个人信息及环境

个人信息

杨雅儒，计85，2017011071。

环境

QT Creator 4.9.1(Enterprise)，QT 5.12.4 (MinGW 7.3.0 64-bit)。

程序设计

液滴的处理

首先要定义一个数组 nowDrop[][]，表示某个格子当前的液滴标号（标号从1开始），为0表示没有液滴。再定义一个颜色QList<> dropColor，用于给出每个标号液滴的颜色。另外还要定义QMap<int,bool> histDrop[][]，用于标记每个格子被哪些液滴经过。

dropCnt 表示至今出现过的液滴种类数。

指令的处理

首先通过文件读入，认为每一行有一个指令。用一个结构体来表示指令，opt表示指令类型，arg[0]~arg[5]为6个参数。

opt值对应的指令为，Move为1，Split为2，Merge为3，Input为4，Output为5，而Mix拆分为若干Move进行处理。

具体参数如下：

```
args:            0  1  2  3  4  5
Move(opt==1):   x1,y1,x2,y2       (从1移动至2)
Split(opt==2):  x1,y1,x2,y2,x3,y3 (由1分成2、3)
Merge(opt==3):  x1,y1,x2,y2       (将1、2合并到中间位置)
Input(opt==4):  x1,y1             (输入到1位置)
Output(opt==5): x1,y1             (由1位置输出)
【新增虚拟opt==6表示分裂的第二阶段，opt==7表示合并的第二阶段】
```

注意在t时刻的操作，将在t+1时刻被观测到。

指令的读取

要完成的工作即：**读取文件并且将其中的字符串转化为对应指令存储下来。**

书写了函数parseFile用于解析文件，parseLine用于解析某一行。

parseLine实现如下：

```
int  MainWindow::parseLine(QString str){
    //失败返回-1，否则返回该条指令的最后执行时刻，注意执行时刻大于MAXTIME时同样返回-1
    Instruction inst;
    QStringList argList = str.split(',') ;
    int time=-1, len=argList.length();
    bool ok;
    QString tmp = argList.at(0);
    time = tmp.right(tmp.length()-tmp.lastIndexOf(' ')).toInt(&ok) ;
    if(!ok) return -1;
    int ret=time+1;

    if(str.left(str.indexOf(' '))=="Move"){
        inst.opt=1;
    } else if(str.left(str.indexOf(' '))=="Split"){
        inst.opt=2;
        ret++;
    } else if(str.left(str.indexOf(' '))=="Merge"){
        inst.opt=3;
        ret++;
    } else if(str.left(str.indexOf(' '))=="Input"){
        inst.opt=4;
    } else if(str.left(str.indexOf(' '))=="Output"){
        inst.opt=5;
    } else if(str.left(str.indexOf(' '))=="Mix"){
        if(len<=1 || (len&1)==0) return -1;
        QPoint last;
        for(int i=1;i<len;i+=2){
            QPoint now;
            QString s = argList.at(i).simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            now.setX(s.toInt(&ok));
            if(!ok) return -1;

            s = argList.at(i+1).simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            now.setY(s.toInt(&ok));
            if(!ok) return -1;

            if(i!=1){
                inst.opt=1;
                inst.arg[0]=last.x();
                inst.arg[1]=last.y();
                inst.arg[2]=now.x();
                inst.arg[3]=now.y();
                if(time+i/2>MAXTIME) return -1;
                instructions[time+i/2-1].append(inst) ;
                lastVis[now.x()][now.y()]=std::max(lastVis[now.x()][now.y()],time+i/2);
            }
            last=now;
        }
        ret=ret+(len-2)/2;
        inst.opt=0;
    }
```

```
    if(inst.opt!=0){
        if((inst.opt==1 && len!=5) || (inst.opt==2 && len!=7) || (inst.opt==3 && len!=5)
                || (inst.opt==4 && len!=3) || (inst.opt==5 && len!=3))
            return -1; //操作数个数不对应
        for(int i=1;i<len;++i){
            QString s = argList.at(i);
            s = s.simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            inst.arg[i-1] = s.toInt(&ok);
            if(!ok) return -1;
        }
        if(time>MAXTIME) return -1;
        instructions[time].append(inst) ;

        //处理lastVis
        if(inst.opt==1){
            lastVis[inst.arg[2]][inst.arg[3]] = std::max(lastVis[inst.arg[2]][inst.arg[3]],
time+1);
        } else if(inst.opt==2){
            lastVis[inst.arg[0]][inst.arg[1]] = std::max(lastVis[inst.arg[0]][inst.arg[1]],
time+1);
            lastVis[inst.arg[2]][inst.arg[3]] = std::max(lastVis[inst.arg[2]][inst.arg[3]],
time+2);
            lastVis[inst.arg[4]][inst.arg[5]] = std::max(lastVis[inst.arg[4]][inst.arg[5]],
time+2);
        } else if(inst.opt==3){
            lastVis[inst.arg[0]][inst.arg[1]] = std::max(lastVis[inst.arg[0]][inst.arg[1]],
time+1);
            lastVis[inst.arg[2]][inst.arg[3]] = std::max(lastVis[inst.arg[2]][inst.arg[3]],
time+1);
            int x=(inst.arg[0]+inst.arg[2])/2, y=(inst.arg[1]+inst.arg[3])/2;
            lastVis[x][y] = std::max(lastVis[x][y], time+2);
        } else if(inst.opt==4){
            lastVis[inst.arg[0]][inst.arg[1]] = std::max(lastVis[inst.arg[0]][inst.arg[1]],
time+1);
        }
    }
    return ret;
}
```

