# 网络五子棋

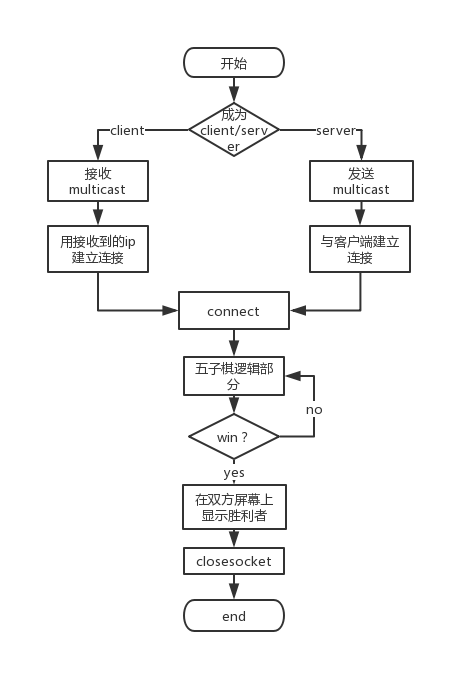
## 程序预期功能

### 本程序预期实现基于socket的双人网络五子棋对战。

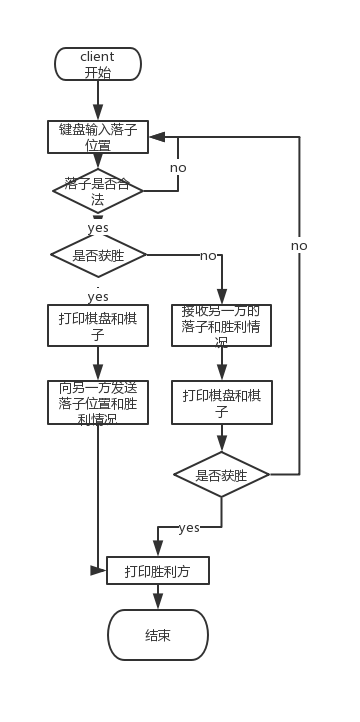
* 自动搜索同网络的其他玩家，搜索到后自动建立连接
* 双方轮流下棋，并将下的位置通过网络传向另一方
* 在本地判断是否胜利，并将胜利状态一并传输
* 在本地终端清屏后打印棋盘，棋局

