**过程**

**1. 首先的问题是下棋的两端应该是什么样的？**

　　我希望下棋的两个人使用相同的程序。所以就不能像FTP一样，一个客户端，一个服务器端，而只能每个程序都是一样的，既是客户端（Client），又是服务器端（Server）。在通信时，己方的*Client*向对方的*Server*发送信息，对方的*Client*向己方的*Server*发送信息。两端都存储棋盘信息，通过通信保持棋盘信息的一致。

　　然后呢，应该是一端点击界面移子之后，应该能通知对方进行相同的移动。

　　综合以上两点，运行过程应该是这样的：

　　当在界面上点击棋子时，先判断当前是否轮到自己落子，如果是，则进行移动，更新界面，并通过*Client*向对方*Server*发送移动信息。对方*Server*收到后，进行同样的移动，更新界面。

　　这里要求Server能随时接到对方发来的消息，所以Server的监听应该是一个额外的线程。

**2. 接下来的问题是怎么表示，怎么存储？**

　　棋盘，应该用二维数组存储比较好，数组坐标（以下所说的"数组*坐标*"是指该二维数组中的一个(x,y)数对）对应棋盘坐标。那么数组里存储什么呢，一共有車、马、象、士、将、砲、卒七种棋子，那么设置一个棋子类作为基类，然后设置七个类继承棋子类？基类有一个move函数，每个子类重写该函数？但是移动似乎只是我程序的一小部分，这样似乎没必要。

　　那么存储整型数值？不同的数值代表不同的棋子？似乎可以。

　　那么就用7个数代替七种棋子，但是棋子有黑白色，要用一个数表示棋子类型（即是車、马或其他）和棋子颜色两个信息，那就用*BLANK*=8代表空子，黑方的車、马、象、士、将、砲、卒分别为1到7，白方的車、马、相、士、帅、炮、兵分别为9到15。

　　这样判断某数组坐标上棋子的颜色，就把其值与*BLANK*比较，大于BLANK为白色，否则为黑色。

　　判断某数组坐标上棋子的类型，则将其值模*BLANK*。

　　另外，因为下棋双方的视角是相反的，所以，棋盘在存储时应该是相反的，移动时的坐标也应该进行转换。

**3. 然后应该怎么通信呢？**

　　我希望这个程序打开后，就能找到对方，并确定谁是黑色，谁是白色。

　　也许可以让*Client*在运行之后就对局域网进行端口扫描，然后给出正在运行此程序的*IP*地址列表，让用户选择要连接到哪个，如果对方已经有了连接，则对方会拒绝此连接，如果对方没有连接，则对方程序会向对方用户提示该连接请求，如果，对方用户同意，则连接建立，否则依然是拒绝此连接。

　　但是，我没有采用以上所述方法（因为太复杂，我还是先做好主体工作吧=\_=）。

　　所以在程序开始运行后，会让用户输入对方的*IP*地址，然后*Server*开始监听。之后*Client*开始向对方发出连接请求。

*Server*监听时，如果收到连接请求，就看对方的*IP*地址是否是用户输入的*IP*地址，如果不是，说明连接请求不是用户所希望的对方发送的，那就继续监听。

*Client*请求连接时，如果对方同意了，就要开始确定自己的颜色了。

　　确定颜色这里困扰了我很久，最后采用的解决方法是这样的：

　　　　核心思想就是谁先发出连接请求，谁就是黑色。

　　　　也就是在*Client*连接上对方之后，要判断*Server*是不是已经连接了对方，如果*Server*已连接，就说明是对方先发出的连接请求，那么对方就是黑色，自己就设为白色。如果*Server*没有连接，就说明自己先连接上了对方，也就是自己是黑色。

　　以上就是编码前及编码时的大致想法。

**代码**

**main.cpp**

#if defined(UNICODE) && !defined(\_UNICODE)

#define \_UNICODE

#elif defined(\_UNICODE) && !defined(UNICODE)

#define UNICODE

#endif

#include <tchar.h>

#include <windows.h>

#include <pthread.h>

#include <windowsx.h>

#include "chinese\_chess.h"

#include "Server.h"

#include "Client.h"

#define WIDTH 600 //界面宽度

#define HEIGHT 600 //界面高度

#define ZERO\_X 70 //棋盘左边界

#define ZERO\_Y 70 //棋盘上边界

#define PIECE\_BKCOLOR RGB(195,163,109) //棋子背景色

#define PIECE\_WH 45 //棋盘每个格子的宽度和高度

HWND hwnd; /\* This is the handle for our window \*/

char\* ots\_ip; //存储对方IP地址的字符串

int port;

bool is\_connect\_alive=false; //是否连接到对方

Board \* chess\_board; //棋盘

Server \*server;