parseFile实现如下：

```
int MainWindow::parseFile(){
    QFile file(filePath);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QMessageBox::critical(this, "错误", "打开文件失败");
        return -1;
    }
    //因为重新打开了文件，记得初始化!!!!
    memset(lastVis,0,sizeof(lastVis));

    for(int i=0;i<=timeLim;++i)
```

```
        instructions[i].clear();
    timeLim=0;
    QTextStream in(&file);
    QString line = in.readLine(); ;
    int cnt=0;
    while(!line.isNull()){
        ++cnt;
        if(line==""){
            line = in.readLine();
            continue;
        }
        int ret = parseLine(line) ;
        if(ret==-1){
            QMessageBox::critical(this, "错误", QString("第%1行指令出错").arg(cnt));
            return -1;
        }
        if(ret>timeLim) timeLim=ret;
        line = in.readLine();
    }
    file.close();
    return 0;
}
```

执行指令和撤销指令

按照类似栈的思路来做，如何一步一步执行的，就如何一步一步逆过来撤销。所以需要规定清楚顺序，例如在Split时严格按照指令顺序先生成x2,y2再生成x3,y3。

另外需要存储一个disapDropStack表示消失了的液滴的栈，执行指令的时候液滴消失则压栈，撤销指令的时候进行弹栈。

Move

很容易实现，修改位置以及histDrop即可，如下：

```
void MainWindow::instMove(int x1, int y1,int x2, int y2, bool rev){
    //从x1,y1移动到x2,y2, rev为true时表明撤销移动
    playMoveSound();
    if(!rev) {
        int drop=nowDrop[x1][y1];
        nowDrop[x2][y2] = drop ;
        nowDrop[x1][y1] = 0;
        histDrop[x2][y2][drop] = histDrop[x2][y2][drop]+1;
    }
    else {
        int drop=nowDrop[x2][y2];
        histDrop[x2][y2][drop] = histDrop[x2][y2][drop]-1;
        nowDrop[x1][y1] = drop;
        nowDrop[x2][y2] = 0;
    }
}
```

Split和Merge

这两个指令跨了两个时刻，所以需要小心处理。

加入两个数组：bool notAlone[][] 来标记不单独画出某个格子的水滴；QPoint midState[][] 来标记中间结点，第一个参数为1表示正在分裂，为2表示正在合并，第二个参数为0表示非中间结点，为1表示水平中间结点，2表示垂直中间结点。

