1. **服务端**
2. **精灵类方法**
3. 计算精灵属性成员函数

* 函数功能介绍：

计算精灵属性，根据不同类型的精灵，主属性增加幅度较其他属性大。

* 函数参数说明

void型

* 函数返回值说明

void型

* 函数算法描述

void BaseSprits::attributeup()//根据类型计算精灵属性

{

switch 类型

{

case Defender://防御属性增幅较大

{

atk = baseatk[0] \* pow(1.1, lv - 1);

def = basedef[0] \* pow(1.2, lv - 1);

attack\_time = base\_atk\_t[0] - (lv / 3)\*0.1;

hp = basehp[0] \* pow(1.1, lv - 1);

break;

}

case Strength://攻击属性增幅较大

{

atk = baseatk[1] \* pow(1.15, lv - 1);

def = basedef[1] \* pow(1.1, lv - 1);

attack\_time = base\_atk\_t[1] - (lv / 3)\*0.1;

hp = basehp[1] \* pow(1.1, lv - 1);

break;

}

case Agility://攻击间隔属性减少较大

{

atk = baseatk[2] \* pow(1.1, lv - 1);

def = basedef[2] \* pow(1.1, lv - 1);

attack\_time = base\_atk\_t[2] - (lv / 3)\*0.2;

hp = basehp[2] \* pow(1.1, lv - 1);

break;

}

case Shield://生命属性增幅较大

{

atk = baseatk[3] \* pow(1.1, lv - 1);

def = basedef[3] \* pow(1.1, lv - 1);

attack\_time = base\_atk\_t[3] - (lv / 3)\*0.1;

hp = basehp[3] \* pow(1.2, lv - 1);

break;

}

}

}

1. 升级成员函数

* **设计模式说明：**此处体现了类适配器模式，在派生类的levelup方法调用了基类的levelup方法和属性计算函数，使无法实例化的基类的虚成员函数接口得以使用。

基类：

* 函数功能介绍：

宠物精灵升级函数，获得经验后，判断所得经验是否足够升级，如果达到满级，则将溢出经验清0。

* 函数参数说明：

int型，精灵所得经验

* 函数返回值说明：

bool型，如果升级则返回true，否则返回false

* 函数算法描述：

bool BaseSprits::levelup(int get\_exp)

{

prelv = lv;//保存当前等级

if 已达到满级

return false;//如果满级则直接退出

else

{

exp += get\_exp;

while 经验大于当前升级所需的经验

{

exp = exp - lvupexp[lv-1];

lv++;

if 达到满级 break;

}

if 达到满级 exp = 0

if 升级

return true;

else

return false;

}

}

派生类：

bool DefSprits::levelup(int get\_exp)

{

if (BaseSprits::levelup(get\_exp))//如果升级

{

BaseSprits::attributeup();//属性提升

return true;

}

else

{

return false;

}

}

1. 精灵被攻击成员函数

* 函数功能介绍：

当精灵受到攻击时，进行剩余生命值得计算，并判断是否阵亡。该函数在对方攻击函数调用之后调用。

* 函数参数说明：

double型，受到的伤害值

* 函数返回值说明：

bool型，当精灵阵亡时，返回false，否则返回true

* 函数算法描述：

bool BaseSprits::isalive(double dps)

{

hp = hp - dps;

if (hp > 0)

{

return true;

}

else

{

return false;

}

}

1. 攻击成员函数：

* **设计模式说明：**采用模板方法模式，将公共的基础伤害计算放在基类（公共类），将需要改变的部分，即属性克制，技能伤害加成等计算放在派生类实现。

基类：

* 函数功能介绍：

计算伤害值，基础伤害等于攻击值减0.5\*防御值。

* 函数参数说明：

BaseSprits类指针，指向敌方精灵，const char\*型字符指针，用于虚函数的改写。

* 函数返回值说明：

double型，返回计算的伤害值，当伤害计算结果小于0，则输出0，否则按计算结果输出。

* 函数算法描述：

double BaseSprits::attack(BaseSprits \*oppo, const char\*)

{

myatk = BaseSprits::get\_atk();//取得自己的攻击力

harm = myatk - oppo->get\_def()\*0.5; //基础伤害等于攻击-防御\*0.5

if (harm >= 0)

return harm;

else

{

return 0;

}

}

派生类：

* 函数功能介绍：

改写基类的攻击函数，根据不同的精灵类型，不同的招式，伤害值的加成各不同。

* 函数参数说明：

BaseSprits类指针，指向敌方精灵，const char\*型字符指针，获得精灵本次攻击的技能。

* 函数返回值说明：

double型，返回计算的伤害值。

* 函数算法描述：

double DefSprits::attack(BaseSprits \*oppo, const char\* mypose)

{

cout << get\_name() << "发动" << mypose << "攻击\t\t";

eharm = 0;

if (!strcmp(DefPose[1], mypose))//不同技能有不同的额外伤害

eharm = 5;

else if (!strcmp(DefPose[2], mypose))

eharm = 10;

else if (!strcmp(DefPose[4], mypose))

eharm = 6;

else if (!strcmp(DefPose[5], mypose))

eharm = 12;

else if (!strcmp(DefPose[7], mypose))

eharm = 4;

else if (!strcmp(DefPose[8], mypose))

eharm = 10;

double rait = 0;//普通攻击，特殊攻击，大招的伤害加成不同。

if (!strcmp(mypose, pose[0]))//出招不同，伤害不同

{

rait = 1;

}

else if (!strcmp(mypose, pose[1]))

{

rait = 1.1;

}

else

{

rait = 1.2;

