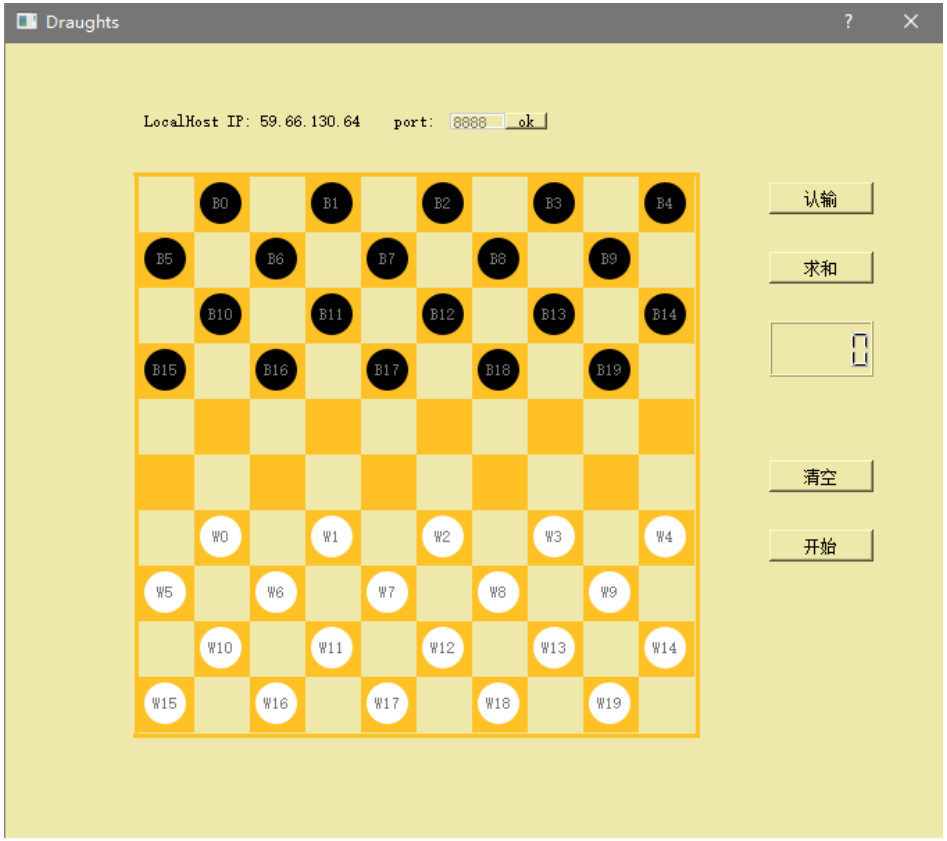
国际跳棋开发文档：

计 65 李文凯 2016011369

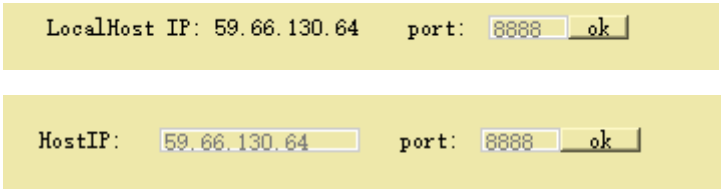


一、UI 界面部分：

此次大作业国际跳棋同样是基于 Qt 编程环境，有了上周数独游戏的铺垫，此次对 Qt 的使用熟练了很多。

UI 部分分为 IP 与端口设置模块，棋盘，以及功能模块。

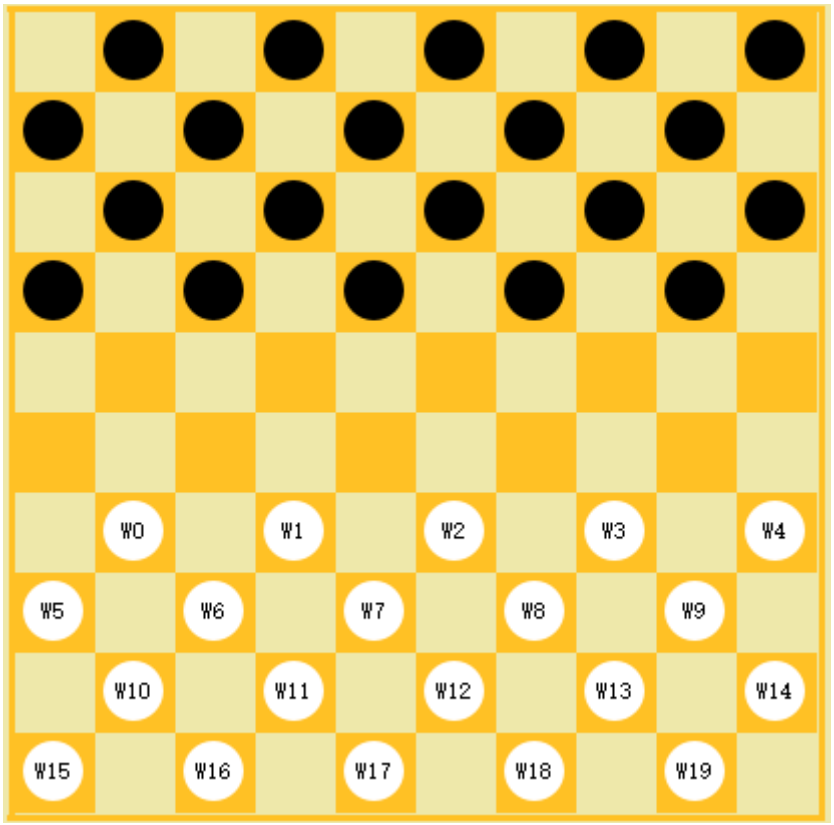
IP 与端口设置模块使用了 Qt 自带的控件 LineEdit，实现上较为简单，在此不做赘述。



功能模块，包含了认输，求和，求和计步器，清空并自定义棋盘，开始游戏五个功能部件。所使用的也都是 Qt 自带的控件，QPUshButton，QLCDnumber。

棋盘部分是 UI 部分的核心，其中棋子为继承 QPUshButton 的 ChessButton 实例，目的是重写 QPUshButton 的 protected 权限函数 FocusInEvent，来达到对棋子按钮选中的控制，棋子调用了 focusInEvent 后，发送信号给底层的 Dialog，有 Dialog 传给游戏逻辑的成员指针，对逻辑中的虚拟棋盘进行相关修改，结果逻辑中的算法，达到移动棋子，吃子的目的。

