Qt 第二周——国际跳棋

一、 工程简介

跳棋是世界上最古老，最普及的智力游戏之一，起源于古埃及、古罗马、古希腊等一些国家和地区。现代国际跳棋是在 12 世纪定型的。在 10*10 的棋盘内，黑白双方各持 20 子，通过斜向移动、跳吃等手段吃掉对方更多的棋子。最终吃掉对方所有棋子或者使对方任何棋子无法移动的一方获得游戏胜利。

这个工程实现了标准的国际跳棋图形程序，能在两台电脑上通过 TCP/IP 网络协议实现远程对战。支持认输与求和功能，并且加入了音效。

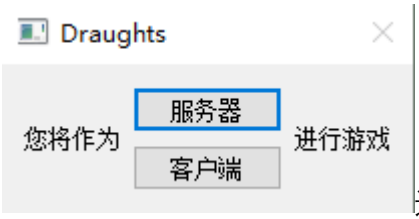
二、 程序结构

这个网络国际跳棋对战程序一共分为三个部分，首先是界面类，包括了主界面 draughts、服务器客户端选择界面和连接界面；其次是连接类，包括了客户端使用的 linkingclient 类和服务器使用的 linking 类，使用 tcpsocket 进行连接；最后是主画布与游戏逻辑结合在一起的 pad 类，生成的画布嵌入 draughts 类进行显示。

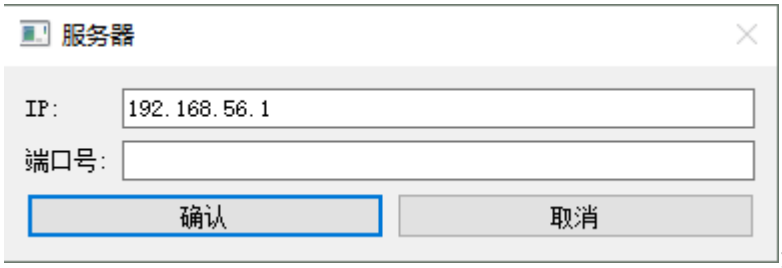
当打开程序时，首先跳出选择成为服务器或是客户端（两者是同一个程序），若成为服务器则显示本地 ip，输入自定的端口号后等待连接；成为客户端则可以自行输入 ip 和端口号进行连接。连接成功后则出现主界面开始游戏。

