# 基于tcp/ip通信的双人五子棋小游戏

## 项目需求

1. **开发背景**

本次项目基于c++语言，通过开发一个基于tcp通信的双人五子棋游戏，希望能够深入理解面向对象编程的知识，对封装，多态，继承加深理解，同时承接c的网络编程知识。

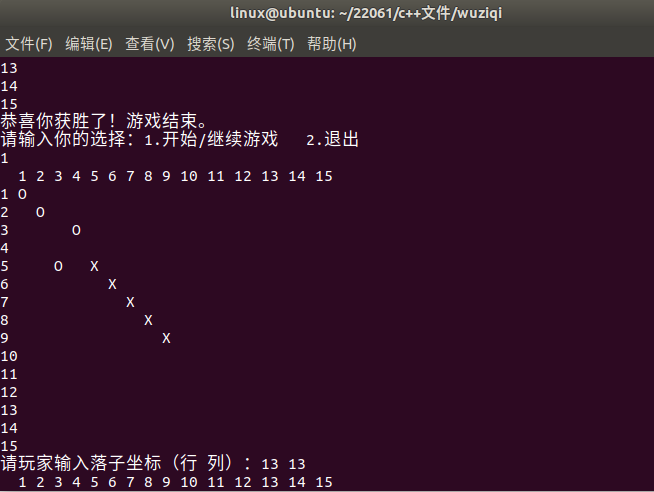
1. **运行环境**

本系统基于WINDOWS socket接口设计,适用于linux等系统。

1. **项目描述**

本项目通过c++和tcp通信技术，实现双人对战五子棋游戏，双方通过输入落子位置进行棋盘操作，包含胜利，和棋，失败三种结果，通过终端界面对棋盘进行实时显示，在游戏结束后弹出对话提示，玩家可以在自己想要退出的时候退出。

1. **效果图**



1. **功能说明**

1.在建立tcp连接后双方输入选择开始游戏和退出

2.对棋盘战况，棋子有实时显示。

3.任意一个方向五子连成一线时游戏胜利。

4.对于胜利方和失败方有不同的提示。

5.在玩家想退出时按2即可退出。

## 二、项目分析

根据系统功能要求，首先，将整个游戏封装成一个类Broad，Board类实现了一个五子棋的棋盘，包含了棋盘的初始化、打印、判断胜负等操作。其中，私有成员变量board保存了棋盘状态，大小为BOARD\_SIZE \* BOARD\_SIZE。构造函数用memset将board初始化为空格，print函数用于在控制台输出当前的棋盘状态，isFull函数判断棋盘是否已经下满，isValidMove函数用于判断指定位置是否可以下子，makeMove函数用于在指定位置下子，checkWin函数判断该位置下子后是否已经胜利。

TCPConnection类实现了一个TCP连接，包含了socket文件描述符的存储和数据的接收与发送。构造函数需要传入一个已经建立好的socket连接的文件描述符，receive函数用于接收对方发送的消息，send函数用于向对方发送消息。

在棋盘的表现形式上，我们使用一个15\*15的字符数组来表示，使用“O”和”X”表示棋子，用这两种字符占据字符数组中的空值表示落子，接收到落子信息或者发出落子信息时都会对该字符数组遍历输出表示棋盘的动态更新，当棋盘占满时和棋退出，每次落子时都会使用四个方向的for循环判断对应方向的落子与本次落子是否相同，相同count++,count等于5时游戏结束，判断某方胜利。