Client \*client;

int chess\_sx=-1; //移动起始位置的数组坐标

int chess\_sy=-1;

int chess\_dx=-1; //移动目标位置的数组坐标

int chess\_dy=-1;

/\* Declare Windows procedure \*/

LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

/\* Make the class name into a global variable \*/

TCHAR szClassName[ ] = \_T("Chinese Chess");

int WINAPI WinMain (HINSTANCE hThisInstance,

HINSTANCE hPrevInstance,

LPSTR lpszArgument,

int nCmdShow) {

MSG messages; /\* Here messages to the application are saved \*/

WNDCLASSEX wincl; /\* Data structure for the windowclass \*/

/\* The Window structure \*/

wincl.hInstance = hThisInstance;

wincl.lpszClassName = szClassName;

wincl.lpfnWndProc = WindowProcedure; /\* This function is called by windows \*/

wincl.style = CS\_DBLCLKS; /\* Catch double-clicks \*/

wincl.cbSize = sizeof (WNDCLASSEX);

/\* Use default icon and mouse-pointer \*/

wincl.hIcon = LoadIcon (NULL, IDI\_APPLICATION);

wincl.hIconSm = LoadIcon (NULL, IDI\_APPLICATION);

wincl.hCursor = LoadCursor (NULL, IDC\_ARROW);

wincl.lpszMenuName = NULL; /\* No menu \*/

wincl.cbClsExtra = 0; /\* No extra bytes after the window class \*/

wincl.cbWndExtra = 0; /\* structure or the window instance \*/

/\* Use Windows's default colour as the background of the window \*/

wincl.hbrBackground = (HBRUSH) COLOR\_BACKGROUND;

/\* Register the window class, and if it fails quit the program \*/

if (!RegisterClassEx (&wincl))

return 0;

/\* The class is registered, let's create the program\*/

hwnd = CreateWindowEx (

0, /\* Extended possibilites for variation \*/

szClassName, /\* Classname \*/

\_T("Chinese Chess"), /\* Title Text \*/

WS\_OVERLAPPEDWINDOW, /\* default window \*/

CW\_USEDEFAULT, /\* Windows decides the position \*/

CW\_USEDEFAULT, /\* where the window ends up on the screen \*/

WIDTH, /\* The programs width \*/

HEIGHT, /\* and height in pixels \*/

HWND\_DESKTOP, /\* The window is a child-window to desktop \*/

NULL, /\* No menu \*/

hThisInstance, /\* Program Instance handler \*/

NULL /\* No Window Creation data \*/

);

/\* Make the window visible on the screen \*/

ShowWindow (hwnd, nCmdShow);

/\* Run the message loop. It will run until GetMessage() returns 0 \*/

while (GetMessage (&messages, NULL, 0, 0)) {

/\* Translate virtual-key messages into character messages \*/

TranslateMessage(&messages);

/\* Send message to WindowProcedure \*/

DispatchMessage(&messages);

}

/\* The program return-value is 0 - The value that PostQuitMessage() gave \*/

return messages.wParam;

}

//把数组坐标转换为界面坐标

void xy\_to\_pixel(int x,int y,int\*pixelx,int \*pixely) {

\*pixely=x\*PIECE\_WH+ZERO\_Y;

\*pixelx=y\*PIECE\_WH+ZERO\_X;

}

//把界面坐标转换为数组坐标

void pixel\_to\_xy(int pixelx,int pixely,int\*x,int \*y) {

int r=PIECE\_WH/2;

\*y=(pixelx-(ZERO\_X-r))/PIECE\_WH;

\*x=(pixely-(ZERO\_Y-r))/PIECE\_WH;

}

//以数组坐标画线

void draw\_line(HDC hdc,int sx,int sy,int dx,int dy) {

int psx,psy,pdx,pdy;

xy\_to\_pixel(sx,sy,&psx,&psy);

xy\_to\_pixel(dx,dy,&pdx,&pdy);

MoveToEx (hdc, psx,psy, NULL) ;

LineTo (hdc, pdx, pdy) ;

}

//以数组坐标画棋子

void paint\_piece(HDC hdc,int x,int y,int color,int type) {

static HBRUSH piece\_brush =CreateSolidBrush (PIECE\_BKCOLOR); //棋子的背景色

if(type==0||color==BLANK)return ;

int px,py;

xy\_to\_pixel(x,y,&px,&py);

int r=PIECE\_WH/2;

