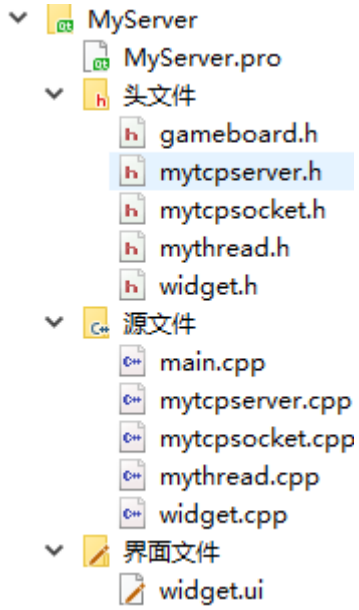
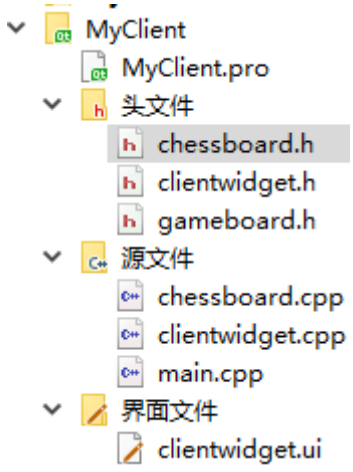
项目结构