处于中间态时，在两侧放上nowDrop，但是设定notAlone[][]来不单独画出这两个水滴，另外在中间加上midState[][]状态。

Split的两步如下：

```
void MainWindow::instSplit1(int x1, int y1,int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1分裂到x2,y2和x3,y3，rev为true时表明撤销分裂的首步
    playSplit1Sound();
    if(!rev){
        nowDrop[x2][y2] = ++dropCnt ;
        nowDrop[x3][y3] = ++dropCnt ;
        histDrop[x2][y2][nowDrop[x2][y2]] = histDrop[x2][y2][nowDrop[x2][y2]]+1;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]+1;
        notAlone[x2][y2] = notAlone[x3][y3] = true;
        midState[x1][y1] = QPoint(1, getMidState(x1,y1,x2,y2));
    } else{
        midState[x1][y1] = QPoint(0,0);
        notAlone[x2][y2] = notAlone[x3][y3] = false;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]-1;
        histDrop[x2][y2][nowDrop[x2][y2]] = histDrop[x2][y2][nowDrop[x2][y2]]-1;
        nowDrop[x2][y2] = nowDrop[x3][y3] = 0;
        dropCnt-=2;
    }
}

void MainWindow::instSplit2(int x1, int y1,int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1分裂到x2,y2和x3,y3，rev为true时表明撤销分裂的第二步
    playSplit2Sound();
    if(!rev){
        disapDropStack.push(nowDrop[x1][y1]);
        nowDrop[x1][y1]=0;
        notAlone[x2][y2]=notAlone[x3][y3]=false;
        midState[x1][y1]=QPoint(0,0);
    } else {
        midState[x1][y1]=QPoint(1,getMidState(x1,y1,x2,y2));
        notAlone[x2][y2]=notAlone[x3][y3]=true;
        nowDrop[x1][y1]=disapDropStack.top();
        disapDropStack.pop();
    }
}
```

Merge操作的两步如下：

```
void MainWindow::instMerge1(int x1, int y1, int x2, int y2, int x3, int y3, bool rev){
```

```
    //从x1,y1和x2,y2合并到x3,y3，rev为true是表示撤销合并的首步
    if(!rev){
        nowDrop[x3][y3] = ++dropCnt ;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]+1;
        notAlone[x1][y1] = notAlone[x2][y2] = true;
        midState[x3][y3] = QPoint(2,getMidState(x2,y2,x3,y3));
    } else{
        midState[x3][y3] = QPoint(0,0);
        notAlone[x1][y1] = notAlone[x2][y2] = false;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]-1;
        nowDrop[x3][y3] = 0;
        dropCnt--;
    }
}

void MainWindow::instMerge2(int x1, int y1, int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1和x2,y2合并到x3,y3，rev为true是表示撤销合并的第二步
    playMergeSound();
    if(!rev){
        disapDropStack.push(nowDrop[x1][y1]);
        disapDropStack.push(nowDrop[x2][y2]);
        nowDrop[x1][y1]=nowDrop[x2][y2]=0;
        notAlone[x1][y1]=notAlone[x2][y2]=false;
        midState[x3][y3]=QPoint(0,0);
    } else {
        midState[x3][y3]=QPoint(2,getMidState(x2,y2,x3,y3));
        notAlone[x1][y1]=notAlone[x2][y2]=true;
        nowDrop[x2][y2]=disapDropStack.top();
        disapDropStack.pop();
        nowDrop[x1][y1]=disapDropStack.top();
        disapDropStack.pop();
    }
}
```