棋盘是重写了 Dialog 的 paintEvent 函数，做了这样几件事情，第一，画棋盘，第二，实现棋子选中时的路线高亮。其中的具体细节在接下来的算法逻辑部分详细介绍。



二、底层逻辑部分：

游戏逻辑部分用三维数组来存储虚拟棋盘，第一维记录该格子中棋子的颜色，第二维记录该棋子对应的编号，方便与 UI 互通，实现单个棋子的调度，减少每次落子的计算负担，同时设置了颜色锁，在玩家选取相同颜色棋子时，只有第一次对同色所有棋子进行寻路操作并记录，之后只需在对应容器中取出所需数据即可，提升了游戏的运行效率。

逻辑部分的重点是兵和王的寻路算法，下面一一进行介绍：

兵的寻路算法：

兵分为两种颜色，在不能吃子时两种颜色的行进方式不同，需要分开处理，向自己的斜前方行进一格即可，当能够吃子时，进行深度优先搜索，搜寻该兵能够通过跳吃走过的所有线路节点，对节点信心进行存储和标记。

兵的寻路算法步骤：1.搜索邻近的四个格子是否能够吃子，不能吃就按照颜色存储自己面前的两个落点，能吃转 2；

2.能够吃子，则进行深度优先搜索，四个方向依次检查，能够吃子则将落点作为一个新的根节点，进行相同的搜索操作，如果该根节点不能够继续跳吃，那么就将当前的线路存储。同时算法中包含回溯，对所有的可能进行尝试，确保了不遗漏任何一种走法。