}

if (oppo->get\_kind() == Defender)

{

return rait\*(eharm+BaseSprits::attack(oppo,mypose));

}

else if(oppo->get\_kind()==Strength || oppo->get\_kind() == Agility)

{

return rait\*1.2\*(eharm+BaseSprits::attack(oppo,mypose));

}

else

{

return rait\*0.8\*(eharm+BaseSprits::attack(oppo,mypose));

}

}

1. **用户类方法**

1）登陆注册成员函数

* 函数功能介绍：

判断是否登陆成功

* 函数参数说明：

SOCKET\* Client，与客户端通信的嵌套字；char\* client\_require，服务端接收到的数据；int &fuser，用于判断此次接受的是用户名还是密码；int &signup，用于判断是登陆还是注册。

* 函数返回值说明：  
  bool型，当用户登陆成功是，返回true，否则返回false
* 函数算法描述：

bool User::isSignupOrLogin(SOCKET \*Client, char \*client\_require, int &fuser, int &signup)

{

recdata = new char[MAX\_BUF];//缓冲区

if 在线

return true;//如果用户在线则直接返回true

if 接收到用户选择注册的命令

{

signup = 1;

}

else if 注册

{

if 接受到注册用户名

{

strcpy(username, client\_require);

fuser = 1;

}

else if 接受到密码

{

if 未接收到注册命令

strcpy(password, client\_require);

if 接受到注册命令

{

if 用户名存在

{

反馈给客户端

fuser = 0;//重新接受用户名

}

else

{

更新用户数据表；

将注册成功信息反馈给客户端；

is\_newuser = 1;

fuser = 0;

signup = 0;

}

}

}

else

{

输入非法反馈给客户端；

fuser = 0;

}

}

else//登陆

{

if 接受用户名

{

ZeroMemory(username, 20);

strcpy(username, client\_require);

fuser = 1;

}

else

{

If 接受密码并未接受到登陆命令

{

ZeroMemory(password, 20);

strcpy(password, client\_require);

}

if 接收到登陆命令

{

if 用户不存在

{

反馈用户名或密码错误给客户端

}

else

{

If 玩家已在线

{

反馈用户已在线给客户端

return true;

}

读出该用户基本信息和精灵信息

反馈登陆成功给客户端

if (n\_pet == 0)//如果当前用户的精灵数为0，则为新用户

is\_newuser = 1;

else

is\_newuser = 0;

if (is\_newuser)//是新用户

{

将是新用户信息反馈给客户端

isonline = 1;

更新在线标志，防止其他用户登陆；

随机分配3只精灵；

}

else

{

将不是新用户反馈给客户端

isonline = 1;

更新在线标志，防止其他用户登陆；

}

return true;

}

}

}

}

return false;

}

* 1. 初始化用户成员函数
* 函数功能介绍：

从数据库读出当前用户的信息和用户的精灵，初始化相应的User对象

* 函数参数说明：

void型

* 函数返回值说明：

void型

* 函数算法描述：

void User::initUser()//读出该用户基本信息和精灵信息

{

char\* command = new char[MAX\_BUF];

sprintf(command, "select \* from myusers where username='%s'",username);

WaitForSingleObject(hmutex, INFINITE);//添加互斥事件

if (mysql\_real\_query(mysql, command, (unsigned long)strlen(command)))//查询用户数据表

{

printf("Select error:%d, %s\n", mysql\_errno(mysql), mysql\_error(mysql));

return;

}

MYSQL\_RES \*res = mysql\_store\_result(mysql);//保存查询结果

MYSQL\_ROW row;

while (row = mysql\_fetch\_row(res))

{

存储用户信息；

}

mysql\_free\_result(res);

ZeroMemory(command, MAX\_BUF);

sprintf(command, "select \* from sprits\_table where username='%s'", username);

if (mysql\_real\_query(mysql, command, (unsigned long)strlen(command)))//查询精灵列表

{

printf("Select error:%d, %s\n", mysql\_errno(mysql), mysql\_error(mysql));

return;

}

delete[] command;

res = mysql\_store\_result(mysql);

while (row = mysql\_fetch\_row(res))

{

添加精灵到用户的精灵链表数组；

}

mysql\_free\_result(res);//释放查询结果

ReleaseMutex(hmutex);//释放互斥事件

}

* 1. 添加精灵成员函数
* 设计模式说明：此函数体现了工厂模式，根据精灵的名称选择具体要实例化的精灵类。
* 函数功能介绍：

根据精灵的经验、精灵名称下标实例化一只精灵，并将精灵添加到该用户对象的精灵链表数组中。

* 函数参数说明：

int exp，精灵的总经验值，int i，精灵名称下标

* 函数返回值说明：

void 型

* 函数算法描述：

void User::AddSprits(int exp,int i)//添加精灵

{

Userspritsunit\* p = new Userspritsunit;

switch (i)

{

case 0:

case 1:

case 2:

{

DefSprits \*sprit = new DefSprits(AllSprits[i]);

sprit->levelup(exp);

p->usersprits = sprit;

break;

}

case 3:

case 4:

case 5:

{

StrSprits \*sprit = new StrSprits(AllSprits[i]);

sprit->levelup(exp);

p->usersprits = sprit;

break;

}

case 6:

case 7:

case 8:

{

AgiSprits \*sprit = new AgiSprits(AllSprits[i]);

sprit->levelup(exp);

p->usersprits = sprit;

break;

}

case 9:

case 10:

case 11:

{

ShiSprits \*sprit = new ShiSprits(AllSprits[i]);

sprit->levelup(exp);

p->usersprits = sprit;

break;