## 程序运行流程

流程图1



流程图2



## 程序系统的结构/子模块

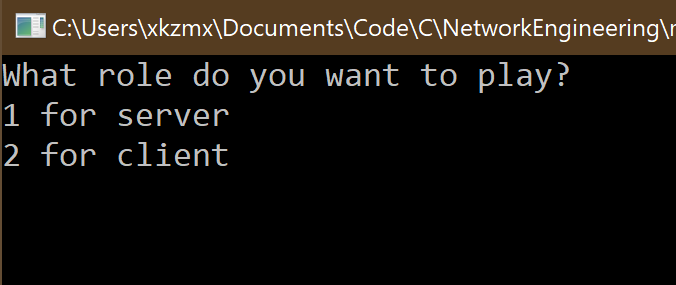


## 程序中用到的有关socket的技术

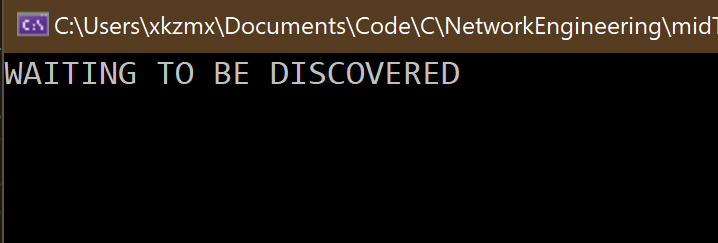
* 在寻找其他玩家时使用了multicast
* bind(s, (SOCKADDR\*)& sin, sizeof(sin));  
  mcast.imr\_interface.S\_un.S\_addr = INADDR\_ANY;  
  mcast.imr\_multiaddr.S\_un.S\_addr = inet\_addr("234.5.6.7");  
  setsockopt(s, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP, (char\*)& mcast, sizeof(mcast));
* s = socket(AF\_INET, SOCK\_DGRAM, 0);   
  mCast.sin\_family = AF\_INET;   
  mCast.sin\_port = htons(4567);   
  mCast.sin\_addr.S\_un.S\_addr = inet\_addr("234.5.6.7");
* 使用TCP传输棋局情况
* sockConn = socket(AF\_INET, SOCK\_STREAM, 0);   
  addrSrv.sin\_family = AF\_INET;   
  addrSrv.sin\_port = htons(1234);   
  addrSrv.sin\_addr.S\_un.S\_addr = inet\_addr(inet\_ntoa(addr));   
  connect(sockConn, (SOCKADDR\*)& addrSrv, sizeof(addrSrv));
* 在传输落子时使用结构体进行传输
* void sendStone(SOCKET sockClient, location stoneLocation, bool winStatus)   
  {   
     
   char win[2];   
   itoa(winStatus, win, 10);   
   send(sockClient, win, sizeof(win), 0);   
   char buff[100];   
   memcpy(buff, &stoneLocation, sizeof(stoneLocation));   
   send(sockClient, buff, 100, 0);   
  }
* void recvStone(SOCKET sockClient, location\* stoneLocation, bool \*winStatus)   
  {   
   char win[2];   
   recv(sockClient, win, sizeof(win), 0);   
   \*winStatus = atoi(win);   
   char recvBuf[100];   
   location recLo;   
   recv(sockClient, recvBuf, 100, 0);   
   memcpy(&recLo, recvBuf, sizeof(recLo));   
   stoneLocation->x = recLo.x;   
   stoneLocation->y = recLo.y;   
  }