Input 和 Output

这两个操作也很容易，如下：

```
int MainWindow::instInput(int x1, int y1, bool rev){
    //输入到x1,y1，rev为true表示撤销输入
    if(inPortList.indexOf(QPoint(x1,y1))==-1)
        return -1;
    if(!rev){
        nowDrop[x1][y1] = ++dropCnt;
        histDrop[x1][y1][nowDrop[x1][y1]] = histDrop[x1][y1][nowDrop[x1][y1]]+1;
    } else {
        histDrop[x1][y1][nowDrop[x1][y1]] = histDrop[x1][y1][nowDrop[x1][y1]]-1;
        nowDrop[x1][y1] = 0;
        --dropCnt;
    }
    return 0;

}
```

```
int MainWindow::instOutput(int x1, int y1, bool rev){
    //从x1,y1输出，rev为true表示撤销输出
    if(outPortList.indexOf(QPoint(x1,y1))==-1)
        return -1;
    if(!rev){
        disapDropStack.push(nowDrop[x1][y1]);
        nowDrop[x1][y1] = 0;
    } else{
        nowDrop[x1][y1] = disapDropStack.top();
        disapDropStack.pop();
    }
    return 0;
}
```

约束的处理

做指令前先存下每个格子对应的液滴编号，在做完一个时刻的指令后，直接枚举每个格子，通过判断其周围八个格子的状态来进行判断即可。**只是需要注意，处于中间态的中间点不需要用来做判断**，这是因为这个中间点的周围八个点被其两侧点的周围八个点完全包含，所以并不需要判断，而若判断的话可能会与其两侧点被误判成不满足约束。

这部分代码如下(其中tmpDrop为上一时刻时每个格子的液滴编号, 抛出异常值3表示不满足静态约束，4表示不满足动态约束):

```
for(int x1=1;x1<=col;++x1){
    for(int y1=1;y1<=row;++y1){
        if(nowDrop[x1][y1] && !midState[x1][y1].x()){
            for(int d=0;d<BANDIRNUM;++d){
                int x2=x1+dir[d][0], y2=y1+dir[d][1];
                if(x2<1||y2<1||x2>col||y2>row||midState[x2][y2].x()) continue;
                if(nowDrop[x2][y2] && nowDrop[x2][y2]!=nowDrop[x1][y1]) {
                    throw 3;
                }
                if(tmpDrop[x2][y2] && tmpDrop[x2][y2]!=nowDrop[x1][y1] && !tmpMidState[x2]
[y2].x() && tmpMidState[x1][y1].y()!=d/2+1) {
                    throw 4;
                }
            }
        }
    }
}
```

需求七

要求加入清洗功能，并可以“时停”，注意支持用户右键设定障碍。

算法设计

在开启清洗功能之后，希望**尽可能不出现污染**，为了达成这个目的，且考虑到用户会设定障碍，那么每次出现“必要”清理的格子时便尽可能早地清理。

对于**必要清洗格子**定义如下：**这个格子已经被污染了，并且这个格子最靠后的被经过的时刻大于当前时刻（也就是之后还会被液滴经过）**。

另外若一个时刻有多个必要且可清理到的格子，考虑清洁液滴从起点开始，求最短路得到最靠近的必要清理格子及这条路径，然后再从这个格子出发继续BFS。直到没有找到必要清理格子，则通过最短路走到出口。

而由于这里移动一次的代价为相等的定值1，所以最短路可以直接用BFS来方便地实现。

算法实现

变量定义

定义 QQueue<QPoint> washPath 用于表示当前的清洗路径；定义 washFlag[][]，表明这个格子在当前的washPath下是否被经过；lastVis[][] 表示每个格子最后被液滴访问的时刻。剩下的一些变量便是bfs中比较标准的变量定义。

函数定义

实现函数 washAddPath(QPoint s, QPoint t)，表明此时已经以s为起点BFS过了，这个函数中将s到t的路径加入到washPath，并且更新washFlag。实现如下：

```
void MainWindow::washAddPath(QPoint s, QPoint t){
    //已经以s为起点BFS过了，现在将s到t的路径加入到washPath
    if(s==t) return ;
    washAddPath(s, bfsPre[t.x()][t.y()]);
    washPath.append(t);
    washFlag[t.x()][t.y()]=true;
}
```