服务器

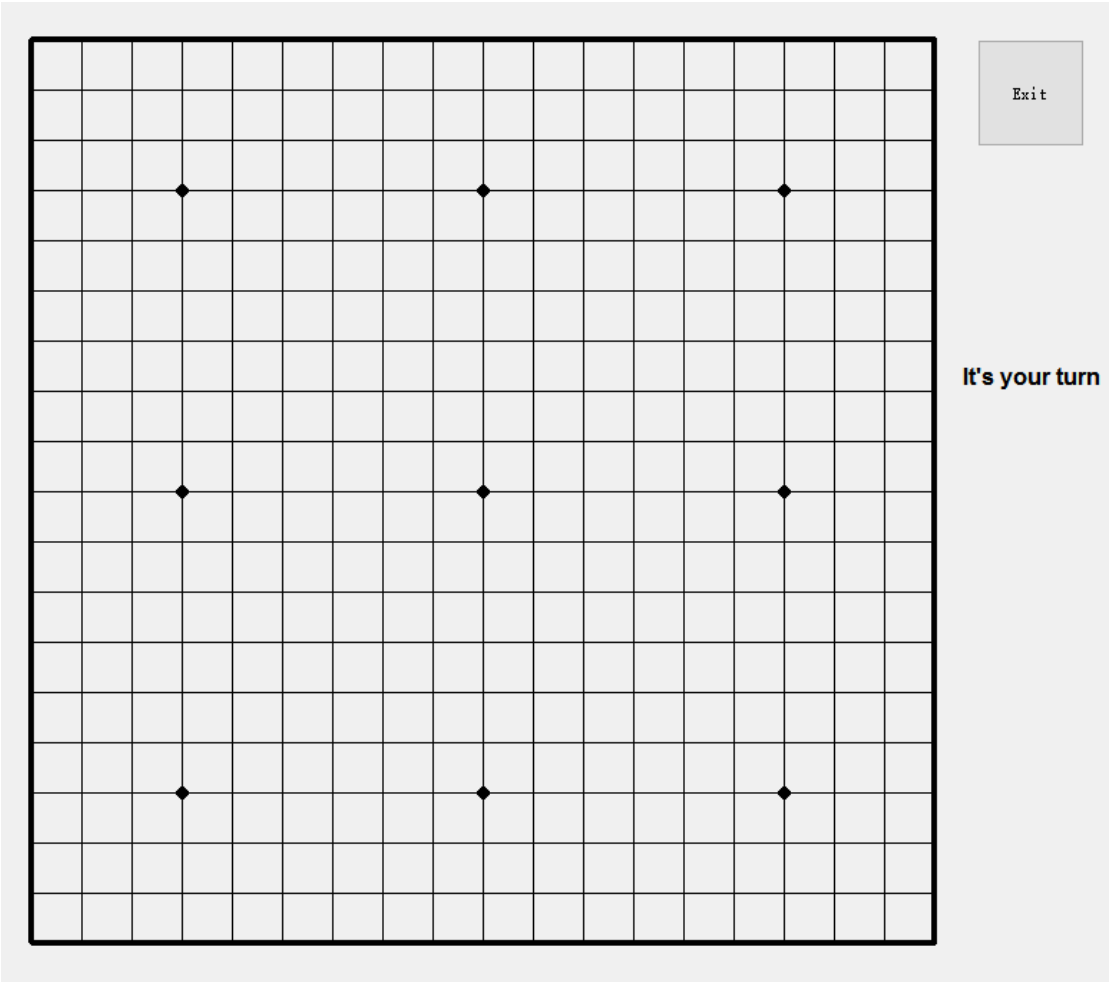


客户端

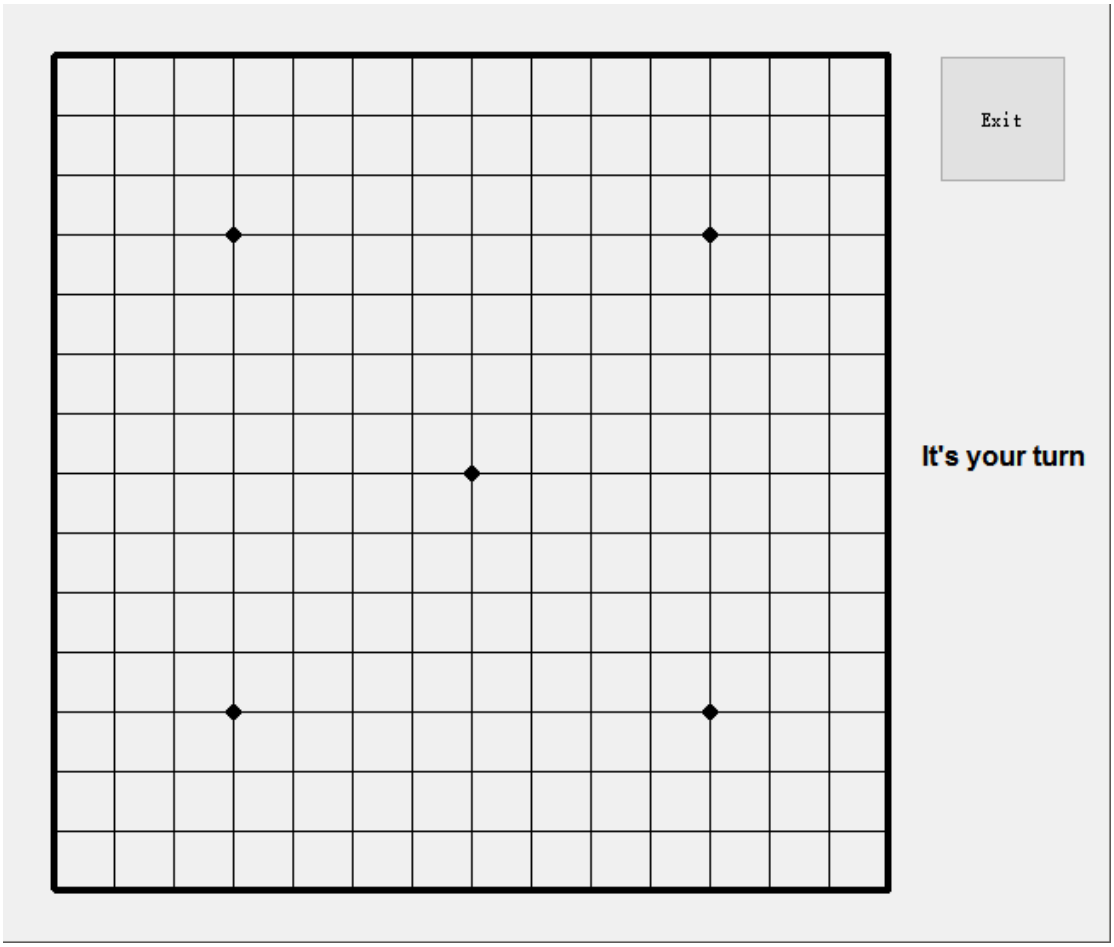


棋盘

放在客户端，棋盘通过 chessboard 类来实现。
将棋盘窗口的继承类提升为 chessboard，通过 paintevent（） 画棋盘



围棋棋盘



五子棋棋盘

链接

通过服务器端建立多线程进行多个客户端链接。

- 自定义进程类 mythread：定义信号和槽连接，进行读取和发送信息
- 自定义 tcpserver 类：在有客户端链接时调用 incomingConnection（）方法进行创建线程。每一个线程都可以独立发送和接收数据。