## 程序运行截图

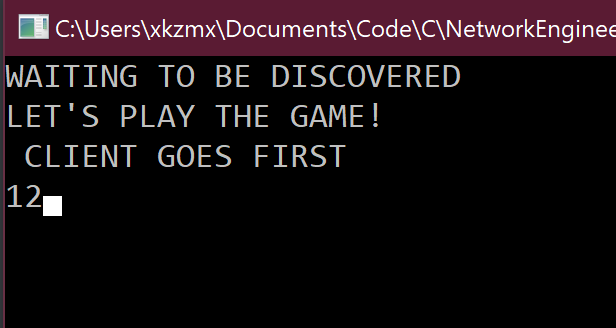
选择角色



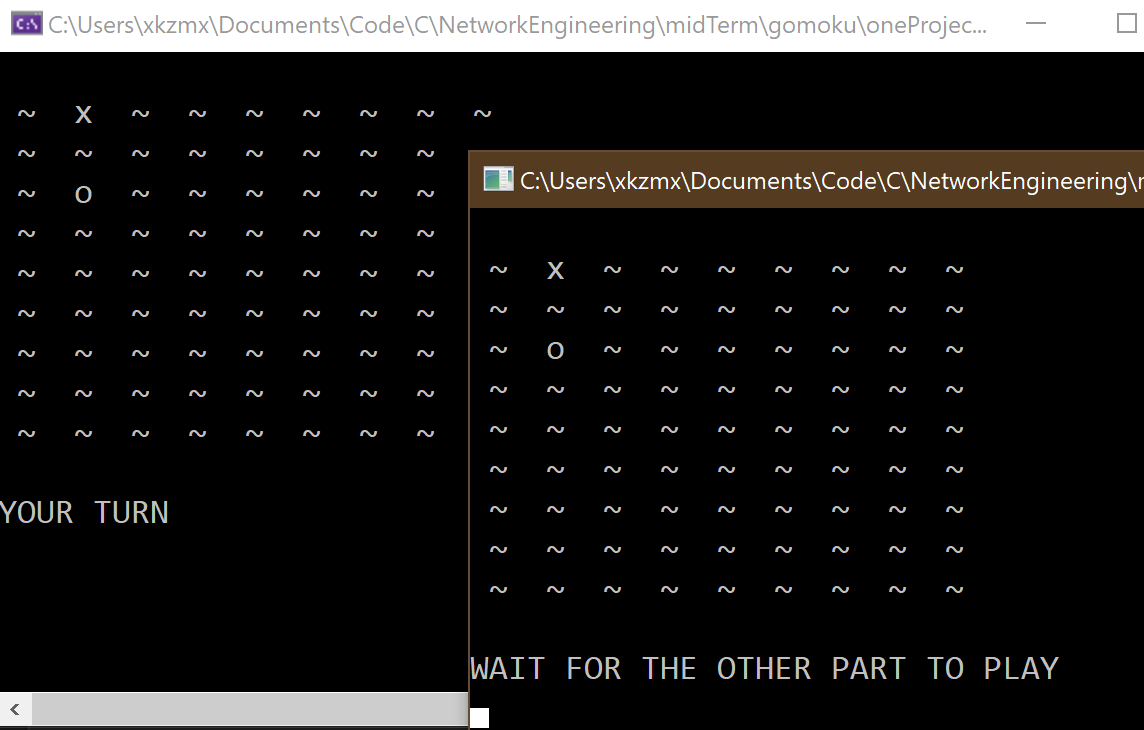
等待连接



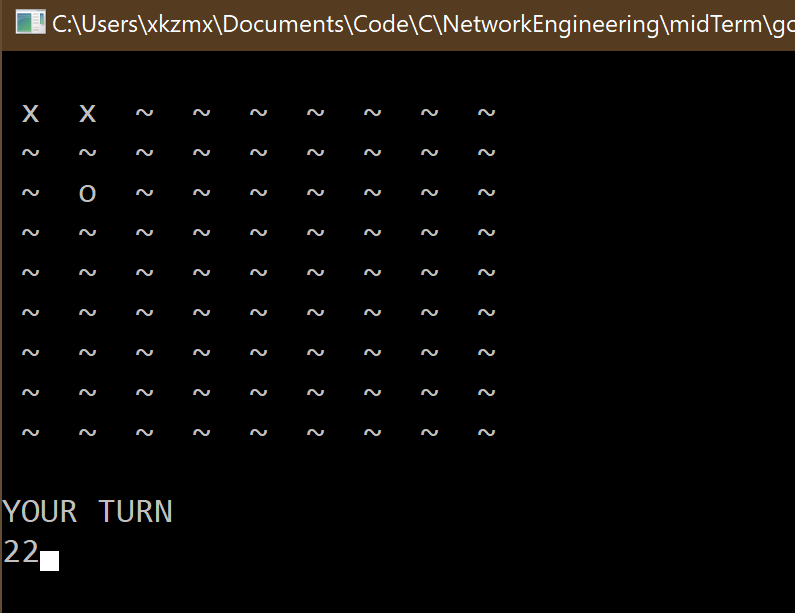
client先走



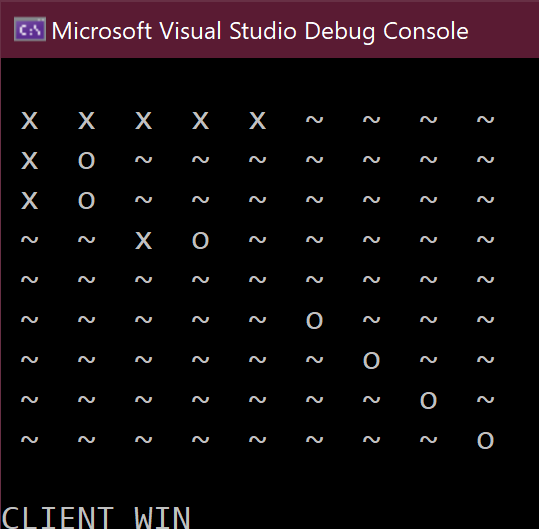
棋局同步



落子



结束，显示胜利者



## 程序源代码

#### 主程序

#include"gomoku.h"  
#include<stdio.h>  
int main()  
{  
 //gcc Precious.cpp gomoku.cpp gomoku.h -lws2\_32 -o output  
  