3.以上算法对所有的同色棋子进行计算，最后容器中对应着棋子编号和该棋子能够到达的终点和对应路径，对容器进行处理，筛掉吃子数少的算法，一方的算法到此算是落下帷幕。

王的寻路算法：兵的行进只有向前一格，或跳吃，每次走至多向前两行，与兵不同，王的走法自由度更高，因此算法与兵比较起来也稍有不同。

王的寻路算法步骤：1.搜索四个方向，每个方向逐格进行判断，不能吃就将四个方向的所有可到位置进行存储，能吃则转 2；

2.对每个方向进行吃子检查，如果碰到能够吃掉的棋子，对跳过棋子之后且未碰到障碍的连续空格作为新的根节点，进行下一次 DFS；

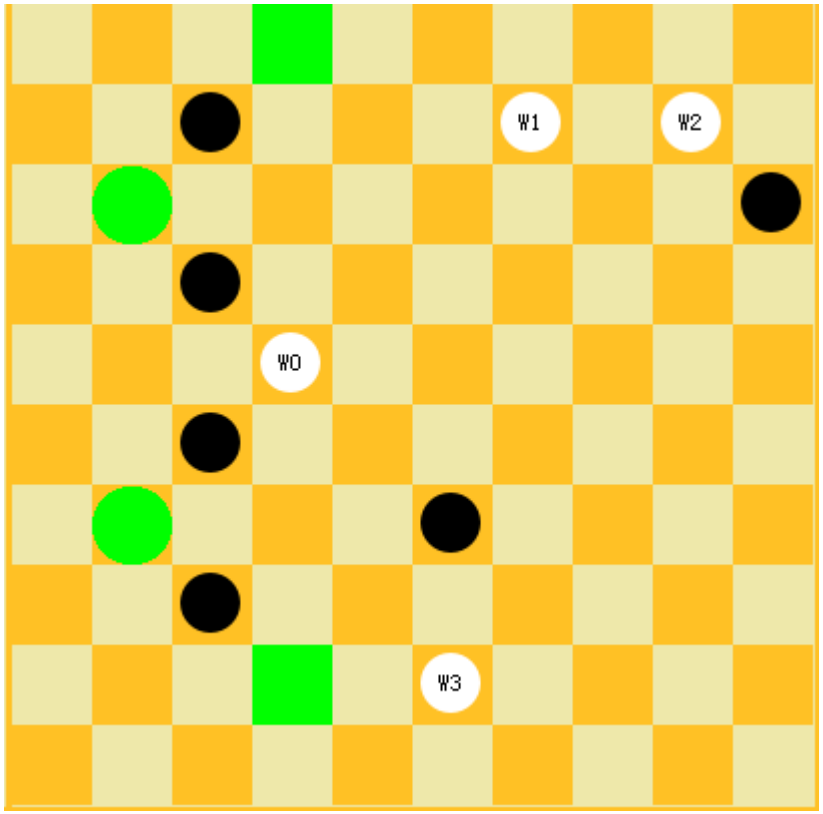
3.跳吃后如果新的根节点不能吃子，就将该节点作为该路径的终点存储；

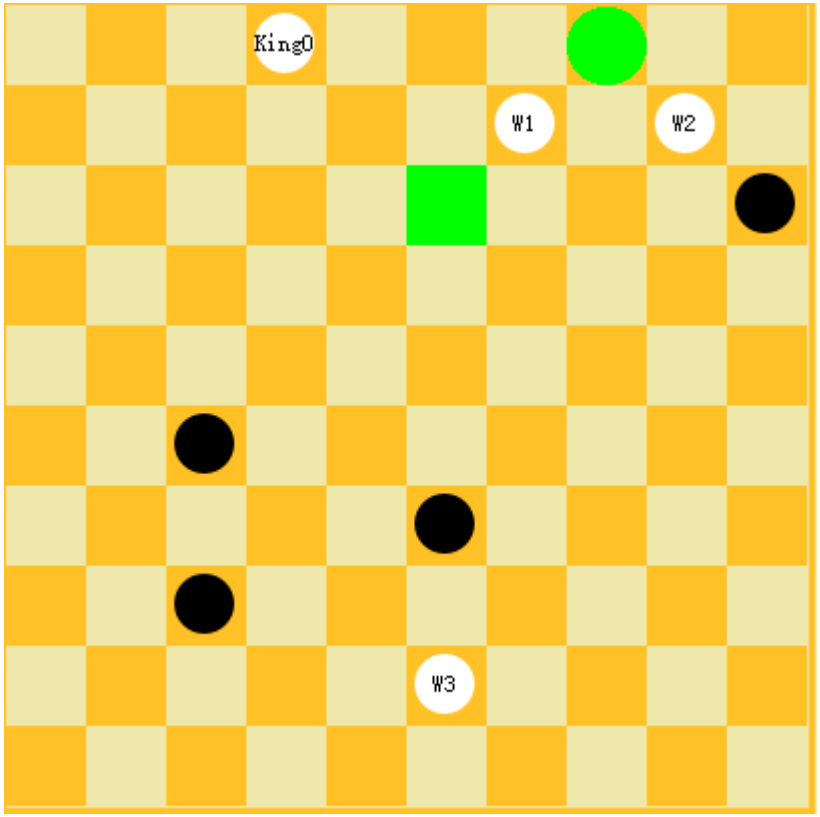
- 4.对容器内的数据进行筛选，除去吃子数少的，算法执行完毕。

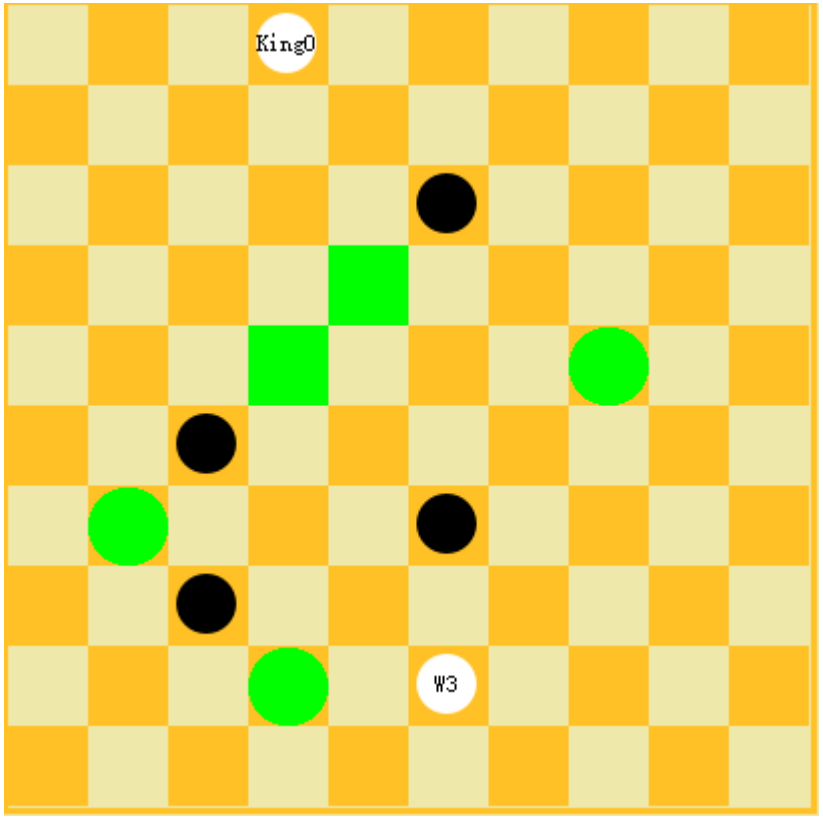
下面附上 DFS 代码以及相关寻路效果图：

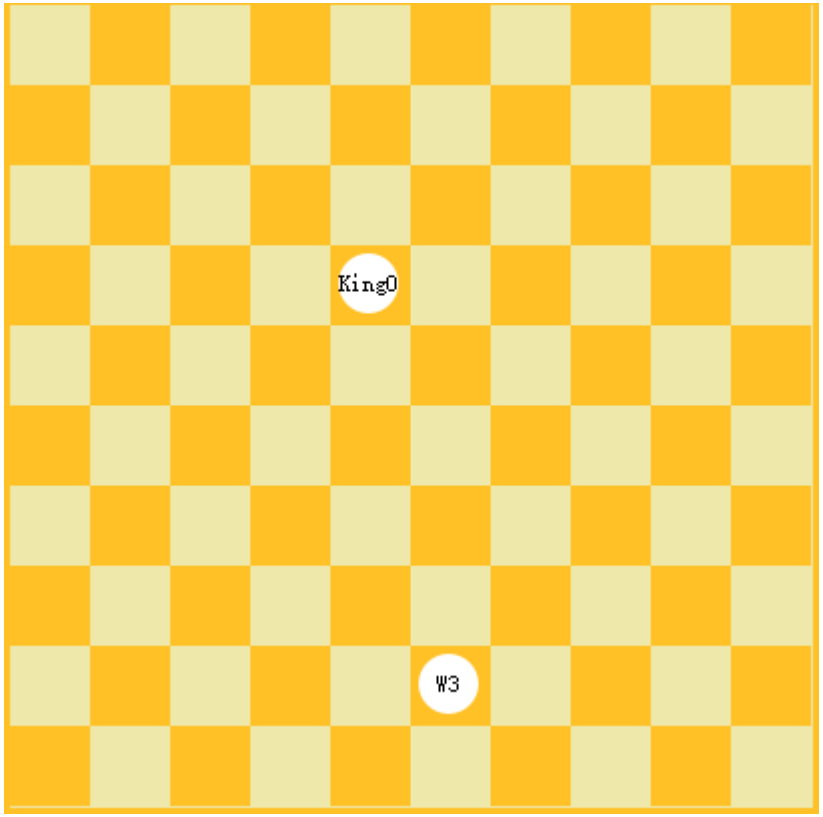
```
void DraughtsBase::dfs(int x, int y, int color,int num)//这个dfs只进行能够吃子的判定，先写兵的走法。complete
{
    int xx,yy,xxx,yyy;
    int i=0;
    int count=0;
    for(i=0;i<4;i++)
    {
        qDebug()<<"第 "<<i<<" 方向";

        xx=x+dx[i];//需要跳过的格子
        yy=y+dy[i];

        xxx=xx+dx[i];//需要到位的格子
        yyy=yy+dy[i];

        //越界就尝试下一个走法
        if(xxx<0||xxx>9)continue;
        if(yyy<0||yyy>9)continue;

        if(a[xxx][yyy][0]!=-1&&!(a[xxx][yyy][1]==num&&a[xxx][yyy][0]==color))continue;//wu chu luo jiao
```

```
        if(a[xx][yy][0]==-1)continue;//该方向无棋可吃

        if(a[xx][yy][0]==color)continue;//同色，尝试下一个

        //接下来就代表可以跳吃
        if(flag[xx][yy]!=1)
        {
            count++;
            flag[xx][yy]=1;//gai qizi yi bei tiaoguo
            //eatvec.push_back(a[xx][yy][1]);//记录吃子
            RoadsNode newNode;
            newNode.owner=num;
            newNode.xpos=xxx;
            newNode.ypos=yyy;
            roads.push_back(newNode);//记录所经位置
            qDebug()<<"new Node:"<<xxx<<' '<<yyy;
            dfs(xxx,yyy,color,num);//由此继续深探


            flag[xx][yy]=0;
            //eatvec.pop_back();
            roads.pop_back();
        }

    }
    qDebug()<<"i: "<<i<<" count: "<<count;
    if(i==4&&count==0)
    {
        qDebug()<<"线路查找完成";
        allroads.push_back(roads);

    }
}
```