}

default:

break;

}

if 链表首节点为空

{

mysprits[i] = p;

p->next = NULL;

p->sprit\_count = 1;

tail[i] = mysprits[i];

}

else

{

tail[i]->next = p;

p->sprit\_count = tail[i]->sprit\_count + 1;//sprit\_count用于区分同种精灵

p->next = NULL;

tail[i] = p;

}

if(isonline)

将精灵插入数据库精灵表中

}

* 1. 随机分配三只精灵成员函数
* 函数功能介绍：

为新用户随机分配三只精灵

* 函数参数说明：

Void型

* 函数返回值说明：

Void型

* 函数算法描述：

void User::RandDistributeSprits()//新用户随机获得三只精灵

{

j = 0;

srand((unsigned int)time(NULL));

for (i = 0; i < 3; i++)

{

j = rand() % 12;

添加精灵到用户对象的精灵链表数组

精灵数量加1

}

* 1. 显示用户信息成员函数
* 函数功能介绍：

将用户的信息发送给客户端

* 函数参数说明：

SOCKET\*client，服务器与客户端交互数据的嵌套字

* 函数返回值说明：

Void型

* 函数算法描述：

void User::ShowUserMessage(SOCKET\* client)

{

char\* senddata = new char[MAX\_BUF];

sprintf(senddata, "%s", username);

send(\*client, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%d",n\_pet);

send(\*client, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%d", highlv\_pet);

send(\*client, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%d", winrate);

send(\*client, senddata, MAX\_BUF, 0);

delete[]senddata;

}

* 1. 显示其他用户信息成员函数
* 函数功能介绍：

将其他用户的基本信息发送到客户端

* 函数参数说明：

SOCKET\*client，服务器与客户端交互数据的嵌套字

* 函数返回值说明：

Void型

* 函数算法描述：

void User::ShowOtherUsers(SOCKET\* client)

{

char \*command = new char[MAX\_BUF];

sprintf(command, "select \* from myusers");

WaitForSingleObject(hmutex, INFINITE);

if (mysql\_real\_query(mysql, command, (unsigned long)strlen(command)))

{

printf("Select error:%d, %s\n", mysql\_errno(mysql), mysql\_error(mysql));

return;

}

delete[]command;

char\* senddata = new char[MAX\_BUF];

MYSQL\_RES \*res = mysql\_store\_result(mysql);//读取所有查询结果

MYSQL\_ROW row;

while (row = mysql\_fetch\_row(res))

{

if (!strcmp(row[0], username))

continue;

else

{

发送其他用户给客户端

}

}

sprintf(senddata, "All Done");//提示客户端发送完成

send(\*client, senddata, MAX\_BUF, 0);

delete[]senddata;

mysql\_free\_result(res);

ReleaseMutex(hmutex);

}

* 1. 显示用户精灵成员函数
* 函数功能介绍：

将某一个用户的精灵发送给客户端：

* 函数参数说明：

SOCKET\*client，服务器与客户端交互数据的嵌套字

char \*otherusername 该用户ID

* 函数返回值说明：

Void型

* 函数算法描述：

void User::ShowOtherUser\_Sprits(SOCKET\* client, char \*otherusername)

{

char\* command = new char[MAX\_BUF];

sprintf(command, "select \* from sprits\_table where username='%s'", otherusername);

WaitForSingleObject(hmutex, INFINITE);

if (mysql\_real\_query(mysql, command, (unsigned long)strlen(command)))

{

printf("Select error:%d, %s\n", mysql\_errno(mysql), mysql\_error(mysql));

}

delete[] command;

char\* senddata = new char[MAX\_BUF];

MYSQL\_RES \*res = mysql\_store\_result(mysql);//读取所有查询结果

MYSQL\_ROW row;

while (row = mysql\_fetch\_row(res))

{

发送该用户精灵到客户端；

}

sprintf(senddata, "%s", "All Done");//提示发送结束

send(\*client, senddata, MAX\_BUF, 0);

delete[]senddata;

mysql\_free\_result(res);

ReleaseMutex(hmutex);

}

* 1. 插入精灵至数据库成员函数
* 函数功能介绍：

将新精灵更新至数据库

* 函数参数说明：

Userspritsunit \*addsprit，将要插入至数据库的精灵节点

* 函数返回值说明：

Void型

* 函数算法描述;

void User::AddSpritsTOLib(Userspritsunit \*addsprit)//增添精灵记录到数据库

{

char\* command = new char[MAX\_BUF];

j=GetSpritsToNum(addsprit->usersprits->get\_name().c\_str());

sprintf(command, "insert into sprits\_table values(%d,'%s',%d,%d,'%s')",

j/3,

addsprit->usersprits->get\_name().c\_str(),

addsprit->usersprits->get\_lv(),

addsprit->usersprits->get\_exp(),

username

);

WaitForSingleObject(hmutex, INFINITE);

if (mysql\_real\_query(mysql, command, (unsigned long)strlen(command)))

{

printf("Insert error:%d, %s\n", mysql\_errno(mysql), mysql\_error(mysql));

return;

}

delete[] command;

ReleaseMutex(hmutex);

}

* 1. 删除用户精灵成员函数
* 函数功能介绍：

将从用户精灵链表数组中删除指定精灵

* 函数参数说明：

const char\* oldsprit，指定精灵的名称

int i，指定精灵在重复精灵中的标号

* 函数返回值说明：

void型

* 函数算法描述：

void User::DeleteSprits(const char\* oldsprit,int i)//删除精灵

{

j = GetSpritsToNum(oldsprit);//获得精灵下标

Userspritsunit\* p = mysprits[j];