 printf("What role do you want to play?\n1 for server\n2 for client\n");  
 int choice;  
 scanf("%d", &choice);  
 system("cls");  
 switch (choice)  
 {  
 case 1:  
 playServer(); // choose to be server  
 break;  
 case 2:  
 playClient(); // choose to be server  
 break;  
 default:  
 printf("Wrong input");  
 break;  
 }  
 printf("PRESS ANY KEY TO EXIT");  
 getchar();  
 return 0;  
}

#### 头文件

#pragma once  
#include<WinSock2.h>  
#include<ws2tcpip.h>  
#include<stdio.h>  
#pragma comment(lib, "WS2\_32")  
#define boardSize 9  
typedef struct LOCATION {  
 int x;  
 int y;  
}location;  
  
class Stone  
{  
public:  
 location input;  
 Stone();  
 ~Stone();  
 bool addPiece(location input, bool client);  
 void display(void);  
 bool checkWin(bool client);  
 void readInput(location\* input, bool client);  
  
private:  
 char symobl[3];//set symbol for each other and void   
 int clientPiece;//record how many pieces the client has put  
 int serverPiece;//record how many pieces the server has put  
 location clientLocation[81];//record the exact location of the pieces  
 location serverLocation[81];//record the exact location of the pieces  
 char allPieces[boardSize \* boardSize];  
 bool pieceAvailability(location input);  
 bool findStone(location targetLocation, location stoneLocation[], int index, int stoneSize);  
 bool checkDiagonal(location tempLoca, location stoneLocation[], int i, int stoneSize);  
 bool checkVertical(location tempLoca, location stoneLocation[], int i, int stoneSize);  
 bool checkHorizontial(location tempLoca, location stoneLocation[], int i, int stoneSize);  
};  
  
class Server  
{  
private:  
 WORD mVersionRequested;  
 WSADATA wsaData;  
 SOCKADDR\_IN addrClient;  
 SOCKADDR\_IN addrSrv;  
 SOCKET sockSrv;  
 int len;  
public:  
 SOCKET sockConn;  
 Server();  
 ~Server();  
};  
  
class Client  
{  
private:  
 WORD mVersionRequested;  
 WSADATA wsaData;  
 SOCKADDR\_IN addrSrv;  
 // char serverAddr[50];  
 // void getServerIP(char\* serverAddr);  
public:  
 SOCKET sockConn;  
 Client(in\_addr addr);  
 ~Client();  
};  
  
class castCli  
{  
 private:  
 WORD mVersionRequested;  
 WSADATA wsaData;  
 int err;  
 SOCKET s;  
 SOCKADDR\_IN mCast;  
 public:  
 castCli();  
 ~castCli();  
};  
class castSer  
{  
 private:  
 WORD mVersionRequested;  
 WSADATA wsaData;  
 int err;  
 SOCKET s;  
 SOCKADDR\_IN sin;  
 ip\_mreq mcast;  
   
 int nAddrLen;  
 char buf[128];  
 public:  
 sockaddr\_in addrfrom;  
 castSer();  
 ~castSer();  
};  
  
//network logic  
void sendStone(SOCKET sockClient, location stoneLocation, bool win);// send input location to the other part  
void recvStone(SOCKET sockClient, location\* stoneLocation, bool \*win);// rece input location from the other part  
  
// supportive func  
int compare1(const void\* a, const void\* b);  
  
// game func  
void playClient(void);  
void playServer(void);