另外实现一个简单的函数 bool washCheckNeed(QPoint s) 用于判断 s 是否为还没有被加入washPath的必要清洗格子，实现如下：

```
bool MainWindow::washCheckNeed(QPoint s){
    //判断s是否为没有被加入washPath的必要清洗格子
    int x=s.x(), y=s.y();
    return lastVis[x][y]>timeNow && !washFlag[x][y] && histDrop[x][y].size()>0 ;
}
```

然后实现函数 QPoint washBFS(QPoint s, bool *ok)，表示从 s 点开始BFS，返回最靠近的必要清洗格子或是(col,row)，注意若不能到达(col,row)则返回QPoint(-1,-1)；另外ok为true表示找到了必要清洗格子，这是为了**处理(col,row)恰好是必要清洗格子的情况**。在这个函数中还需要调用washAddPath函数将s和找到的点之间的路径加入washPath。实现如下：

```
QPoint MainWindow::washBFS(QPoint s, bool *ok){
    //从s点开始BFS，返回最靠近的必要清洗格子或是(col,row)，注意若不能到达(col,row)则返回QPoint(-1,-1)
    //ok为true表示找到了必要清洗格子（这是为了处理(col,row)恰好是必要清洗格子的情况)
    bfsQue.clear();
    bfsQue.push_back(s);
```

```
    memset(bfsDis,0,sizeof(bfsDis));
    bfsDis[s.x()][s.y()]=1;
    QPoint ret=QPoint(-1,-1);
    *ok = false;
    while(!bfsQue.empty()){
        QPoint a=bfsQue.first();
        bfsQue.pop_front();
        for(int i=0;i<MOVEDIRNUM;++i){
            int x=a.x()+dir[i][0], y=a.y()+dir[i][1];
            if(!washCheckPoint(QPoint(x,y))) continue ;
            if(!bfsDis[x][y]){
                bfsDis[x][y]=bfsDis[a.x()][a.y()]+1;
                bfsPre[x][y]=a;
                bfsQue.push_back(QPoint(x,y));
                if(washCheckNeed(QPoint(x,y)) && ret==QPoint(-1,-1)){
                    ret = QPoint(x,y);
                    *ok = true;
                }
            }
        }
    }
    if(bfsDis[col][row] && ret==QPoint(-1,-1)) ret = QPoint(col,row);
    if(ret!=QPoint(-1,-1)) washAddPath(s,ret);
    return ret;
}
```

最后在最外层实现bool wash()函数，用于实现单步下的清洗，返回true表示进行清洗。wash()函数中，首先初始化变量，然后通过一个while循环不停寻找最近的未被清洗的必要清洗格子，即调用washBFS，最后判断是否找到清洗路径，若是则返回true进行清洗，否则返回false。实现如下：

```
bool MainWindow::wash(){
    //用于做单步下的清洗，返回值表示这一步是否进行清洗
    washPath.clear();
    memset(washFlag,false,sizeof(washFlag));
    washPath.append(QPoint(1,1));
    if(washCheckPoint(QPoint(1,1))==false) return false;
    QPoint now=QPoint(1,1);
    bool ok=true;
    bool has=false;
    while(ok){
        now = washBFS(now, &ok);
        if(ok) has=true;
        if(now==QPoint(-1,-1)) return false;
    }
    return has;
}
```

加分项

为了确定一个合适的Input、Merge、Split、Mix顺序，考虑原顺序下既然能够执行，就暂时不改变这个顺序。

那么存下一个操作队列，每次仅考虑做这个队列的第一个操作。

对于Input，注意到题目要求只能从同样的输入端口进入（不过这个输入端口可以改变位置）。

端口的构造

最上面一排等间距摆放输入端口，最下面一排等间距摆放输出端口，最左边中间是清洗液滴输入端口，最右边中间是清洗液滴输出端口。

指令的定义

结构体 RouteInstruction

存储单步指令，用于预处理。

定义opt表示操作类型，QList<int> args为参数列表。

```
args:            0  1  2  3  4  5  6  ...
Move(opt==1):   x1,y1,x2,y2       (从1移动至2)
Split(opt==2):  x1,y1,x2,y2,x3,y3 (由1分成2、3)
Merge(opt==3):  x1,y1,x2,y2       (将1、2合并到中间位置)
Input(opt==4):  x1,y1             (输入到1位置)
Output(opt==5): x1,y1             (由1位置输出)
Mix(opt==6):    x1,y1,x2,y2,x3,y3,x4,...
```