Userspritsunit\* last = mysprits[j];

if (mysprits[j] == tail[j])

{

if (mysprits[j]->usersprits->get\_lv() == 15)

modify\_highlvpet(false);

DelSpritFromLib(mysprits[j]);//从数据库删除

delete mysprits[j]->usersprits;

delete mysprits[j];

mysprits[j] = NULL;

return;

}

if 在链表头

{

last = p->next;

mysprits[j] = last;

}

else

{

while (--i)

{

last = p;

p = p->next;

}

last->next = p->next;

}

if删除的是高级精灵

modify\_highlvpet(false);

DelSpritFromLib(p);

delete p->usersprits;

delete p;

for (last = last->next; last; last = last->next)

{

int pre = last->sprit\_count;

last->sprit\_count--;

}

}

* 1. 重新分配初始化精灵成员函数

此处也体现了工厂模式。

* 函数功能介绍：

当用户精灵数为0时，系统随机分配一个初始精灵

* 函数参数说明：

int i，随机产生的精灵名称数组下标

* 函数返回值说明：

void型

* 函数算法描述：

void User::reallocate(int i)//重新分配一只初始精灵

{

mysprits[i] = new Userspritsunit;

BaseSprits\* newsprit = NULL;

j = i / 3;

switch(j)

{

case 0:

{

DefSprits \*sprit = new DefSprits(AllSprits[i]);

newsprit = sprit;

break;

}

case 1:

{

StrSprits \*sprit = new StrSprits(AllSprits[i]);

newsprit = sprit;

break;

}

case 2:

{

AgiSprits \*sprit = new AgiSprits(AllSprits[i]);

newsprit = sprit;

break;

}

case 3:

{

ShiSprits \*sprit = new ShiSprits(AllSprits[i]);

newsprit = sprit;

break;

}

default:

break;

}

mysprits[i]->usersprits = newsprit;

mysprits[i]->sprit\_count=1;

mysprits[i]->next = NULL;

AddSpritsTOLib(mysprits[i]);

}

* 1. 战斗结算成员函数
* 函数功能介绍：

接收用户发来的对战精灵信息、对战结果信息，并将战斗结算发送给客户端。

* 函数参数说明：

int n，玩家精灵在同种精灵链表的位置序号

int j，玩家精灵名称数组下标

char\* name，玩家精灵名称，

SOCKET\* client，服务端与客户端进行通信的嵌套字

int flag，判断是升级赛还是决斗赛

* 函数返回值说明：

void型

* 函数算法描述：

void User::AssignToFight(int n, int j,char\* name, SOCKET\* client,int flag)

{

char\* senddata = new char[MAX\_BUF];

Userspritsunit \*p=mysprits[j];

遍历找到该用户的派出的精灵

BaseSprits\* assigned = p->usersprits;

char \*revdata = new char[MAX\_BUF];

将对战精灵信息反送给客户端；

接受战斗结果；

if赢得比赛

{

assigned->attributeup(assigned->get\_kind());

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%s获得100经验",assigned->get\_name().c\_str());

send(\*client, senddata, MAX\_BUF, 0);

if 升级

{

UpdataSprit(p);

if (15 == assigned->get\_lv())

modify\_highlvpet(true);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata,"lvup");

send(\*client, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%d",assigned->get\_lv());

send(\*client, senddata, MAX\_BUF, 0);

}

else

{

UpdataSprit(p);

}

if (!flag)//如果是决斗赛

{

modify\_winrate(true);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "Do nothing");

send(\*client, senddata, MAX\_BUF, 0);

AddSprits(name, 0, j);

modify\_npet(true);

}

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "Done");

send(\*client, senddata, MAX\_BUF, 0);

}

else

{

assigned->attributeup(assigned->get\_kind());

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%s获得50经验", assigned->get\_name().c\_str());

send(\*client, senddata, MAX\_BUF, 0);

if (assigned->levelup(50))

{

UpdataSprit(p);

if (15 == assigned->get\_lv())

modify\_highlvpet(true);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "lvup");

send(\*client, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "%d", assigned->get\_lv());

send(\*client, senddata, MAX\_BUF, 0);

}

else

{

UpdataSprit(p);

}

if (!flag)

{

modify\_winrate(false);

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "Do nothing");

send(\*client, senddata, MAX\_BUF, 0);

}

ZeroMemory(senddata, MAX\_BUF);

sprintf(senddata, "Done");

send(\*client, senddata, MAX\_BUF, 0);

if(!flag)

RandSelect\_3Sprits(client);//随机选择三只或者用户全部精灵（少于等于三只时），发送给用户

}

}

* 1. 战斗失败结算成员函数
* 函数功能介绍：

如果用户精灵数大于三只，则随机选择三只精灵发送给客户端，否则将用户所有精灵发送给客户端。

* 函数参数说明：

SOCKET\* client，服务端与客户端进行通信的嵌套字

* 函数返回值说明：

void型

* 函数算法描述：

void User::RandSelect\_3Sprits(SOCKET\* client)

{

bool\* flag = new bool[n\_pet];//精灵被选中时对应数组元素为true

for (i = 0; i < n\_pet; i++)

flag[i] = false;

char \* buf = new char[MAX\_BUF];

srand((unsigned int)time(NULL));//设置随机数

int count = 3;

if (n\_pet <= 3)//精灵数目小于等于3，选择用户所有精灵

{

for (int i = 0; i < spritsnum; i++)

{

for (Userspritsunit\* p = mysprits[i]; p; p=p->next)