#### 函数主体

#include<stdlib.h>  
#include<algorithm>  
#include"gomoku.h"  
#include<WinSock2.h>  
#include<stdio.h>  
#include<string>  
#pragma comment(lib, "WS2\_32")  
  
Stone::Stone()  
{  
 symobl[0] = '~';// no piece on this location  
 symobl[1] = 'x';// client piece on this location  
 symobl[2] = 'o';// server piece on this location  
 clientPiece = 0;  
 serverPiece = 0;  
 for (int i = 0; i < boardSize \* boardSize; i++)  
 allPieces[i] = symobl[0];  
}  
  
Stone::~Stone()  
{  
}  
  
bool Stone::addPiece(location input, bool client)  
{  
 // Discription: add a piece to the current gomoku  
 // Input: input: location of the piece that need to be added  
 // client: true for client/ false for server}  
 // Output: NULL  
 bool availability;  
 availability = pieceAvailability(input);  
 if (!availability)  
 return false;  
 if (client)  
 {  
 this->clientLocation[this->clientPiece] = input;  
 this->clientPiece++;  
 location stoneLoca[81];  
 for (int i = 0; i < this->clientPiece; i++)  
 stoneLoca[i] = this->clientLocation[i];  
 qsort(stoneLoca, this->clientPiece, sizeof(stoneLoca[0]),compare1);  
 //qsort(stoneLoca, this->clientPiece, sizeof(stoneLoca[0]), compare2);  
 for (int i = 0; i < this->clientPiece; i++)  
 this->clientLocation[i] = stoneLoca[i];  
 this->allPieces[(input.x - 1) \* boardSize + input.y - 1] = this->symobl[1];  
  
 }  
 else  
 {  
 this->serverLocation[this->serverPiece] = input;  
 this->serverPiece++;  
 location stoneLoca[81];  
 for (int i = 0; i < this->serverPiece; i++)  
 stoneLoca[i] = this->serverLocation[i];  
 qsort(stoneLoca, this->serverPiece, sizeof(stoneLoca[0]), compare1);  
 //qsort(stoneLoca, this->serverPiece, sizeof(stoneLoca[0]), compare2);  
 for (int i = 0; i < this->serverPiece; i++)  
 this->serverLocation[i] = stoneLoca[i];  
 this->allPieces[(input.x - 1) \* boardSize + input.y - 1] = this->symobl[2];  
 }  
 return true;  
}  
  
bool Stone::pieceAvailability(location input)  
{  
 char piece = this->allPieces[(input.x - 1) \* boardSize + input.y - 1];  
 if (piece != this->symobl[0])  
 return false;  
 return true;  
}  
  
void Stone::display(void)  
{  
 // Discription: display the current gomoku board  
 // Input: current the current gomoku board  
 // Output: NULL  
 system("cls");  
 for (int i = 0; i < boardSize; i++)  
 {  
 printf("\n");  
 for (int j = 0; j < boardSize; j++)  
 printf(" %c ", this->allPieces[i \* boardSize + j]);  
 }  
 printf("\n");  
 }  
  
bool Stone::checkWin(bool client)  
{  
 // Discription: check after input did you win  
 // Input: current: the current gomoku board  
 // client: true to check client win/ false to check server win  
 // Output: bool win  
 int stoneSize;  
 location stoneLocation[81];  
 bool winStatus = false;  
 if (client)  
 {  
 stoneSize = this->clientPiece;  
 for (int i = 0; i < stoneSize; i++)  
 stoneLocation[i] = this->clientLocation[i];  
 }  
 else  
 {  
 stoneSize = this->serverPiece;  
 for (int i = 0; i < stoneSize; i++)  
 stoneLocation[i] = this->serverLocation[i];  
 }  
  