SelectObject (hdc,piece\_brush ) ;

SelectObject (hdc, GetStockObject (NULL\_PEN)) ;

Ellipse(hdc,px-r,py-r,px+r,py+r);

char \*text=new char[5];

switch(type) {

case JU:

strcpy(text,"車");

break;

case MA:

strcpy(text,"马");

break;

case XIANG:

if(color==BLACK)strcpy(text,"象");

else strcpy(text,"相");

break;

case SHI:

strcpy(text,"士");

break;

case JIANG:

if(color==BLACK)strcpy(text,"将");

else strcpy(text,"帅");

break;

case PAO:

if(color==BLACK)strcpy(text,"砲");

else

strcpy(text,"炮");

break;

case ZU:

if(color==BLACK)strcpy(text,"卒");

else

strcpy(text,"兵");

break;

default:

strcpy(text,"");

}

SetBkColor(hdc,PIECE\_BKCOLOR);//设置文字背景色

if(color==BLACK) {

SetTextColor(hdc,RGB(0,0,0)); //设置文字颜色

} else {

SetTextColor(hdc,RGB(255,255,255));

}

TextOut (hdc, px-r/2, py-r/2,text , strlen("马")) ;

delete text;

}

void\* main\_listen(void \*) {

server->listen\_message();

return 0;

}

//创建线程，使server开始监听

bool start\_listen() {

pthread\_t listen\_p;

int ret;

ret= pthread\_create( &listen\_p, NULL, main\_listen,NULL ); //

if( ret != 0 ) { //创建线程成功返回0

//printf("pthread\_create error:error\_code=%d\n",ret );

handle\_error(THREAD\_ERROR,true,true);

return false;

}

return true;

}

void\* chess\_connect(void \*) {

client->connect\_to\_ots(); //client开始连接对方server，连接成功后返回

InvalidateRect(hwnd,NULL,true);

}

void init() {

server=new Server();//创建Server对象

client=new Client(); //创建Client对象，

start\_listen(); //创建线程，server开始监听

Sleep(1000);

pthread\_t connect\_p;

int ret;

ret= pthread\_create( &connect\_p, NULL, chess\_connect,NULL); //

if( ret != 0 ) { //创建线程成功返回0

//printf("pthread\_create error:error\_code=%d\n",ret );

handle\_error(THREAD\_ERROR,true,true);

return ;

}

}

/\* This function is called by the Windows function DispatchMessage() \*/

LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {

static POINT mouse;

static HDC hdc;

static PAINTSTRUCT ps ;

static int iofip=0; //index of ots\_ip

switch (message) { /\* handle the messages \*/

case WM\_CREATE: {

port=35536;

ots\_ip=new char[20];

strcpy(ots\_ip,"");

}

break;

case WM\_KEYDOWN://识别按键，显示输入的内容（对方IP地址）

if(server!=NULL)break;

if(wParam==13) {//如果是ENTER，则初始化server、client，并开始连接，连接成功后初始化board

init();

Sleep(100);

InvalidateRect(hwnd,NULL,true);

}

if(wParam==VK\_BACK) {//删除键

if(iofip==0)return 0;

iofip--;

ots\_ip[iofip]='\0';

}

if(wParam<106&&wParam>95) {//小键盘数字键

wParam-=48;

}

if(wParam<58&&wParam>47) {//主键盘数字键

ots\_ip[iofip]='0'-48+wParam;

iofip++;

ots\_ip[iofip]='\0';

}

if(wParam==110||wParam==229) {//小数点键，小键盘110，主键盘229

ots\_ip[iofip]='.';

iofip++;

ots\_ip[iofip]='\0';

}

InvalidateRect(hwnd,NULL,true);

break;

case WM\_PAINT: {

static HBRUSH bk\_brush =CreateSolidBrush (RGB(240,240,240)); //棋子的背景色

hdc=BeginPaint (hwnd,&ps) ;

static HFONT hFont;

LOGFONT lf;

lf.lfHeight=PIECE\_WH/2;

lf.lfWidth=0;

lf.lfEscapement=0;

lf.lfOrientation=0 ;

lf.lfWeight=5;

lf.lfItalic=0 ;

lf.lfUnderline=0 ;

lf.lfStrikeOut=0 ;

lf.lfCharSet=DEFAULT\_CHARSET ;

lf.lfOutPrecision=0 ;

lf.lfClipPrecision=0 ;

lf.lfQuality=0 ;

lf.lfPitchAndFamily=0 ;

lstrcpy (lf.lfFaceName, \_T("楷体") );

hFont = CreateFontIndirect (&lf) ;