W1
W2
W0
W3


King0
W1
W2
W3


King0
W3


King0
W3

三、网络编程模块：

网络编程在此次编程大作业中是新的内容，经过学习后对 TCP 和 UDP 两种传输协议有了更深层次的了解。

TCP 像是在 server 和 client 之间建设了一个通道，连接成功后即可保持彼此通信的状态，而 UDP 则是每次将数据包直接丢给制定的 IP 地址和端口，执行完这一步操作后就对结果不予理睬。TCP 注重安全性和数据的完整，而 UDP 更着眼于即时的效率问题。

按照功能要求，用户在 lineEdit 中填入 IP 和端口号，实现 server 与 client 的互联，附上实现代码：

```
void Draughts::initServer()
{
    int port=ui->lineEdit->text().toInt();
    qDebug()<<"port: "<<port;
    if(port!=0)
    {
    this->listenSocket =new QTcpServer();
    this->listenSocket->listen(QHostAddress::Any,port);
    connect(this->listenSocket,SIGNAL(newConnection()),this,SLOT(acceptConnection()));
    ui->lineEdit->setEnabled(false);
    }
}
```

```
void Draughts::connectHost()
{
    if(IP!=""&&port!=0)
    {
    this->readWriteSocket = new QTcpSocket();
    this->readWriteSocket->connectToHost(QHostAddress(IP),port);
    QObject::connect(this->readWriteSocket,SIGNAL(readyRead()),this,SLOT(recvMessage()));

    ui->IPselectEdit->setEnabled(false);
    ui->portSelectEdit->setEnabled(false);
    qDebug()<<"connected!";
    }
}
```

四、编程过程及经验收获：

在周四晚上的时候，游戏逻辑和界面就已经基本实现了，但是那个时候的方案在每次点击按钮是都会对同色所有棋子进行寻路，对于 10*10 的棋盘可能运算量不大，大概是强迫症驱使，还是对整体的架构进行了改进，结果一改就是一整天，但是最后还是取得了自己想要的效果，由此心中还是充满了成就感。

改进后的版本只有在第一次选择了该色的棋子后，对该色所有棋子进行一次寻路，之后点击同色其他棋子只是将数据取出，并对落点进行高亮而已，可以使省去了很多重复无意义的计算工作，减轻了主线程的计算负担，最重要是心里觉得舒服。

在起初编写代码时，我对两种颜色的棋子进行了区分，这样导致在其他模块的编写过程中出现了很多判断和冗余的代码，后来仔细一想，是因为颜色参数取的不好，如果将黑白对应正 1 和负 1，那么每次传递颜色参数时，在函数体中直接取反，可以省去很多的 if 判断语句，在棋子的存储中也应该使用二维数组，第一位表示颜色，第二位记录编号，这样会大大提升代码的复用率，减少冗余代码。

两周的 Qt 学习，着实掌握了先前不了解的知识和技能，虽然忙碌，但是收获满满。