 if (stoneSize < 5)  
 return false;  
 //check horizontial  
 for(int i =0;i<stoneSize;i++)  
 {  
 location tempLoca = stoneLocation[i];  
 if (stoneSize - i < 5)  
 return winStatus;  
 if(checkHorizontial(tempLoca,stoneLocation,i, stoneSize)||checkVertical(tempLoca,stoneLocation,i,stoneSize)||checkDiagonal(tempLoca,stoneLocation,i,stoneSize))  
 {  
 winStatus = true;  
 return winStatus;  
 }  
 }  
  
 return winStatus;  
}  
  
void Stone::readInput(location\* input, bool client)  
{  
 char inLocation[3];  
 scanf("%s", &inLocation);  
 int temp = atoi(inLocation);  
 input->x = temp / 10;  
 input->y = temp % 10;  
 this->addPiece(\*input, client);  
}  
  
bool Stone::checkHorizontial(location tempLoca, location stoneLocation[],int i,int stoneSize)  
{  
 //check horizontial  
  
 bool winStatus = false;  
 if ((boardSize-tempLoca.y) < 4)  
 return winStatus;  
 else  
 {  
 int ylabel = tempLoca.y;  
 for (int j = i+1; j < i+5; j++)  
 {  
 ylabel++;  
 if (stoneLocation[j].y != ylabel)  
 {  
 winStatus = false;  
 break;  
 }  
 if (j == i + 4)  
 winStatus = true;  
 }  
 }  
 return winStatus;  
}  
  
bool Stone::checkVertical(location tempLoca, location stoneLocation[],int i,int stoneSize)  
{  
 //check vertical  
  
 bool winStatus = false;  
 if ((boardSize-tempLoca.x) < 4)  
 winStatus = false;  
 else  
 {  
 //int xlabel = tempLoca.x;  
 if(findStone({tempLoca.x+1,tempLoca.y},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+2,tempLoca.y},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+3,tempLoca.y},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+4,tempLoca.y},stoneLocation,i,stoneSize))  
 winStatus = true;  
 }  
 return winStatus;  
}  
  
bool Stone::checkDiagonal(location tempLoca, location stoneLocation[],int i,int stoneSize)  
{  
 //check diagonal  
  
 bool winStatus = false;  
 //check left  
 if(tempLoca.x <5)  
 winStatus = false;  
 else  
 {  
 if((boardSize-tempLoca.y) < 4)  
 winStatus = false;  
 else  
 {  
 if(findStone({tempLoca.x+1,tempLoca.y-1},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+2,tempLoca.y-2},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+3,tempLoca.y-3},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+4,tempLoca.y-4},stoneLocation,i,stoneSize))  
 winStatus = true;  
 }  
   
 }  
 //check right  
  
 if(winStatus)  
 return winStatus;  
 else if((boardSize-tempLoca.x) < 4)  
 winStatus = false;  
 else  
 {  
 if(findStone({tempLoca.x+1,tempLoca.y+1},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+2,tempLoca.y+2},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+3,tempLoca.y+3},stoneLocation,i,stoneSize)&&findStone({tempLoca.x+4,tempLoca.y+4},stoneLocation,i,stoneSize))  
 winStatus = true;  
 }  
   
 return winStatus;  
  
}  
  
bool Stone::findStone(location targetLocation, location stoneLocation[], int index, int stoneSize)  
{  
 bool discovered = false;  
 for (int i = index;i<stoneSize;i++)  
 {  
 if((stoneLocation[i].x == targetLocation.x)&&(stoneLocation[i].y == targetLocation.y))  
 {  
 discovered = true;  
 break;  
 }  
 }  
 return discovered;  
}  
  