SelectFont(hdc,hFont);

SelectObject(hdc,bk\_brush);

Rectangle(hdc,0,0,WIDTH,HEIGHT);

SetBkColor(hdc,RGB(240,240,240));

if(chess\_board==NULL) {//显示输入的IP地址

char tip[20]="请输入对方IP地址：";

Rectangle(hdc,WIDTH/5,HEIGHT/2-10,WIDTH/5\*4,HEIGHT/2+30);

TextOut(hdc,WIDTH/5,HEIGHT/2-50,tip,strlen(tip));

SetBkColor(hdc,RGB(240,240,240));

TextOut(hdc,WIDTH/5+5,HEIGHT/2,ots\_ip,strlen(ots\_ip));

if(server!=NULL) { //board==NULL而server!=NULL表示正在连接过程中

char tip[20]="正在连接......";

SetBkColor(hdc,RGB(240,240,240));

TextOut(hdc,WIDTH/5,HEIGHT/2+50,tip,strlen(tip));

}

EndPaint(hwnd,&ps);

break;

}

char text[10]="你的颜色:";

if(chess\_board->get\_color()==BLACK) {

strcat(text," 黑");

} else {

strcat(text," 白");

}

TextOut (hdc, 5, 5,text , strlen(text)) ;

int M=chess\_board->get\_M();

int N=chess\_board->get\_N();

//画棋盘

for(int i=0; i<M; i++) {

draw\_line(hdc,i,0,i,N-1);

}

for(int i=0; i<N; i++) {

draw\_line(hdc,0,i,N/2,i);

}

for(int i=0; i<N; i++) {

draw\_line(hdc,N/2+1,i,N,i);

}

draw\_line(hdc,0,3,2,5);

draw\_line(hdc,0,5,2,3);

draw\_line(hdc,9,3,7,5);

draw\_line(hdc,7,3,9,5);

draw\_line(hdc,4,0,5,0);

draw\_line(hdc,4,8,5,8);

//画棋子

for(int i=0; i<M; i++) {

for(int j=0; j<N; j++) {

paint\_piece(hdc,i,j,chess\_board->get\_color(i,j),chess\_board->get\_type(i,j));

}

}

EndPaint(hwnd,&ps);

}

break;

case WM\_LBUTTONUP: {

if(chess\_board==NULL)break;

if(!chess\_board->is\_my\_turn())break;//当前没轮到自己下棋

GetCursorPos(&mouse);//获取鼠标的屏幕坐标

ScreenToClient(hwnd,&mouse);//转换为界面坐标

int x,y;

pixel\_to\_xy(mouse.x,mouse.y,&x,&y);//转换为数组坐标

if(chess\_board->get\_color(x,y)==chess\_board->get\_color()) {//点击的是自己的棋子

chess\_sx=x;

chess\_sy=y;

break;

}

if(chess\_sx==-1||chess\_sy==-1) {//起始坐标未赋值且点击的不是自己的棋子，则break

break;

}

chess\_dx=x;

chess\_dy=y;

if(chess\_board->my\_move\_piece(chess\_sx,chess\_sy,chess\_dx,chess\_dy)) { //如果移动棋子合法

client->send\_message("move",chess\_sx,chess\_sy,chess\_dx,chess\_dy); //向对方发送移子信息

InvalidateRect(hwnd,NULL,true);

if(chess\_board->get\_is\_win()==WIN) {

chess\_board->init();//重新初始化棋盘，重下一盘

MessageBox(hwnd,"你赢了","获胜！",NULL);

InvalidateRect(hwnd,NULL,true);

}

}

chess\_sx=-1;

chess\_sy=-1;

break;

}

case WM\_DESTROY:

if(server!=NULL)server->close();

if(client!=NULL)client->close();

PostQuitMessage (0); /\* send a WM\_QUIT to the message queue \*/

break;

default: /\* for messages that we don't deal with \*/

return DefWindowProc (hwnd, message, wParam, lParam);

}

return 0;