三、 代码实现

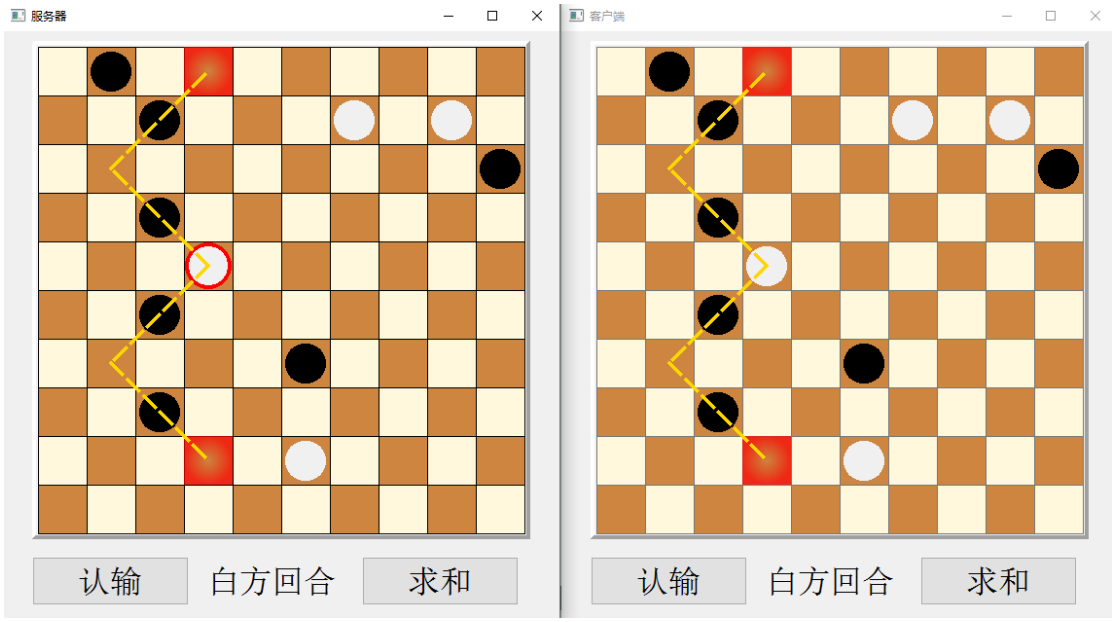
1、 界面



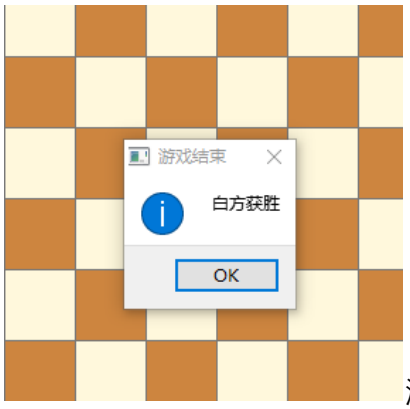
选择界面



连接界面，点击确认后将两个 lineedit 中的信息作为参数创建 tcpsocket 并连接。



进入游戏后，只有在玩家回合内，画布才会对鼠标点击进行响应。当玩家未点击时，显示所有可走路径。当玩家点击特定棋子时，只显示该棋子的可走路径。当选择终点后，棋子跳过被吃子到达终点。



游戏结束时显示提示信息。

2、 画布的绘制

我创建了 posi 类来转换鼠标点击和格子坐标。当点击发生时，我首先进行判断，是否点空、是否点击在可以移动的棋子上、是否点击在终点，并依据判断的结果绘制。绘制时首先绘制背景（因为背景不变），然后根据棋子信息绘制棋子。当点击在可以移动的棋子上，我判断点击位置并在该棋子外绘制一圈红圈表示选中。接着绘制路径，根据路径信息绘制可选的终点。若有吃子，根据路径信息则用黄色虚线依次连接落子位置。

```
QPoint pnt=ev->pos();
posi *tem=new posi(pnt);
bool ista=false;
qDebug()<<tem->l<<tem->x;
if(!pa.isEmpty()){//判断是否正确点击在终点位置 ...}
qDebug()<<"ista"<<ista;
if(tem->x==5||chess[tem->l][tem->x]==0){//选中空位
    if(!st||(st&&!ista)){//点空 ...
    }else{//点对 ...}
}
if(chess[tem->l][tem->x]==((turn==1)?2:1)||chess[tem->l][tem->x]==((turn==1)?2:1)+4){//选中己方
if(chess[tem->l][tem->x]==turn||chess[tem->l][tem->x]==turn+4){//选中对方 ...}
```

3、 信息的储存

我使用了一个二维正式数组 chess 储存棋子信息，0 为空，1 为白，2 为黑等等，绘制时依照 chess 在该点的信息绘制棋子。我创建了一个 path 类储存一条路径，其中包含四个 QList<int>，分别储存依次走过的坐标（包括起点终点）和吃子的坐标，还有一个数据成员 n 储存吃子数。我用 QList<path> allpa 储存所以可走的的路径信息，用 QList<path> pa 储存当前可选的路径，并依据 pa 在画布上绘制路径。绘制时，我用两个 posi 类 be、ta 表示选择的起终点信息（因为不存在两条路径有相同的起终点），依此决定当前用户选择的路径，储存为 cpath。最后我使用 act()函数，