Server::Server()  
{  
  
 mVersionRequested = MAKEWORD(2, 2);  
 WSAStartup(mVersionRequested, &wsaData);  
 sockSrv = socket(AF\_INET, SOCK\_STREAM, 0);  
 addrSrv.sin\_family = AF\_INET;  
 addrSrv.sin\_port = htons(1234);  
 addrSrv.sin\_addr.S\_un.S\_addr = htonl(INADDR\_ANY);  
 bind(sockSrv, (SOCKADDR\*)& addrSrv, sizeof(addrSrv));  
 listen(sockSrv, 10);  
 len = sizeof(addrClient);  
 sockConn = accept(sockSrv, (SOCKADDR\*)& addrClient, &len);  
}  
  
Server::~Server()  
{  
 closesocket(sockConn);  
}  
  
Client::Client(in\_addr addr)  
{  
 mVersionRequested = MAKEWORD(2, 2);  
 WSAStartup(mVersionRequested, &wsaData);  
 sockConn = socket(AF\_INET, SOCK\_STREAM, 0);  
 addrSrv.sin\_family = AF\_INET;  
 addrSrv.sin\_port = htons(1234);  
 //getServerIP(serverAddr);  
 addrSrv.sin\_addr.S\_un.S\_addr = inet\_addr(inet\_ntoa(addr));  
 connect(sockConn, (SOCKADDR\*)& addrSrv, sizeof(addrSrv));  
}  
  
Client::~Client()  
{  
 closesocket(sockConn);  
}  
  
//network logic  
//void Client::getServerIP(char\* serverAddr)  
//{  
// printf("Please input the ip address of the server\n");  
// //fgets(serverAddr, 50, stdin);  
// scanf("%s", serverAddr);  
//}  
  
castSer::castSer()  
{  
 mVersionRequested = MAKEWORD(2, 2);  
 WSAStartup(mVersionRequested, &wsaData);  
 s = socket(AF\_INET, SOCK\_DGRAM, 0);  
 sin.sin\_family = AF\_INET;  
 sin.sin\_port = htons(4567);  
 sin.sin\_addr.S\_un.S\_addr = INADDR\_ANY;  
 bind(s, (SOCKADDR\*)& sin, sizeof(sin));  
 mcast.imr\_interface.S\_un.S\_addr = INADDR\_ANY;  
 mcast.imr\_multiaddr.S\_un.S\_addr = inet\_addr("234.5.6.7");  
 setsockopt(s, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP, (char\*)& mcast, sizeof(mcast));  
 nAddrLen = sizeof(addrfrom);  
 while (true)  
 {  
 int nRet = recvfrom(s, buf, strlen(buf), 0, (sockaddr\*)& addrfrom, &nAddrLen);  
   
 if (nRet != SOCKET\_ERROR)  
 break;  
 }  
}  
castSer::~castSer()  
{  
 closesocket(s);  
 WSACleanup();  
  
}  
  
castCli::castCli()  
{  
 mVersionRequested = MAKEWORD(2, 2);  
 WSAStartup(mVersionRequested, &wsaData);  
 s = socket(AF\_INET, SOCK\_DGRAM, 0);  
 mCast.sin\_family = AF\_INET;  
 mCast.sin\_port = htons(4567);  
 mCast.sin\_addr.S\_un.S\_addr = inet\_addr("234.5.6.7");  
 char sz[] = "This is just a placeholder. \r\n";  
 sendto(s, sz, strlen(sz), 0, (sockaddr\*)& mCast, sizeof(mCast));  
}  
castCli::~castCli()  
{  
 closesocket(s);  
 WSACleanup();  
}  
  
void sendStone(SOCKET sockClient, location stoneLocation, bool winStatus)  
{  
  
 // char x[2];  
 // char y[2];  
 char win[2];  
 // itoa(stoneLocation.x, x, 10);  
 // itoa(stoneLocation.y, y, 10);  
 itoa(winStatus, win, 10);  
 // send(sockClient, x, sizeof(x), 0);  
 // send(sockClient, y, sizeof(y), 0);  
 send(sockClient, win, sizeof(win), 0);  
 char buff[100];  
 memcpy(buff, &stoneLocation, sizeof(stoneLocation));  
 send(sockClient, buff, 100, 0);  
  