}

**chinese\_chess.h**

#ifndef CHINESE\_CHESS\_H\_INCLUDED

#define CHINESE\_CHESS\_H\_INCLUDED

#define JU 1

#define MA 2

#define XIANG 3

#define SHI 4

#define JIANG 5

#define PAO 6

#define ZU 7

#define BLANK 8 //空子

#define BLACK -1

#define WHITE 1

#define WIN 1

#define LOSE -1

class Board {

private:

bool turn; //是否轮到自己下棋

int color; //自己的颜色

int M,N; //棋盘行数、列数

int \*\*b; //二维数组

int is\_win; //是否胜利

bool is\_out(int x,int y) {//坐标是否出界

return x>M||y>N||x<0||y<0;

}

bool is\_same\_color(int sx,int sy,int dx,int dy) {//源坐标与目的坐标是否是同一颜色

return get\_color(sx,sy)==get\_color(dx,dy);

}

void swap\_num(int & num1,int& num2) {//交换两个数

num1+=num2;

num2=num1-num2;

num1=num1-num2;

}

int get\_abs(int num) {//取得绝对值

return num>=0?num:-num;

}

int num\_of\_not\_blank\_betweenn(int sx,int sy,int dx,int dy) {//返回在起始坐标和目的坐标之间棋子的个数

if(!(sx==dx||sy==dy))return -1;

int num=0;

if(sy>dy)swap\_num(sy,dy);

if(sx>dx)swap\_num(sx,dx);

if(sx==dx) {

for(int i=sy+1; i<dy; i++) {

if(b[sx][i]!=BLANK)num++;

}

}

if(sy==dy) {

for(int i=sx+1; i<dx; i++) {

if(b[i][sy]!=BLANK)num++;

}

}

return num;

}

bool is\_correct\_move\_JU(int sx,int sy,int dx,int dy) {

return num\_of\_not\_blank\_betweenn(sx,sy,dx,dy)==0;

}

bool is\_correct\_move\_MA(int sx,int sy,int dx,int dy) {

int x=dx-sx,y=dy-sy;

if(get\_abs(x)==2&&get\_abs(y)==1) {

if(get\_color(sx+x/2,sy)==BLANK)return true;//硌马蹄检测

}

if(get\_abs(x)==1&&get\_abs(y)==2) {

if(get\_color(sx,sy+y/2)==BLANK)return true;//硌马蹄检测

}

return false;

}

bool is\_correct\_move\_XIANG(int sx,int sy,int dx,int dy) {

int x=dx-sx,y=dy-sy;

if(!(get\_abs(x)==2&&get\_abs(y)==2)) return false;

if(get\_color(sx+x/2,sy+y/2)==BLANK)return true;//硌象蹄检测

return false;

}

bool is\_correct\_move\_SHI(int sx,int sy,int dx,int dy) {

int x=dx-sx,y=dy-sy;

if(!(get\_abs(x)==1&&get\_abs(y)==1)) return false;

if(dx<7)return false;

if(dy<3||dy>5)return false;

return true;

}

bool is\_correct\_move\_JIANG(int sx,int sy,int dx,int dy) {

int x=dx-sx,y=dy-sy;

if(!((get\_abs(x)==1&&get\_abs(y)==0)||(get\_abs(x)==0&&get\_abs(y)==1))) return false;

if(dx<7)return false;

if(dy<3||dy>5)return false;

for(int i=0; i<3; i++) {//明将检测

if(get\_type(i,dy)==JIANG) {

if(num\_of\_not\_blank\_betweenn(dx,dy,i,dy)==0) return false;

return true;

}

}

return true;

}

bool is\_correct\_move\_PAO(int sx,int sy,int dx,int dy) {

int n=get\_color(dx,dy)==BLANK?0:1;

return num\_of\_not\_blank\_betweenn(sx,sy,dx,dy)==n;

}

bool is\_correct\_move\_ZU(int sx,int sy,int dx,int dy) {

if(dx>sx)return false;

int x=dx-sx,y=dy-sy;

if(get\_abs(x)+get\_abs(y)!=1)return false;

if(sx>4&&get\_abs(x)!=1)return false;//过河前只能向前走

return true;

}

bool is\_correct\_move(int sx,int sy,int dx,int dy) {

if(sx==dx&&sy==dy) {

return false;

}

if(is\_out(sx,sy)||is\_out(dx,dy)) {

return false;

}

if(get\_color(sx,sy)!=color) {

return false;

}

if(is\_same\_color(sx,sy,dx,dy)) {

return false;