{

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "%s", p->usersprits->get\_name().c\_str());

send(\*client, buf, MAX\_BUF, 0);

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "%d", p->usersprits->get\_lv());

send(\*client, buf, MAX\_BUF, 0);

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "%d", p->sprit\_count);

send(\*client, buf, MAX\_BUF, 0);

count--;

}

}

while (count--)//发送空白给用户

{

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "blank");

send(\*client, buf, MAX\_BUF, 0);

}

}

else

{

for (int k = 0; k < 3; k++)

{

int j = (rand() % n\_pet) + 1;

while (flag[j - 1])//确保选择的精灵不会再次被选

{

Sleep(100);

j = (rand() % n\_pet) + 1;

}

flag[j - 1] = true;

int i = 0;

for (; i < spritsnum; i++)

{

if (tail[i])

j = j - tail[i]->sprit\_count;

if (j <= 0)

break;

}

Userspritsunit\* p = mysprits[i];

j = j + tail[i]->sprit\_count;

for (; j - 1; j--)

p = p->next;

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "%s", p->usersprits->get\_name().c\_str());

send(\*client, buf, MAX\_BUF, 0);

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "%d", p->usersprits->get\_lv());

send(\*client, buf, MAX\_BUF, 0);

ZeroMemory(buf, MAX\_BUF);

sprintf(buf, "%d", p->sprit\_count);

send(\*client, buf, MAX\_BUF, 0);

}

}

}

1. **多线程**
   * 1. 等待客户端连接线程函数

* 函数功能介绍：

循环等待客户端连接

* 函数参数说明：

LPVOID pm，传递给线程函数处理的参数

* 函数返回值说明：

void型

* 函数算法描述：

void connect\_server(LPVOID pm)

{

server\* My\_server = (server\*)pm;

n = 0;

cout << endl << endl << "连接状况:" << endl;

while (true)

{

My\_server->users[n].sClient = new SOCKET;

sockaddr\_in remoteAddr;

int nAddrlen = sizeof(remoteAddr);

printf("等待连接...\n");

while (true)

{

\*My\_server->users[n].sClient = accept(My\_server->sserver, (SOCKADDR \*)&remoteAddr, &nAddrlen);//从等待队列中选择一个创建连接，若无则阻塞

if (INVALID\_SOCKET == \*My\_server->users[n].sClient)

{

int err = WSAGetLastError();

if (err == WSAEWOULDBLOCK)

{

Sleep(100);

continue;

}

else

{

printf("accept failed!\n");

closesocket(My\_server->sserver);

WSACleanup();

return;

}

}

break;

}

printf("接受到一个连接：%s \r\n", inet\_ntoa(remoteAddr.sin\_addr));

HANDLE Hthread = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)revdata, (LPVOID)&My\_server->users[n], 0, NULL);//为每个用户连接创建一个线程，应用计数为2

if (Hthread)

{

CloseHandle(Hthread);//避免内核对象泄露，引用计数减1

}

Sleep(1000);

n++;

}

}

* + 1. 与客户端数据交互线程线程函数
* 函数功能介绍：

与客户端进行数据交互，每个客户端对应一个线程

* 函数参数说明：

LPVOID pm，传递给线程函数处理的参数

* 函数返回值说明：

void型

* 函数算法描述：

void revdata(LPVOID pm)

{

User user;

user\_socket\* player = (user\_socket\*)pm;

WaitForSingleObject(hMutex, INFINITE);

player->useraddr = &user;

user\_count++;

ReleaseMutex(hMutex);

int retVal;

char \*revData = new char[MAX\_BUF];

int fuser = 0;

int signup = 0;

while (true) //循环接收数据

{

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

if (SOCKET\_ERROR == retVal)

{

int err = WSAGetLastError();

if (err == WSAEWOULDBLOCK)

{

Sleep(100);

continue;

}

else if (err == WSAETIMEDOUT || err == WSAENETDOWN)

{

printf("recv failed!\n");

closesocket(\*player->sClient);

WSACleanup();

return ;

}

}

else if (retVal > 0)

{

if (!strcmp(revData, "Quit"))//用户退出客户端，线程释放

{

user.modify\_isonline(0);

user.UpdataUser\_isonline();//将数据库中用户在线标志位改为0

break;

}

if (!user.isSignupOrLogin(player->sClient, revData, fuser, signup))//用户登录客户端

{

continue;

}

else

{

if (!strcmp(revData, "Show MySprits"))//查看用户精灵

user.ShowUserSprits(player->sClient);

else if (!strcmp(revData, "Show My Message"))//查看用户基本信息

user.ShowUserMessage(player->sClient);

else if (!strcmp(revData, "Show other users"))//查看其他用户基本信息

{

user.ShowOtherUsers(player->sClient);

}

else if(!strcmp(revData,"Show someone's sprits"))//查看其他用户的精灵

{

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

user.ShowOtherUser\_Sprits(player->sClient,revData);

}

else if (!strcmp(revData, "Show virtual sprits"))//将虚拟精灵表发给用户

{

SendVirtualSpritsToClient(player->sClient);

}

else if (!strcmp(revData, "Train")|| !strcmp(revData, "Fight"))//用户选择升级赛或者决斗赛

{

int flag = revData[0] - 'F';//判断标志位 =0表示决斗赛

int sprit\_count = 0;

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

if (!strcmp(revData, "Exit"))

continue;

sprit\_count = CharToInt(revData);

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

int j = GetSpritsToNum(revData);

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

user.AssignToFight(sprit\_count,j, revData, player->sClient,flag);//用户选择精灵对战

}

else if (!strcmp(revData, "Lose a sprit"))//用户决斗赛失败，失去一只精灵

{

int samesprit = 0;

