printf("等待事件出错，错误码%d\n", a);

continue;

}

if (WSA\_WAIT\_TIMEOUT == nRes) continue;

//获取事件

WSANETWORKEVENTS NetworkEvents;

if (SOCKET\_ERROR == WSAEnumNetworkEvents(esSet.sockall[i], esSet.eventall[i], &NetworkEvents))

{

int a = WSAGetLastError();

printf("得到事件出错，错误码%d\n", a);

break;

}

else

{

//ACCEPT事件

if (NetworkEvents.lNetworkEvents&FD\_ACCEPT)

{

if (0 == NetworkEvents.iErrorCode[FD\_ACCEPT\_BIT])

{

acceptEvent(i);

}

else continue;

}

//READ事件

if (NetworkEvents.lNetworkEvents&FD\_READ)

{

if (0 == NetworkEvents.iErrorCode[FD\_READ\_BIT])

{

if (!readEvent(i)) continue;

}

else

{

printf("socket fail,error code:%d\n", NetworkEvents.iErrorCode[FD\_READ\_BIT]);

continue;

}

}

//CLOSE事件

if (NetworkEvents.lNetworkEvents&FD\_CLOSE)

{

closeEvent(NetworkEvents, i);

}

}

}

}

WSACloseEvent(eventServer);//释放句柄

closesocket(socketServer);

WSACleanup();

}

客户端连接：

SOCKET player::connectServer()

{

SetConsoleCtrlHandler(fun, TRUE);//点×退出

WORD wdVersion = MAKEWORD(2, 2);

WSADATA wdSockMsg;

int nRes = WSAStartup(wdVersion, &wdSockMsg);

if (0 != nRes)//检查网络库是否顺利打开

{

switch (nRes)

{

case WSASYSNOTREADY:

cout << "重启电脑或检查网络库" << endl; break;

case WSAVERNOTSUPPORTED:

cout << "请更新网络库" << endl; break;

case WSAEINPROGRESS:

cout << "请重新启动" << endl; break;

case WSAEPROCLIM:

cout << "请关掉一些不必要的软件"; break;

}

}

if (2 != HIBYTE(wdSockMsg.wVersion) || 2 != LOBYTE(wdSockMsg.wVersion))//检查网络库版本号

{

cout << "网络库版本出错" << endl;

WSACleanup();

return 0;

}

SOCKET socketServer = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (INVALID\_SOCKET == socketServer)//检查socket是否成功创建

{

int a = WSAGetLastError();

WSACleanup();

return 0;

}

struct sockaddr\_in serverMsg;//绑定服务器socket的地址端口

inet\_pton(AF\_INET, "127.0.0.1", &serverMsg.sin\_addr.S\_un.S\_addr);

serverMsg.sin\_family = AF\_INET;

serverMsg.sin\_port = htons(12345);

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

if (SOCKET\_ERROR == connect(socketServer, (const struct sockaddr\*)&serverMsg, sizeof(serverMsg)))//检查是否绑定成功

{

int a = WSAGetLastError();

cout << "连接错误，错误码：" << a << endl;

closesocket(socketServer);

WSACleanup();

return false;

}

return socketServer;

}

* **注册及登录**

数据包第一个值为用户的选择：

#define CHOICE\_LOGIN 0//登录

#define CHOICE\_REGIST 1//注册

#define CHOICE\_CHECK 2//查看所有用户的信息

#define CHOICE\_LOGOUT 3//登出

#define CHOICE\_UPDATE 4//更新玩家战斗后的状态

客户端登录或注册函数：

bool player::enter(SOCKET socketServer,int choice,string Password) //

{ stringstream ss;

ss << choice << " " << name<<" "<<Password<<"\0";

char sendBuf[1500] = { '\0' };

memcpy(sendBuf, ss.str().c\_str(), 1500);

if (SOCKET\_ERROR == send(socketServer,sendBuf, name.length() +Password.length()+ 3, 0))//发送登录或注册的选择+用户名+密码

{

int a = WSAGetLastError();

cout << "发送出错，错误码：" << a<<endl;

return false;

}

char recvBuf[1500] = { 0 };

int res = recv(socketServer, recvBuf, 1499, 0);

if (0 == res)

{

cout << "接收中断，服务器下线" << endl;

return false;

}

else if (SOCKET\_ERROR == res)

{

int a = WSAGetLastError();

cout << "接收出错，错误码" <<a<< endl;

return false;

}

else

{

string s = recvBuf;

stringstream ss(s);

password = Password;

//获取玩家持有的精灵信息

ss >> pokNum;

for (int i = 1; i <= 6; i++) ss >> medal[i];

ss >> winnum >> totalnum;

if (pokNum == -1)

{

cout << "失败" << endl;

return false;

}

for (int i = 1; i <= pokNum; i++)

{

int pokSeq, lv;

float exp;

ss >> pokSeq >> lv >> exp;//依次读取精灵编号、等级、经验值

Pokemon\* pok = nullptr;

pok = pok->newPok(pokSeq);//创建精灵

pok->initPok(lv, exp);//将精灵初始化为玩家之前升级后的状态

poks.push\_back(pok);//加入玩家持有精灵的vector中

}

return true;

}

}

服务器注册函数：

bool player::regist(string Name,string Password,player& p) //注册成功返回true

{

for (vector<player>::iterator it = Players.begin(); it != Players.end(); it++)

{

if (Name == it->name) return false;//重名

}

p.password = Password;

p.assignPok();//随机分配三个精灵

//将新注册的玩家信息写入文件

ofstream fs("playerPok.txt", ofstream::app);

p.writePlayer(fs);

fs.close();

//Players为储存所有玩家的容器，末尾会存在初始化时加入的空玩家的信息，将其后移。

for (vector<player>::iterator it = Players.begin(); it != Players.end(); it++)

{

if (it->getName() == "")

{