数据传输

通过 json 串来传送数据：

- 发送数据：

```
QJsonObject json;
json.insert("x", (int)x);
```

```
json.insert("y",(int)y);
json.insert("currentstone", QString(current_stone));
jsonArray.insert(0,json);

QJsonDocument document;
document.setArray(jsonArray);
str = document.toJson(QJsonDocument::Compact);
```

- 接收数据（解析 json）：

```
QJsonParseError json_error;
QJsonDocument parse_doucment = QJsonDocument::fromJson(byte_array,
&json_error);
if(json_error.error == QJsonParseError::NoError)
{
    if(parse_doucment.isArray())
    {…}
  }
```

算法

算法放在服务器端进行判断，返回数据给客户端

- 围棋吃子算法：
  利用队列进行广度优先搜索，通过查看当前棋子上下左右的状态，判断其是否有气。

```
bool go_bfs(int x,int y,StoneType current,std::queue<point> &save){
    //广度优先搜索，如果当前一片区域的棋子没有气，则 remove
    int dir[4][2]= {{-1,0},{1,0},{0,-1},{0,1}};//记录四个方向
    point temp;

    //把当前坐标加入队列
    temp.x=x,temp.y=y;
    q.push(temp);
    save.push(temp);
    while(!q.empty())
    {
        //获取队头坐标
        temp=q.front();
        q.pop();
        x=temp.x;
        y=temp.y;
        vis[x][y]=true;
        //对当前坐标的周围进行查看
```

```
        for(int i=0; i<4; i++)
        {
            int u,v;
            u=x+dir[i][0];
            v=y+dir[i][1];
            //坐标 u,v 没有棋子 代表有气存在 返回 true
            if(this->Go.checkStone(u,v)!=BLACK&&this->Go.checkStone(u,v)!=WHITE)
                return true;
            //否则 将坐标 u，v 加入队列
            else if(this->Go.checkStone(u,v)==current&&!vis[u][v])
            {
                temp.x=u;
                temp.y=v;
                q.push(temp);
                save.push(temp);
            }
        }//end for
    }
    return false;

}

void go_check(){
    int dir[4][2]= {{-1,0},{1,0},{0,-1},{0,1}};//记录四个方向
    int u,v;
    count=1;
    //初始化 vis
    for(int i=0;i<19;i++)
        for(int j=0;j<19;j++)
            vis[i][j]=false;

    std::queue<point> save;//保存要拿走的棋子

    //1.先看周围的对手的棋子是否有 没有气的区域，有的话则拿走
    for(int i=0; i<4; i++)
    {
        u=x+dir[i][0];
        v=y+dir[i][1];
        if(u<0||u>19||v<0||v>19)
            continue;

        //如果是没有访问过的对手的棋子，进行广搜

if(Go.checkStone(u,v)!=NONE&&this->Go.checkStone(x,y)!=this->Go.checkStone(u,v)
```

```
&&!vis[u][v]){

            if(go_bfs(u,v,this->Go.checkStone(u,v),save)){//当前区域有气，则把
save 和 q 清空
                while(!save.empty())
                    save.pop();
                while(!q.empty())
                    q.pop();
            }else
            {//当前区域没有气，则将 save 中的坐标加入 json 串中并拿走棋子

                while(!save.empty())
                {
                    point temp=save.front();
                    save.pop();
                    QJsonObject json;
                    json.insert("x", temp.x);
                    json.insert("y", temp.y);
                    jsonArray.insert(count++,json);
                    Go.removeStone(temp.x,temp.y);
                }
            }
        }
    }
   // qDebug()<<x<<y<<"after";
    //2.再查看当前棋子的区域有没有气，没有气则移除
    if(go_bfs(x,y,this->Go.checkStone(x,y),save)){
        while(!save.empty())
            save.pop();
        while(!q.empty())
            q.pop();
    }
    while(!save.empty())
    {
        point temp=save.front();
        save.pop();
        QJsonObject json;
        json.insert("x", temp.x);
        json.insert("y", temp.y);
        jsonArray.insert(count++,json);
        Go.removeStone(temp.x,temp.y);
    }
  //  qDebug()<<x<<y<<"end";
}
```