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

samesprit = CharToInt(revData);

ZeroMemory(revData, MAX\_BUF);

retVal = recv(\*player->sClient, revData, MAX\_BUF, 0);

revData[retVal] = '\0';

user.DeleteSprits(revData, samesprit);

user.modify\_npet(false);

}

else

{

continue;

}

}

}

}

closesocket(\*player->sClient);//线程结束前将socket释放

return;

}

1. **客户端**

PlayerSprits类：

* **设计模式说明：**该类采用了对象适配器，组合模式，能够在成员函数中调用了精灵派生类的方法。
  1. 初始化精灵链表成员函数
* 函数功能介绍：

将服务端发送过来的精灵信息初始化该精灵链表

* 函数参数说明：

SOCKET &client，客户端与服务端数据交互的嵌套字

* 函数返回值说明：

Void型

* 函数算法描述：

void PlayerSprits::initSpritsList(SOCKET &client)//初始化精灵链表

{

playsprit\* newsprit=first;

char \*revdata = new char[MAX\_BUF];

int retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

while未接收到结束提示

{

switch接收到的精灵类型

{

case 0://defender

{

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

DefSprits \*sprit = new DefSprits(revdata);

newsprit->usersprit = sprit;

break;

}

case 1://strength

{

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

StrSprits \*sprit = new StrSprits(revdata);

newsprit->usersprit = sprit;

break;

}

case 2://agility

{

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

AgiSprits \*sprit = new AgiSprits(revdata);

newsprit->usersprit = sprit;

break;

}

case 3://shield

{

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

ShiSprits \*sprit = new ShiSprits(revdata);

newsprit->usersprit = sprit;

break;

}

default:

{

break;

}

}

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

lv = CharToInt(revdata);

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

int exp\_sum = CharToInt(revdata);

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

exp\_sum = exp\_sum + lvupexp[i-1];

newsprit->usersprit->levelup(exp\_sum);

spritscount++;//精灵总数

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

newsprit->csamesprit = CharToInt(revdata);

tail = newsprit;

newsprit->next = new playsprit;

newsprit = newsprit->next;

newsprit->next = NULL;

newsprit->usersprit = NULL;

ZeroMemory(revdata, MAX\_BUF);

retval = recv(client, revdata, MAX\_BUF, 0);

revdata[retval] = '\0';

}

if (first == newsprit)

return;

delete newsprit;

tail->next = first;

current = tail;

}

* 1. 显示精灵链表成员函数
* 函数功能介绍：

显示精灵列表。

* 函数参数说明：

CListCtrl\* list，列表控件指针

* 函数返回值说明：

Void型

* 函数算法描述：

void PlayerSprits::ShowSpritsList(CListCtrl\* list)

{

CString name, kind, lv, exp,hp, atk, def, atk\_time;

playsprit\* p = first;

char str[10];

for (int i = 0; i < spritscount; i++,p=p->next)

{

name = p->usersprit->get\_name().c\_str();

list->InsertItem(i, name);

kind = classes[p->usersprit->get\_kind()];

list->SetItemText(i, 1, kind);

ZeroMemory(str, 10);

sprintf\_s(str, 10, "%d", p->usersprit->get\_lv());

lv = str;

list->SetItemText(i, 2, lv);

ZeroMemory(str, 10);

sprintf\_s(str, 10, "%d", p->usersprit->get\_exp());

exp = str;

list->SetItemText(i, 3, exp);

ZeroMemory(str, 10);

sprintf\_s(str, 10, "%d", (int)p->usersprit->get\_hp());

hp = str;

list->SetItemText(i, 4, hp);

ZeroMemory(str, 10);

sprintf\_s(str, 10, "%d", (int)p->usersprit->get\_atk());

atk = str;

list->SetItemText(i, 5, atk);

ZeroMemory(str, 10);

sprintf\_s(str, 10, "%d", (int)p->usersprit->get\_def());

def = str;

list->SetItemText(i, 6, def);

ZeroMemory(str, 10);

sprintf\_s(str, 10, "%.1lf", p->usersprit->get\_attack\_time());

atk\_time = str;

list->SetItemText(i, 7, atk\_time);

}

}

* 1. 显示精灵技能列表成员函数
* 函数功能介绍：

显示精灵技能列表。

* 函数参数说明：

CListCtrl\* list，技能列表变量指针

char\* name，精灵的名称

int lv，精灵的等级

* 函数返回值说明：

Void型

* 函数算法描述：
  1. 显示当前精灵位图成员函数
* 函数功能介绍：

显示精灵位图

* 函数参数说明：

CStatic\* pwnd，picture控件句柄指针

* 函数返回值说明：

Void型

* 函数算法描述：

void PlayerSprits::ShowCurrentSpritImage(CStatic\* pwnd)

{

CBitmap pic\_sprit;

CString path;

char p[30];

sprintf\_s(p, 30, "res/%s.bmp", current->usersprit->get\_name().data());

path = p;

HBITMAP pic = (HBITMAP)::LoadImage(NULL, path, IMAGE\_BITMAP, 0, 0, LR\_CREATEDIBSECTION | LR\_DEFAULTSIZE | LR\_LOADFROMFILE);

pwnd->SetBitmap(pic);

DeleteObject(pic);

}

CMysprits类：

1. 显示选中精灵的技能表函数

* 函数功能介绍：

玩家点击精灵列表某一行时，显示相应的精灵位图

* 函数算法描述：

void CMySprits::OnNMClickListSprits(NMHDR \*pNMHDR, LRESULT \*pResult)//显示选中精灵的招式表