}  
  
void recvStone(SOCKET sockClient, location\* stoneLocation, bool \*winStatus)  
{  
 // char x[2];  
 // char y[2];  
 char win[2];  
 // recv(sockClient, x, sizeof(x), 0);  
 // recv(sockClient, y, sizeof(y), 0);  
 recv(sockClient, win, sizeof(win), 0);  
 // stoneLocation->x = atoi(x);  
 // stoneLocation->y = atoi(y);  
 \*winStatus = atoi(win);  
 char recvBuf[100];  
 location recLo;  
 recv(sockClient, recvBuf, 100, 0);  
 memcpy(&recLo, recvBuf, sizeof(recLo));  
 stoneLocation->x = recLo.x;  
 stoneLocation->y = recLo.y;  
}  
  
//supportive function  
int compare1(const void\* a, const void\* b)  
{  
  
 location\* stone1 = (location\*)a;  
 location\* stone2 = (location\*)b;  
 if (stone1->x != stone2->x)  
 return (stone1->x - stone2->x);  
 else  
 {  
 return(stone1->y - stone2->y);  
 }  
}  
  
void playClient(void)  
{  
 castSer ser;  
 Client clientSock(ser.addrfrom.sin\_addr);  
 Stone clientStone;  
 location recvLocation;  
 printf("LET'S PLAY THE GAME!\n CLIENT GOES FIRST\n");  
 bool cliWinStatus = false;  
 bool serWinStatus = false;  
 for (int i = 0; i < boardSize \* boardSize; i++)  
 {  
 clientStone.readInput(&clientStone.input, true);  
 clientStone.display();  
 cliWinStatus = clientStone.checkWin(true);  
 sendStone(clientSock.sockConn, clientStone.input, cliWinStatus);  
 if (cliWinStatus || serWinStatus)  
 break;  
 printf("\nWAIT FOR THE OTHER PART TO PLAY\n");  
 recvStone(clientSock.sockConn, &recvLocation, &serWinStatus);  
 if (cliWinStatus || serWinStatus)  
 break;  
 clientStone.addPiece(recvLocation, false);  
 clientStone.display();  
 printf("\nYOUR TURN\n");  
 }  
 if (cliWinStatus)  
 printf("\nCLIENT WIN\n");  
 else if (serWinStatus)  
 printf("\nSERVER WIN\n");  
 else  
 printf("\nNO WINNER\n");  
}  
  
void playServer(void)  
{  
 castCli cli;  
 Server serverSock;  
 Stone serverStone;  
 location recvLocation;  
 printf("LET'S PLAY THE GAME!\n CLIENT GOES FIRST");  
 bool cliWinStatus = false;  
 bool serWinStatus = false;  
 for (int i = 0; i < boardSize \* boardSize; i++)  
 {  
 printf("\nWAIT FOR THE OTHER PART TO PLAY\n");  
 recvStone(serverSock.sockConn, &recvLocation, &cliWinStatus);  
 serverStone.addPiece(recvLocation, true);  
 serverStone.display();  
 if (cliWinStatus || serWinStatus)  
 break;  
 printf("\nYOUR TURN\n");  
 serverStone.readInput(&serverStone.input, false);  
 serWinStatus = serverStone.checkWin(false);  
 sendStone(serverSock.sockConn, serverStone.input, serWinStatus);  
 serverStone.display();  
 if (cliWinStatus || serWinStatus)  
 break;  
 }  
 if (cliWinStatus)  
 printf("\nCLIENT WIN\n");  
 else if (serWinStatus)  
 printf("\nSERVER WIN\n");  
 else  
 printf("\nNO WINNER\n");  
  
}  
  
//int compare2(const void\* a, const void\* b)  
//{  
//  
// location\* stone1 = (location\*)a;  
// location\* stone2 = (location\*)b;  
//  
// return (stone1->y - stone2->y);  
//}