根据 cpath 绘制走棋的（伪）动画效果，利用 QEventLoop 在每次落子后暂停 0.5 秒，把吃过的子变为灰色。最后再统一消除吃过的子。

4、 路径搜索

在每一步行动结束后，我会重新搜索路径并将结果储存在 allpa 中。所有搜索都使用当前棋盘的 chess 信息。我使用 search(), trans()，nextmove()三个函数进行 DFS 搜索。首先在 search 中我循环对每个当前回合方的棋子进行 trans 搜索，返回一个 QList<path>，最后把所以搜索结果压成一个 QList<path>返回。在 trans 中，我判断当前棋子是否为王，若是则调用 nextmove。Nextmove 返回该棋子在当前棋局下往左下、左上、右下、右上四个方向所有可走路径。然后我判断是否可走，若可走则暂时改变 chess 信息，然后以当前路径终点为起点深探。最后恢复 chess 信息，把所有可走信息返回，作为该棋子的可走信息返回。在搜寻中，如果出现某条路径吃子数比之前的都多，则清空已有路径。所以任何返回的 QList<path>中的路径都是最长路径。当进行完操作进入对方回合时，首先判断 chess 是否没有某方的棋子，然后判断 allpa 是否为空，是的话则我方胜利。

```
if(!king){
    //白:吃左下；黑吃左上
    if((_l+ud*2>=0)&&(_l+ud*2<=9)&&(_x-1>=0)&&chess[_l+ud][_x+left]==turn&&chess[_l+ud*2][_x-1
    }//白:吃右下；黑吃右上
    if((_l+ud*2>=0)&&(_l+ud*2<=9)&&(_x+1<=4)&&chess[_l+ud][_x+right]==turn&&chess[_l+ud*2][_x+
    }//白:吃左上；黑吃左上
    if((_l-ud*2>=0)&&(_l-ud*2<=9)&&(_x-1>=0)&&chess[_l-ud][_x+left]==turn&&chess[_l-ud*2][_x-1
    }//白:吃右上；黑吃右下
    if((_l-ud*2>=0)&&(_l-ud*2<=9)&&(_x+1<=4)&&chess[_l-ud][_x+right]==turn&&chess[_l-ud*2][_x+
    if(!em&&!change){ ...}
    if(em&&temp.isEmpty()){//如果不处于初次移动则不可不吃子移动，若仍未移动，说明无棋可吃 ...}
}else{
    for(int k=1;k<=4;k++){
        path tpath=nextmove(k,_l,_x);
        if(tpath.l.size()!=0){
            if(tpath.n==0&&!em)
                continue;
            qDebug()<<"&&&&&&&&"<<k;
            for(int i=0;i<tpath.l.size();i++){ ...}
        }
    }
    if(!em&&!change){ ...}
}
return temp;
```

5、 消息传递

因为我的服务器和客户端程序完全相同，所以对每一步搜索出的 allpa 应该完全一样。因此在玩家选择路径后，我只需要在 allpa 中寻找到玩家选择的路径编号，并将该编号通过 tcpsocket 传给对方即可。收到编号后，画布从 allpa 中读取该编号路径并依此操作。这样大大简化了消息传递的难度。若玩家认输、求和、胜利，则发送 1000 以上的信号。

6、 音效

使用了 Qsound::play(“wavpath”)函数。

四、 反思展望

这次大作业虽然基本完成，但是这个程序仍有许多可以完善的地方。首先是程序的 ui 不够美观，可以加入图片、按钮图标等来丰富程序。

我在基本完成程序，开始测试时发现我的棋盘和测试样例的黑白正好相反。因为若要修改代码量略大，我就在绘制时把所有黑棋画成白色，白

棋画成黑色，并在初始化的时候注意黑白相反。这样就让程序完全符合测试要求了。

在搜索路径时，我的搜索方式不够简洁，理论上可以用一个返回该路径吃子数整数的函数来进行 DFS，可惜没有尝试使用。

最后我在编写程序初期在连接类采用的是多线程方法。但是编写过程中发现 run 函数无法保持运行，阻塞有点难以执行，于是最后放弃了多线程，采用了单纯的组合模式。如果实现多线程信息收发，应该可以实现对局对战同时进行。