}

switch(get\_type(sx,sy)) {

case JU:

return is\_correct\_move\_JU(sx,sy,dx,dy);

case MA:

return is\_correct\_move\_MA(sx,sy,dx,dy);

case XIANG:

return is\_correct\_move\_XIANG(sx,sy,dx,dy);

case SHI:

return is\_correct\_move\_SHI(sx,sy,dx,dy);

case JIANG:

return is\_correct\_move\_JIANG(sx,sy,dx,dy);

case PAO:

return is\_correct\_move\_PAO(sx,sy,dx,dy);

case ZU:

return is\_correct\_move\_ZU(sx,sy,dx,dy);

default:

return false;

}

}

void move\_s\_to\_d(int sx,int sy,int dx,int dy) { //移动操作

if(get\_type(dx,dy)==JIANG) { //如果目的棋子是将

if(get\_color(dx,dy)==color)set\_win(LOSE);//如果是自己的将，则输

else set\_win(WIN);//如果是对方的将，则赢

}

b[dx][dy]=b[sx][sy];

b[sx][sy]=BLANK;

change\_turn();

}

void init\_pieces() {

for(int i=0; i<M; i+=M-1) {//第一行和最后一行（即车马象士将士象马车）

for(int index=0; index<N; index++) {

if(index<N/2+1)b[i][index]=index+1;

else b[i][index]=N-index;

}

}

//卒所在的行

for(int index=0; index<N; index+=2) {

b[3][index]=ZU;

}

for(int index=0; index<N; index+=2) {

b[6][index]=ZU;

}

b[2][1]=PAO;

b[M-1-2][1]=PAO;

b[2][N-1-1]=PAO;

b[M-1-2][N-1-1]=PAO;

int s,d;//存储起始行和终点行

if(color==BLACK) {

s=0;//从0行到M/2行，即棋盘上半部分

d=M/2;

} else {

s=M/2;//棋盘下半部分

d=M;

}

//从s行到d行，把非BLANK的值加BLANK，使小于BLANK的代表黑色棋，大于BLANK的代表白色棋

for(int index=s; index<d; index++) {

for(int j=0; j<N; j++) {

if(b[index][j]!=BLANK) {

b[index][j]+=BLANK;

}

}

}

}

public:

Board(int c) {

color=c;

M=10;

N=9;

b=new int\*[M];

for(int i=0; i<M; i++) {

b[i]=new int[N];

}

init();

}

void init() {//棋盘初始化

is\_win=0;

turn=color==BLACK?true:false;

for(int i=0; i<M; i++) {

for(int j=0; j<N; j++) {

b[i][j]=BLANK;

}

}

init\_pieces();

}

int get\_M() {

return M;

}

int get\_N() {

return N;

}

int get\_color() {//获取己方的颜色

return color;

}

int get\_color(int x,int y) {//获取棋盘某一坐标上棋子的颜色

return b[x][y]>BLANK?WHITE:b[x][y]<BLANK?BLACK:BLANK;

}

int get\_type(int x,int y) {//获取棋子类型（空、车、马、象、士、将、炮、卒）

return b[x][y]!=BLANK?b[x][y]%BLANK:BLANK;

}

void set\_win(int is) {

is\_win=is;

}

int get\_is\_win() {

return is\_win;

}

void change\_turn() {

turn=turn==true?false:true;

}

bool is\_my\_turn() {

return turn;

}

void othside\_move\_piece(int sx,int sy,int dx,int dy) {//对方移子

sx=M-1-sx;//先进行坐标转换，因对方视角与己方相反

sy=N-1-sy;

dx=M-1-dx;

dy=N-1-dy;

move\_s\_to\_d(sx,sy,dx,dy);

}

bool my\_move\_piece(int sx,int sy,int dx,int dy) { //己方主动移子

if(!is\_correct\_move(sx,sy,dx,dy))return false;

move\_s\_to\_d(sx,sy,dx,dy);

return true;

}

};

#endif // CHINESE\_CHESS\_H\_INCLUDED

**Server.h**

#ifndef SERVER\_H\_INCLUDED

#define SERVER\_H\_INCLUDED

#include<stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <winsock2.h>

#include"chinese\_chess.h"

#define INIT\_ERROR 1

#define BIND\_ERROR 2

#define LISTEN\_ERROR 3

#define CONNECT\_ERROR 4

#define SEND\_ERROR 5

#define ACCEPT\_ERROR 6

#define ALIVE\_ERROR 7

#define THREAD\_ERROR 8

int error\_num;

extern HWND hwnd;

extern Board\* chess\_board;

extern char\* ots\_ip;

extern int port;

extern bool is\_connect\_alive;