RouteOperation

存储操作，opt同上（无Move），另外还有若干变量，定义如下：

```
struct RouteOperation{
    int opt, mixLen, pt1, pt2, pt3, oriTime ;
    //1:Move(无)  2:Split  3:Merge  4:Input  5:Output  6:Mix
    //oriTime为该操作原本的执行时间
    //若为Input\Output\Mix，则pt1为点编号
    //若为Merge，则pt1、pt2合并生成pt3；若为Split，则pt1分裂生成pt2、pt3。
    RouteOperation(int _opt=0, int _mixLen=0, int _pt1=0, int _pt2=0, int _pt3=0, int
_oriTime=0){
        opt=_opt; mixLen=_mixLen; pt1=_pt1; pt2=_pt2; pt3=_pt3; oriTime=_oriTime;
    }
};
```

操作的实现

定义格子的阻塞值为**该格子到最近端口的曼哈顿距离的相反数乘上常系数K。**

Input

对于操作队列的队首，如果是Input就直接尽可能Input进来，否则暂停操作。

Output

如果一个液滴已经做完了操作，则将其Output放入队列队首。

Output时，向其输出端口寻路，若不能到达，则暂停操作。

Merge

找到两个左右相隔一个位置的格子，记为格子A和格子B，**满足AB中间的格子下方无污渍，且靠左(相同列时靠下)的液滴走到格子A距离平方加靠右(相同列时靠上)的液滴走到格子B距离平方在开根之后再加上中间格子的阻塞值之后最小**，此时暂停操作。

Split

下述"无污渍"表示不存在污渍或污渍为该液滴自身产生。

则对能走到的格子寻路，且满足**该格子及其左、右、左下、右下均无污渍，且阻塞值最小**，进行分裂操作。

Mix

BFS后找到一个同样形状的2*k或k*2的矩形，且满足**均无污渍且阻塞值之和加上(当前位置到矩阵最近距离*2*k)最小**，将其周围的点加上ban禁止清洗液滴经过，进行操作即可，执行完毕后取消ban。无法找到则暂停操作。