{

获得选择的行数；

if点击的位置合法

{

获得被选中的精灵名称；

获得被选中的精灵等级；

flag = true;

m\_PoseList->DeleteAllItems();

显示该精灵的技能表；

}

\*pResult = 0;

重绘界面；

}

1. 排序响应函数

* 函数功能介绍：

点击精灵列表的某列时，进行排序，再点击一次进行逆序排序

* 函数算法描述：

void CMySprits::OnLvnColumnclickListSprits(NMHDR \*pNMHDR, LRESULT \*pResult)//点击某列的排序响应函数

{

获得点击的列；

if 如果当前点击的列于上一次点击的列数不相等

{

dwSelColID = pNMLV->iSubItem;//选择的列

保持排序顺序；

}

else

改变排序顺序；

if 点击列数大于3

return;

int count = m\_SpritsList->GetItemCount();

for (i = 0; i < count; i++)

m\_SpritsList->SetItemData(i, i);

m\_SpritsList->SortItems(CompareFunc, (LPARAM)(m\_SpritsList));//调用排序函数

\*pResult = 0;

}

1. 排序函数

* 函数功能介绍：

排序函数，通过字符串比较实现

* 函数算法描述：

static int CALLBACK CompareFunc(LPARAM lParam1, LPARAM lParam2, LPARAM lParamSort)//比较函数

{

CListCtrl \*pListCtrl = (CListCtrl \*)lParamSort;

LVFINDINFO findInfo;

findInfo.flags = LVFI\_PARAM;

找到表中的任意两行；

提取对应列的值；

获得对应列字符串；

if 升序

return strcmp(str1, str2);

else

return strcmp(str2, str1);

}

COtherPlayerDlg类：

1. 选中用户函数

* 函数功能介绍：

点击其他用户列表，获得相应的用户名

* 函数算法描述：

void COtherPlayerDlg::OnNMClickListOthers(NMHDR \*pNMHDR, LRESULT \*pResult)

{

获得点击的行数；

if 点击的位置合法

{

获得相应的用户名

}

\*pResult = 0;

}

1. 初始化用户列函数

* 函数功能介绍：

将服务端发来的其他用户信息初始化用户列，并显示

* 函数参数说明：

SOCKET &client，客户端与服务器的通信的嵌套字

CListCtrl\* list，其他用户列表

* 函数返回值说明：

Void型

* 函数算法描述：

void COtherPlayerDlg::initList(SOCKET &client, CListCtrl\* list)//初始化用户表

{

CString name, npet, n\_high, win;

char\* revdata = new char[MAX\_BUF];

接受数据；

int n = 0;

while (strcmp(revdata, "All Done"))

{

name = revdata;

接受数据；

npet = revdata;//精灵数量

接受数据；

n\_high = revdata;//高级精灵数量

接受数据；

win = revdata;

win = win + \_T('%');//胜率

接受数据；

isonline[n] = CharToInt(revdata);//在线信息

list->InsertItem(n, name);

list->SetItemText(n, 1, npet);

list->SetItemText(n, 2, n\_high);

list->SetItemText(n, 3, win);

n++;

player\_count++;

接受数据；

}

}

1. 查看用户精灵函数

* 函数功能介绍：

将某用户ID发送给客户端，请求该用户的精灵信息

* 函数参数说明：

SOCKET &client，客户端与服务器的通信的嵌套字

char \* name，选中的用户名

* 函数返回值说明：

Void型

* 函数算法描述：

void COtherPlayerDlg:: initCurSpritsList(SOCKET &client, char \* name)

{

char \* senddata = new char[MAX\_BUF];

sprintf\_s(senddata, strlen("Show someone's sprits") + 1, "Show someone's sprits");

send(client, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf\_s(senddata, 20, "%s", name);

send(client, senddata, MAX\_BUF, 0);

}

CfightDlg类：

* 1. 定时器相应函数：
* 函数功能介绍：

精灵对战函数，设置两个分别以玩家精灵的攻击间隔和系统精灵的攻击间隔为定时间隔的定时器，当定时器响应时，对应的精灵攻击，并设置一个间隔为1秒的定时器，用于输出战斗过程的擦除。

* 函数参数说明：

UINT\_PTR nIDEvent，响应的定时器ID

* 函数返回值说明：

void型

* 函数算法描述：

void CFightDlg::OnTimer(UINT\_PTR nIDEvent)//定时器响应函数

{

switch (nIDEvent)

{

case 1://玩家精灵攻击

{

显示精灵发动攻击及其所用技能

if (!flag3)

{

设置定时器3；

flag3 = true;

}

if系统精灵闪避

{

显示系统精灵闪避

}

else

{

harm = 0;

if暴击

{

harm = 2 \* (int)mysprit->attack(enemy, pose);

sprintf\_s(out, MAX\_BUF, "受到%d点暴击伤害", harm);

ehstr = out;

}

else

{

harm = (int)mysprit->attack(enemy, pose);

sprintf\_s(out, MAX\_BUF, "受到%d点伤害", harm);

ehstr = out;

}

If系统精灵阵亡

{

系统精灵血条清空；

KillTimer(1);

KillTimer(2);

iswin = true;

显示玩家获胜，并反馈给服务端

EndFight();

}

else

{

刷新系统精灵当前的状态；

}

}

break;

}

case 2://敌方精灵攻击

{

显示系统精灵发动攻击及其所用技能；

if (!flag4)

{

设置定时器4；

flag4 = true;