## 三、项目设计

### 1．类的设计

### // 棋盘类

### class Board {

### private:

### char board[BOARD\_SIZE][BOARD\_SIZE];//设置棋盘大小

### public:

### Board() //初始化棋盘

### void print() //在终端打印棋盘

### bool isFull()//判断棋盘是否下满，作为和棋的依据

### bool isValidMove(int x, int y)//用于判断指定位置是否可以下子

### void makeMove(int x, int y, char player) //在指定位置下子

### bool checkWin(int x, int y) //判断该位置下子后是否已经胜利

### };

### // TCP 连接类,实现了一个TCP连接

### class TCPConnection {

### private:

### int socket\_fd;//用于存储 socket 文件描述符

### public:

### TCPConnection(int sock) : socket\_fd(sock) {}//构造函数

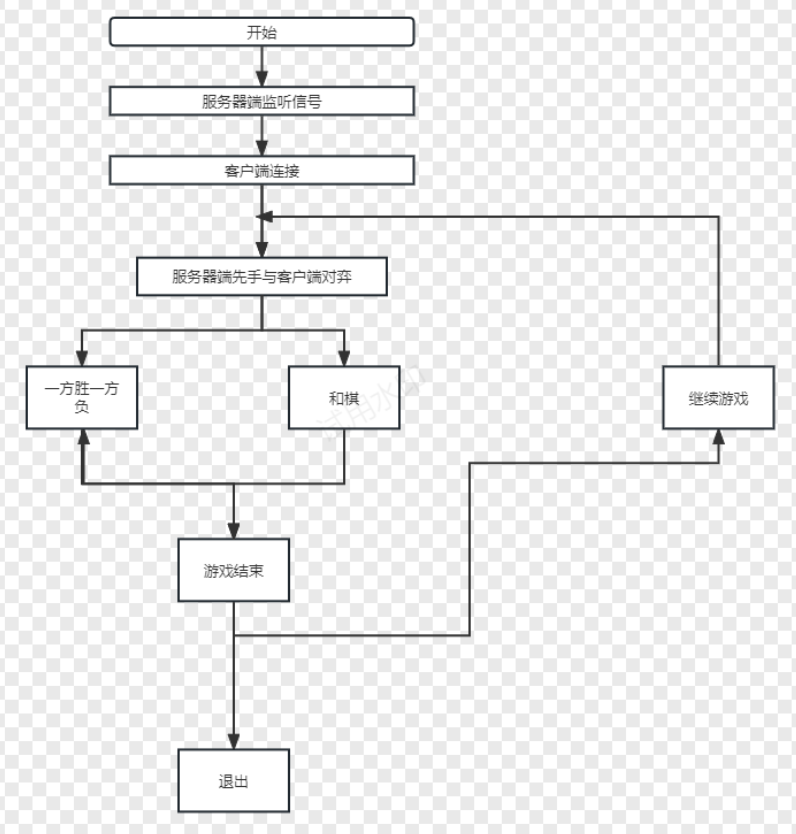
### std::string receive() //用于接收对方发送的消息，并以字符串形式返回接收到的消息

### void send(const std::string& message) //函数用于向对方发送消息，参数 message 是要发送的消息内容，类型为 std::string

### };

### 2．主程序设计

程序流程图如图所示:



程序流程图

1. **具体代码**
2. **头文件wuziqi.h**

#ifndef WUZIQI\_H

#define WUZIQI\_H

#include<iostream>

#include <string>

#include <cstring> // For memset

#include <sys/socket.h>

#include <arpa/inet.h>

#include<unistd.h>

#include<sstream>

const int BOARD\_SIZE = 15;

// 棋盘类:

class Board {

private:

char board[BOARD\_SIZE][BOARD\_SIZE];

public:

Board() {

memset(board, ' ', sizeof(board));

}

void print() {

std::cout << " ";

for (int i = 0; i < BOARD\_SIZE; ++i) {

std::cout << i + 1 << ' ';

}

std::cout << std::endl;

for (int i = 0; i < BOARD\_SIZE; ++i) {

std::cout << i + 1 << ' ';

for (int j = 0; j < BOARD\_SIZE; ++j) {

std::cout << board[i][j] << ' ';

}

std::cout << std::endl;

}

}

bool isFull() {

for (int i = 0; i < BOARD\_SIZE; ++i) {

for (int j = 0; j < BOARD\_SIZE; ++j) {

if (board[i][j] == ' ') {

return false;

}

}

}

return true;

}

bool isValidMove(int x, int y) {

return x >= 0 && y >= 0 && x < BOARD\_SIZE&& y < BOARD\_SIZE&& board[x][y] == ' ';

}

void makeMove(int x, int y, char player) {

board[x][y] = player;

}

bool checkWin(int x, int y) {

char player = board[x][y];