//线程参数结构体

typedef struct server\_args {

SOCKET\* Com\_Sock;

char \* rebuf;

} server\_args;

//校验检测，与客户端的添加校验是相反操作

//最后一位之前的所有字符相加取模后，如果等于最后一个字符，则校验通过

bool server\_check(char \* r) {

int len=strlen(r);

len--;

int s=0;

for(int i=0; i<len; i++) {

s+=r[i];

}

if(r[len]==(s%5+'0')) {

r[len]='\0';

return true;

}

return false;

}

//错误处理，is\_tell控制是否显示错误信息，is\_exit控制是否退出程序

int handle\_error(int err,bool is\_tell,bool is\_exit) {

error\_num=err;

if(!is\_tell)return error\_num;

char error[30]="";

switch(error\_num) {

case INIT\_ERROR:

strcpy(error,"初始化错误");

break;

case BIND\_ERROR:

strcpy(error,"绑定端口错误");

break;

case LISTEN\_ERROR:

strcpy(error,"监听错误");

break;

case ACCEPT\_ERROR:

strcpy(error,"接受连接错误");

break;

case CONNECT\_ERROR:

strcpy(error,"无法连接");

break;

case ALIVE\_ERROR:

strcpy(error,"连接已断开");

break;

case THREAD\_ERROR:

strcpy(error,"线程无法创建");

break;

case SEND\_ERROR:

strcpy(error,"发送错误");

}

char error\_message[50];

strcpy(error\_message,"错误：");

strcat(error\_message,error);

if(is\_exit)strcat(error\_message,"\n程序将退出。");

MessageBox(hwnd,error\_message,"错误",MB\_OK);

if(is\_exit)exit(0);

return error\_num;

}

void\* handle\_message(void\*ar) {

server\_args \* serarg=(server\_args \* )ar;

char \*recv=serarg->rebuf;

SOCKET\* CommandSock=serarg->Com\_Sock;

if(server\_check(recv)) {//校验通过发送okok（OK），不通过发送noto（NOTOK）

send(\*CommandSock,"okok",4,0);

} else {

send(\*CommandSock,"noto",4,0);

return ar;

}

if(strncmp(recv,"move",4)==0) {

char \* pch;

//将recvBuf以逗号拆分

pch = strtok (recv,",");

pch = strtok (NULL,",");

int xys[4];

int index=0;

while (pch != NULL) {

xys[index]=atoi(pch);//char\* 转换为int

index++;

pch = strtok (NULL, ",");

}

chess\_board->othside\_move\_piece(xys[0],xys[1],xys[2],xys[3]);

if(chess\_board->get\_is\_win()==LOSE) {

chess\_board->init();//如果输了，则重新初始化棋盘，再下一盘

MessageBox(hwnd,"你输了","失败！",NULL);

}

InvalidateRect(hwnd,NULL,true);

}

delete recv;

}

class Server {

private:

SOCKET Server\_Sock;

SOCKADDR\_IN server\_addr;

SOCKADDR\_IN client\_addr;

char recvBuf[20];

public:

Server() {

WSADATA wsa;

/\*初始化socket资源\*/

if (WSAStartup(MAKEWORD(1,1),&wsa) != 0) {

handle\_error(INIT\_ERROR,true,true);

return;

}

if((Server\_Sock = socket(AF\_INET, SOCK\_STREAM, 0))==-1) {

handle\_error(INIT\_ERROR,true,true);

return;

}

ZeroMemory((char \*)&server\_addr,sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(port); /\*本地监听端口\*/

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); /\*有IP\*/

if(bind(Server\_Sock,(struct sockaddr \*)&server\_addr,sizeof(server\_addr))==-1) {

handle\_error(BIND\_ERROR,true,true);

return;

}

if(listen(Server\_Sock,5)==-1) { //其中第二个参数代表能够接收的最多的连接数

handle\_error(LISTEN\_ERROR,true,true);

return;

}

strcpy(recvBuf,"");

}

void listen\_message() {

int len=sizeof(SOCKADDR);

while(true) {

SOCKET Command\_Sock = accept(Server\_Sock, (SOCKADDR\*)&client\_addr,&len);

if(Command\_Sock == INVALID\_SOCKET) {

closesocket(Command\_Sock);

handle\_error(ACCEPT\_ERROR,false,false);

continue;

}

if(client\_addr.sin\_addr.s\_addr!=inet\_addr(ots\_ip)) {//如果接收的socket不是预期的对方的，则发送wron，继续等待

send(Command\_Sock,"wron",4,0);

closesocket(Command\_Sock);

continue;