}

If用户精灵闪避

{

显示玩家精灵闪避；

}

else

{

harm = 0;

if暴击

{

harm = 2 \* (int)enemy->attack(mysprit, pose);

sprintf\_s(out, MAX\_BUF, "受到%d点暴击伤害", harm);

mhstr = out;

}

else

{

harm = (int)enemy->attack(mysprit, pose);

sprintf\_s(out, MAX\_BUF, "受到%d点伤害", harm);

mhstr = out;

}

If玩家精灵阵亡

{

玩家血条清空；

KillTimer(1);

KillTimer(2);

iswin = false;

显示玩家失败，并反馈给服务端

EndFight();

}

else

{

刷新玩家精灵当前的状态；

}

}

break;

}

case 3://玩家精灵攻击信息显示一秒擦除

{

KillTimer(3);

清空玩家精灵攻击状态框；

清空系统精灵生命状态框；

break;

}

case 4://玩家精灵生命信息显示一秒擦除

{

KillTimer(4);

清空系统精灵攻击状态框；

清空玩家精灵生命状态框；

break;

}

}

CDialogEx::OnTimer(nIDEvent);

}

* 1. 战斗结算显示函数：
* 函数功能介绍：

显示战斗结算详情

* 函数参数说明：

void型

* 函数返回值说明：

void型

* 函数算法描述：

void CFightDlg::EndFight()//战斗结算

{

从服务器接受获得的经验值；

显示获得的经验值；

if 升级

{

显示升级过程；

ZeroMemory(buf, MAX\_BUF);

revtal = recv(sclient, buf, MAX\_BUF, 0);

buf[revtal] = '\0';

str = buf;

}

if (!strcmp(buf, "Done"))//判断是否接受完毕

{

this->OnCancel();

GetParent()->ShowWindow(SW\_SHOW);

}

If 赢得决斗赛

{

显示获得的精灵；

this->OnCancel();

GetParent()->ShowWindow(SW\_SHOW);

}

else if 输掉决斗赛

{

ZeroMemory(buf, MAX\_BUF);

revtal = recv(sclient, buf, MAX\_BUF, 0);

buf[revtal] = '\0';

if (!strcmp(buf, "Done"))

{

this->OnCancel();

GetParent()->ShowWindow(SW\_SHOW);

}

}

}

ClossDlg类：

选择精灵丢弃函数：

* 函数功能介绍：

点击舍弃，对应精灵信息发送给用户，如果点击对应的框空白（即用户精灵数小于3），则直接返回。

* 函数参数说明：

void型

* 函数返回值说明：

void型

* 函数算法描述：

void CLossDlg::OnBnClickedLoss1()//选择第一只舍弃

{

if (!strcmp(spritsname[0], "blank"))

return;//空白则退出

char \* senddata = new char[MAX\_BUF];

sprintf\_s(senddata, MAX\_BUF, "Lose a sprit");

send(sclient, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf\_s(senddata, MAX\_BUF, "%d", samesprit[0]);

send(sclient, senddata, MAX\_BUF, 0);

ZeroMemory(senddata, MAX\_BUF);

sprintf\_s(senddata, MAX\_BUF, "%s", spritsname[0]);

send(sclient, senddata, MAX\_BUF, 0);

delete[] senddata;

if (!strcmp(spritsname[1], "blank") && !strcmp(spritsname[2], "blank"))

MessageBox(\_T("抱歉，您已无精灵继续战斗。\n系统为您随机分配一只精灵，请前往精灵查收"),NULL,NULL);

for (int i = 0; i < 3; i++)

delete[]spritsname[i];

this->OnCancel();

GetParent()->ShowWindow(SW\_SHOW);

// TODO: 在此添加控件通知处理程序代码

}

CPreForFIghtDlg类：

缓冲显示位图函数：

* 函数功能介绍：

采用缓冲显示位图，消除显示位图时的闪烁。

* 函数算法描述：

BOOL CPreForFIghtDlg::OnEraseBkgnd(CDC\* pDC)

{

CRect rect,srect;

CDC dcmem1, dcmem2;

GetClientRect(&srect);

CStatic\* pWnd = (CStatic\*)GetDlgItem(IDC\_MYSPRITS\_FIGHT);

pWnd->GetClientRect(&rect);

dcmem1.CreateCompatibleDC(pWnd->GetDC());

pDC->FillSolidRect(0, 0, srect.Width(), srect.Height(), RGB(255, 255, 255));

pWnd->GetDC()->FillSolidRect(rect, RGB(255, 255, 255));

player.ShowAssignedSpritImage(pWnd,pDC,srect,rect,dcmem1,flag1);//在内存中将玩家精灵位图先绘制一遍

flag1 = false;

CRect mrect;

CStatic\* tpWnd = (CStatic\*)GetDlgItem(IDC\_STATIC\_MYSPRIT);

tpWnd->GetClientRect(&mrect);

tpWnd->GetDC()->FillSolidRect(mrect, RGB(255, 255, 255));

player.ShowAssignedSpritMsg(tpWnd);

DeleteObject(tpWnd);

CRect Vrect;

CStatic\* vpWnd = (CStatic\*)GetDlgItem(IDC\_ENEMY);

vpWnd->GetClientRect(&Vrect);

dcmem2.CreateCompatibleDC(vpWnd->GetDC());

virtual\_player.ShowAssignedSpritImage(vpWnd, pDC,srect,Vrect,dcmem2,flag2); ;//在内存中将系统精灵位图先绘制一遍

flag2 = false;

DeleteObject(pWnd);

DeleteObject(vpWnd);

dcmem1.DeleteDC();

dcmem2.DeleteDC();

return FALSE;

}