// 水平方向

int count = 0;

for (int i = y - 4; i <= y + 4; ++i) {

if (i < 0 || i >= BOARD\_SIZE) {

continue;

}

if (board[x][i] == player) {

count++;

}

else {

count = 0;

}

if (count == 5) {

return true;

}

}

// 垂直方向

count = 0;

for (int i = x - 4; i <= x + 4; ++i) {

if (i < 0 || i >= BOARD\_SIZE) {

continue;

}

if (board[i][y] == player) {

count++;

}

else {

count = 0;

}

if (count == 5) {

return true;

}

}

// 左上到右下斜线

count = 0;

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

int row = x + i;

int col = y + i;

if (row < 0 || row >= BOARD\_SIZE || col < 0 || col >= BOARD\_SIZE) {

continue;

}

if (board[row][col] == player) {

count++;

}

else {

count = 0;

}

if (count == 5) {

return true;

}

}

// 右上到左下斜线

count = 0;

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

int row = x + i;

int col = y - i;

if (row < 0 || row >= BOARD\_SIZE || col < 0 || col >= BOARD\_SIZE) {

continue;

}

if (board[row][col] == player) {

count++;

}

else {

count = 0;

}

if (count == 5) {

return true;

}

}

return false;

}

};

// TCP 连接类

class TCPConnection {

private:

int socket\_fd;

public:

TCPConnection(int sock) : socket\_fd(sock) {}

std::string receive() {

char buffer[1024];

memset(buffer, 0, sizeof(buffer));

recv(socket\_fd, buffer, sizeof(buffer), 0);

return std::string(buffer);

}

void send(const std::string& message) {

::send(socket\_fd, message.c\_str(), message.length(), 0);

}

};

#endif//WUZIQI\_H

1. **play1**

#include <iostream>

#include<unistd.h>

#include<sstream>

#include <string>

#include <cstring> // For memset

#include <sys/socket.h>

#include <arpa/inet.h>

#include "../include/wuziqi.h"

int main() {

// 创建 socket

int listen\_socket = socket(AF\_INET6, SOCK\_STREAM, 0);

// 绑定地址和端口

sockaddr\_in6 server\_addr{};

server\_addr.sin6\_family = AF\_INET6;

server\_addr.sin6\_port = htons(8888); // 设置端口号

server\_addr.sin6\_addr = in6addr\_any; // 使用任意可用网络地址

bind(listen\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr));

// 监听连接请求

listen(listen\_socket, 1);

std::cout << "等待玩家加入..." << std::endl;

// 接受客户端连接

sockaddr\_in6 client\_addr{};

socklen\_t client\_addr\_len = sizeof(client\_addr);

int client\_socket = accept(listen\_socket, (sockaddr\*)&client\_addr, &client\_addr\_len);

std::cout << "玩家已加入，可以开始游戏！" << std::endl;

// 创建棋盘对象

Board board;

// 创建 TCP 连接对象

TCPConnection connection(client\_socket);

do {

int choice = 0;

std::cout << "请输入你的选择：1.开始/继续游戏 2.退出" << std::endl;

std::cin >> choice;

if (choice == 1) {

board = Board();

while (true) {

// 显示棋盘

board.print();

// 等待玩家落子

std::cout << "请玩家输入落子坐标（行 列）：";

int x, y;

std::cin >> x >> y;

// 检查落子是否有效

if (!board.isValidMove(x - 1, y - 1)) {

std::cout << "无效的落子位置，请重新输入！" << std::endl;

continue;

}

// 更新棋盘状态

board.makeMove(x - 1, y - 1, 'X');

board.print();

// 发送落子信息给对手

std::string move = std::to\_string(x) + " " + std::to\_string(y);

connection.send(move);

// 检查胜负

if (board.checkWin(x - 1, y - 1)) {

std::cout << "恭喜你获胜了！游戏结束。" << std::endl;

break;

}

// 判断平局

if (board.isFull()) {

std::cout << "平局！游戏结束。" << std::endl;

break;

}

// 等待对手落子

std::cout << "等待对手落子..." << std::endl;

std::string opponent\_move = connection.receive();

std::istringstream iss(opponent\_move);

iss >> x >> y;