- 五子棋算法：
  检查当前棋子四个方向上是否连成五个棋子

```
bool five_check(){
    for (int i = 0; i < 5; i++)
    {
        //纵向
        if(y - i >= 0 &&
                y + 4 - i <= 15 &&
                Five.checkStone(x,y-i)==Five.checkStone(x,y + 1 - i)  &&
                Five.checkStone(x,y-i)==Five.checkStone(x,y + 2 - i) &&
                Five.checkStone(x,y-i)==Five.checkStone(x,y + 3 - i) &&
                Five.checkStone(x,y-i)==Five.checkStone(x,y + 4 - i))
            return true;

        //横行
        if(x - i >= 0 &&
                x + 4 - i <= 15 &&
                Five.checkStone(x - i,y)==Five.checkStone(x + 1 - i,y)  &&
                Five.checkStone(x - i,y)==Five.checkStone(x + 2 - i,y) &&
                Five.checkStone(x - i,y)==Five.checkStone(x + 3 - i,y) &&
                Five.checkStone(x - i,y)==Five.checkStone(x + 4 - i,y))
            return true;

        //左上到右下
        if(x - i >= 0 &&
                y - i >= 0 &&
                x + 4 - i <= 15 &&
                y + 4 - i <= 15 &&
                Five.checkStone(x - i,y - i)==Five.checkStone(x + 1 - i,y + 1 -
i)  &&
                Five.checkStone(x - i,y - i)==Five.checkStone(x + 2 - i,y + 2 -
i) &&
                Five.checkStone(x - i,y - i)==Five.checkStone(x + 3 - i,y + 3 -
i) &&
                Five.checkStone(x - i,y - i)==Five.checkStone(x + 4 - i,y + 4 -
i))
            return true;

        //从左下到右上
        if(x + i <= 0xF &&
                y - i >= 0 &&
                x - 4 + i >= 0 &&
```

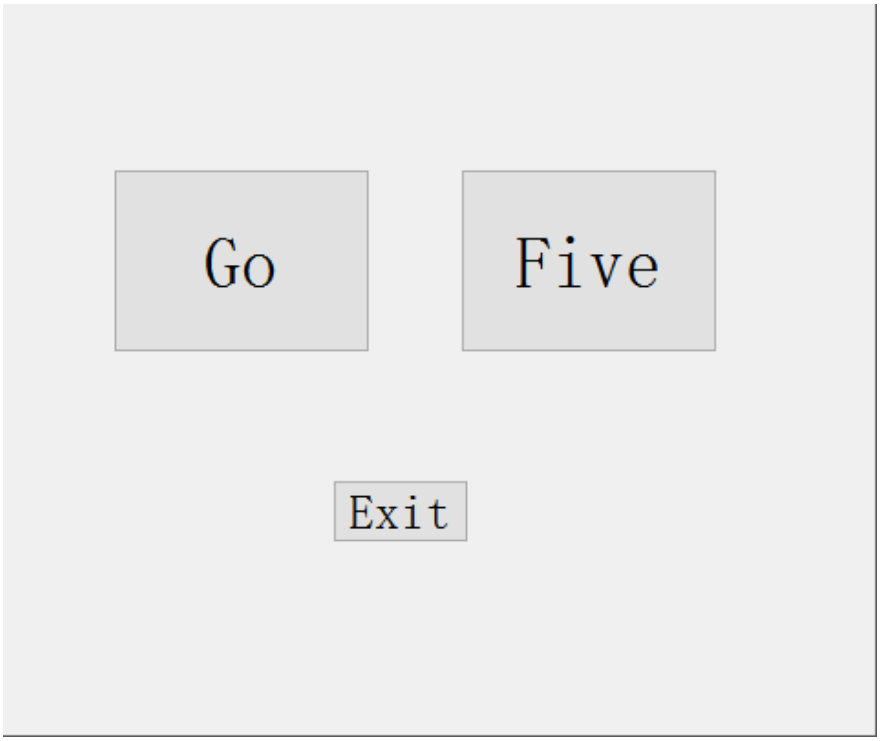
```
                y + 4 - i <= 15 &&
                Five.checkStone(x + i,y - i)==Five.checkStone(x - 1 + i,y + 1 -
i)  &&
                Five.checkStone(x + i,y - i)==Five.checkStone(x - 2 + i,y + 2 -
i) &&
                Five.checkStone(x + i,y - i)==Five.checkStone(x - 3 + i,y + 3 -
i) &&
                Five.checkStone(x + i,y - i)==Five.checkStone(x - 4 + i,y + 4 -
i))
            return true;
    }

    return false;

}
```