}

send(Command\_Sock,"righ",4,0);

is\_connect\_alive=true;

while(true) {

if(recv(Command\_Sock,recvBuf,20,0)<=0) {//recv返回小于等于0的值，则连接已断开

handle\_error(ALIVE\_ERROR,true,true);

closesocket(Command\_Sock);

close();

return ;

}

char \*rbuf=new char[20];

strcpy(rbuf,recvBuf);

server\_args serarg;

serarg.Com\_Sock=&Command\_Sock;

serarg.rebuf=rbuf;

pthread\_t handle\_m;

int ret;

ret= pthread\_create( &handle\_m, NULL, handle\_message,&serarg); //

if( ret != 0 ) { //创建线程成功返回0

// printf("pthread\_create error:error\_code=%d\n",ret );

handle\_error(THREAD\_ERROR,true,true);

return ;

}

strcpy(recvBuf,"");

}

closesocket(Command\_Sock);

}

}

void close() {

closesocket(Server\_Sock);

WSACleanup();

}

};

#endif // SERVER\_H\_INCLUDED  
**Client.h**

#ifndef CLIENT\_H\_INCLUDED

#define CLIENT\_H\_INCLUDED

#include <stdio.h>

#include <winsock2.h>

#include"chinese\_chess.h"

//为字符串添加校验信息，对所有字符求和，模5之后转化为字符放在字符串最后

void client\_check(char\* r) {

int len=strlen(r);

int s=0;

for(int i=0; i<len; i++) {

s+=r[i];

}

r[len]=s%5+'0';

r[len+1]='\0';

}

class Client {

private:

SOCKET Client\_Sock;

SOCKADDR\_IN server\_addr;

char sendBuf[20];

public:

Client() {

WSADATA wsa;

/\*初始化socket资源\*/

if (WSAStartup(MAKEWORD(1,1),&wsa) != 0) {

handle\_error(INIT\_ERROR,true,true);

return; //代表失败

}

if((Client\_Sock = socket(AF\_INET, SOCK\_STREAM, 0))==-1) {

handle\_error(INIT\_ERROR,true,true);

return; //代表失败

}

server\_addr.sin\_addr.S\_un.S\_addr=inet\_addr(ots\_ip);

server\_addr.sin\_family=AF\_INET;

server\_addr.sin\_port=htons(port);

strcpy(sendBuf,"");

}

void connect\_to\_ots() {

while(connect(Client\_Sock,(SOCKADDR\*)&server\_addr,sizeof(SOCKADDR)) ==-1) {

handle\_error(CONNECT\_ERROR,false,false);

Sleep(100);

//printf( "%d ", WSAGetLastError());

}

char rec[5];

recv(Client\_Sock,rec,4,0);

if(strncmp(rec,"wron",4)==0) { //收到wrong，说明对方所输入的IP不是己方IP

MessageBox(hwnd,"对方输入的IP不是你\n程序将退出","错误",NULL);

exit(-1);

}

//谁先连接谁是黑色

//如果server已经收到连接，则说明是对方先连接自己，则自己应为白色，否则自己是黑色

if(is\_connect\_alive) {

chess\_board=new Board(WHITE);

} else {

chess\_board=new Board(BLACK);

}

}

void close() {

closesocket(Client\_Sock);

WSACleanup();

}

int send\_message(char \* message) {

strcpy(sendBuf,message);

client\_check(sendBuf);

int len;

int try\_time=0;

while(true) {

len=send(Client\_Sock,sendBuf,strlen(sendBuf)+1,0);

if(len!=(strlen(sendBuf)+1)) {

handle\_error(SEND\_ERROR,false,false);

//printf( "%d ", WSAGetLastError());

}

char rec[5];

recv(Client\_Sock,rec,4,0);

if(strncmp(rec,"okok",4)==0) {//收到OK说明数据已经正确被对方收到

break;

}

if(try\_time>20) { //尝试20次，数据仍无法正确送达，则退出

handle\_error(SEND\_ERROR,true,true);

}

try\_time++;

}

return len;

}

int send\_message(const char \* message,int sx,int sy,int dx,int dy) {

char\* message\_temp=new char[20];

sprintf(message\_temp,"%s,%d,%d,%d,%d,",message,sx,sy,dx,dy);

int len=send\_message(message\_temp);

delete message\_temp;

return len;

}

};

#endif // CLIENT\_H\_INCLUDED