// 更新棋盘状态

board.makeMove(x - 1, y - 1, 'O');

board.print();

// 检查胜负

if (board.checkWin(x - 1, y - 1)) {

std::cout << "对手获胜！游戏结束。" << std::endl;

break;

}

}

} else if (choice == 2) {

close(listen\_socket);

close(client\_socket);

return 0;

} else {

std::cout << "你的选择有误，请重新输入！" << std::endl;

}

} while (true);

return 0;

}

1. **Play2**

#include<iostream>

#include <string>

#include <cstring> // For memset

#include <sys/socket.h>

#include <arpa/inet.h>

#include<unistd.h>

#include<sstream>

#include "../include/wuziqi.h"

int main() {

// 创建 socket

int client\_socket = socket(AF\_INET6, SOCK\_STREAM, 0);

// 设置服务器地址和端口

sockaddr\_in6 server\_addr{};

server\_addr.sin6\_family = AF\_INET6;

server\_addr.sin6\_port = htons(8888);

if (inet\_pton(AF\_INET6, "::1", &(server\_addr.sin6\_addr)) <= 0) {

std::cerr << "Invalid address/Address not supported!" << std::endl;

return -1;

}

// 连接到服务器

if (connect(client\_socket, (sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

std::cerr << "Failed to connect to server!" << std::endl;

return -1;

}

std::cout << "已连接到服务器，可以开始游戏！" << std::endl;

// 创建棋盘对象

Board board;

// 创建 TCP 连接对象

TCPConnection connection(client\_socket);

// 游戏逻辑

bool gameOver = false;

while (!gameOver) {

// 等待服务器发送数据

int choice = 0;

std::cout << "请输入你的选择：1.开始/继续游戏 2.退出" << std::endl;

std::cin >> choice;

std::cin.ignore(); // 清除输入缓冲区中的换行符

if (choice == 1) {

std::string receiveData = connection.receive();

if (receiveData.empty()) {

std::cerr << "Failed to receive data from server!" << std::endl;

break;

}

board = Board();

while (true) {

// 显示棋盘

board.print();

// 处理服务器发送的数据等

// 处理对手落子信息

int x, y;

std::istringstream iss(receiveData);

if (!(iss >> x >> y)) {

std::cerr << "Invalid move received from the opponent!" << std::endl;

break;

}

std::cout << "对手落子：" << x << "," << y << std::endl;

// 更新棋盘状态

board.makeMove(x - 1, y - 1, 'X');

board.print();

// 等待玩家落子

std::cout << "请玩家输入落子坐标（行 列）：";

std::cin >> x >> y;

// 检查落子是否有效

if (!board.isValidMove(x - 1, y - 1)) {

std::cout << "无效的落子位置，请重新输入！" << std::endl;

continue;

}

// 更新棋盘状态

board.makeMove(x - 1, y - 1, 'O');

board.print();

// 发送落子信息给对手

std::string move = std::to\_string(x) + " " + std::to\_string(y);

try {

connection.send(move);

std::cout << "已发送落子信息：" << move << std::endl;

} catch (const std::exception& e) {

std::cerr << "Failed to send move information to the server: " << e.what() << std::endl;

break;

}

// 检查胜负

if (board.checkWin(x - 1, y - 1)) {

std::cout << "恭喜你获胜了！游戏结束。" << std::endl;

break;

}

// 判断平局

if (board.isFull()) {

std::cout << "平局！游戏结束。" << std::endl;

break;

}

// 等待对手落子

std::cout << "等待对手落子..." << std::endl;

std::string opponent\_move = connection.receive();

std::istringstream issOpponent(opponent\_move);

if (!(issOpponent >> x >> y)) {

std::cerr << "Invalid move received from the opponent!" << std::endl;

break;

}

// 更新棋盘状态

board.makeMove(x - 1, y - 1, 'X');

board.print();

// 检查胜负

if (board.checkWin(x - 1, y - 1)) {

std::cout << "对手获胜！游戏结束。" << std::endl;

break;

}

}

} else if (choice == 2) {

// 关闭连接

close(client\_socket);

return 0;

} else {

std::cout << "你的选择有误" << std::endl;

}

}

}