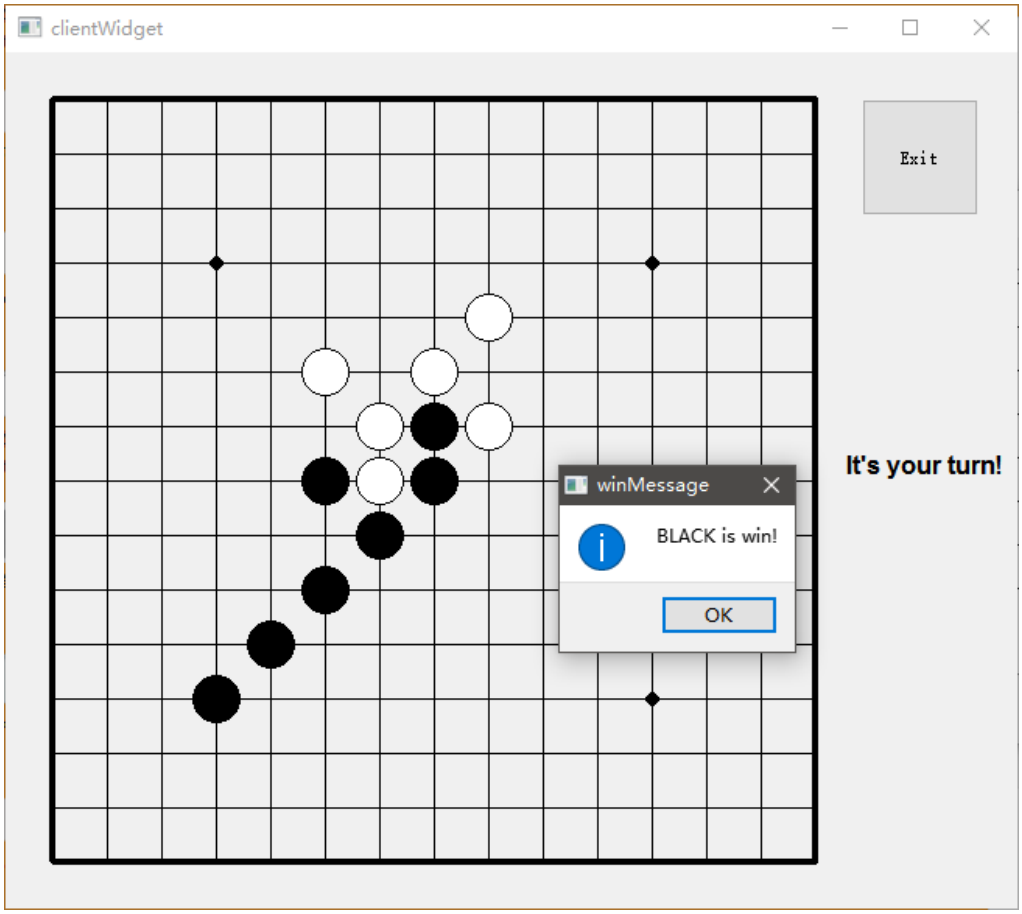
界面设计

客户端：
一共有三个界面，主界面，五子棋界面，围棋界面。通过 stackedWidget 控件进行切换。



主界面

当游戏结束时服务器发送结束信息，客户端弹出窗口。



服务器：
共一个界面，通过界面显示收发的信息。