代码

在自动规划模式下，点击下一步时执行如下代码：

```
void MainWindow::routeNextStep(){
    if(routeOperPoint >= routeOperations.length()) return ;

    ui->actionNextStep->setEnabled(false);
    memcpy(routeLastDrop, nowDrop, sizeof(nowDrop)) ;
    memset(routeMoved,0,sizeof(routeMoved));
    //debug("test1");
    RouteOperation oper = routeOperations.at(routeOperPoint);
    routeCriticalDrop=-1;
    debugOper(oper);
    //debug("test2");
    routeMoveSuc = false;

    if(oper.opt==2){
        //Split
        int drop1=oper.drop1, drop2=oper.drop2, drop3=oper.drop3;
        routeCriticalDrop=drop1;
        QPoint p = routeGetDropPos(drop1) ;
        debug(QString("Split. drop:%1 routeSplitTarget:
(%2,%3)").arg(drop1).arg(routeSplitTarget.x()).arg(routeSplitTarget.y())) ;
        if(routeSplitTarget!=QPoint(-1,-1) || routeGetSplitTarget(drop1, p)){
            QPoint p1 = routeSplitTarget , p2 = routeSplitTarget + QPoint(-1,0) , p3 =
routeSplitTarget + QPoint(1,0) ;
            if(routeSplitMid){
                notAlone[p2.x()][p2.y()] = notAlone[p3.x()][p3.y()] = false ;
                midState[p1.x()][p1.y()] = QPoint(0, 0) ;
                nowDrop[p1.x()][p1.y()] = 0 ;
                routeMoved[drop1] = routeMoved[drop2] = routeMoved[drop3] = true ;
                routeSplitMid=false;
                routeOperPoint++;
                memset(routeWashBan,0,sizeof(routeWashBan)) ;
```

```
                routeSplitTarget=QPoint(-1,-1);
                routeSplitPath.clear();
                playSplit2Sound();
                routeMoveSuc = true;
            } else if(routeSplitPath.length()==1){
                if(routeCheckPos(drop1, p2) && routeCheckPos(drop1, p3)){
                    routeSplitMid=true;
                    routePlaceDrop(drop2, routeSplitTarget+QPoint(-1,0)) ;
                    routePlaceDrop(drop3, routeSplitTarget+QPoint(1,0)) ;
                    routeMoved[drop1] = routeMoved[drop2] = routeMoved[drop3] = true ;
                    notAlone[p2.x()][p2.y()] = notAlone[p3.x()][p3.y()] = true ;
                    midState[p1.x()][p1.y()] = QPoint(1, 1) ;
                    playSplit1Sound();
                    routeMoveSuc = true;
                }
            } else if(routeSplitPath.length()>1 && routeCheckPos(drop1, routeSplitPath.at(1))){
                routeMoveDrop(drop1, routeSplitPath.at(0), routeSplitPath.at(1)) ;
                routeSplitPath.pop_front() ;
                routeMoved[drop1] = true ;
                routeMoveSuc = true;
            }
        }
    } else if(oper.opt==3){
        //Merge
        int drop1=oper.drop1, drop2=oper.drop2, drop3=oper.drop3;
        routeCriticalDrop=drop1;
        if(routeMergeTarget1!=QPoint(-1,-1) || routeGetMergeTarget(drop1,
routeGetDropPos(drop1), drop2, routeGetDropPos(drop2))){
            QPoint p1=routeMergeTarget1, p2=routeMergeTarget2, p3=(p1+p2)/2 ;
            //有Target
            debug(QString("Merge. Target1:(%1,%2)  Target2:(%3,%4)  Len1:%5
Len2:%6").arg(p1.x()).arg(p1.y()).arg(p2.x()).arg(p2.y())
                  .arg(routeMergePath1.length()).arg(routeMergePath2.length()));
            if(routeMergeMid){
          //      debug("test3");
                routeMergeMid=false;
                nowDrop[p1.x()][p1.y()]=nowDrop[p2.x()][p2.y()]=0;
                midState[p3.x()][p3.y()] = QPoint(0, 0) ;
                notAlone[p1.x()][p1.y()]=notAlone[p2.x()][p2.y()]=false ;
                memset(routeWashBan,0,sizeof(routeWashBan));
                routeMergePath1.clear();
                routeMergePath2.clear();
                routeMergeTarget1=routeMergeTarget2=QPoint(-1,-1);
                routeOperPoint++;
                routeMoved[drop1+MAXM] = routeMoved[drop2+MAXM] = routeMoved[drop3+MAXM] = true
;
                routeMoveSuc = true;
                playMergeSound();
            } else if(routeMergePath1.length()==1 && routeMergePath2.length()==1){
          //      debug("test4");
                routeMergeMid=true;
                routePlaceDrop(drop3, p3);

                midState[p3.x()][p3.y()] = QPoint(2, 1) ;
```

```
                notAlone[p1.x()][p1.y()]=notAlone[p2.x()][p2.y()]=true ;
                routeMoved[drop1+MAXM] = routeMoved[drop2+MAXM] = routeMoved[drop3+MAXM] = true
;
                routeMoveSuc = true;
            }
            else{
          //      debug("test5");
                if(routeMergePath1.length()>1 && routeCheckPos(drop1, routeMergePath1.at(1))){
                    routeMoveDrop(drop1,routeMergePath1.at(0),routeMergePath1.at(1)) ;
                    routeMergePath1.pop_front();
                    routeMoved[drop1+MAXM] = true ;
                    routeMoveSuc = true;
                }
                if(routeMergePath2.length()>1 && routeCheckPos(drop2, routeMergePath2.at(1))){
                    routeMoveDrop(drop2,routeMergePath2.at(0),routeMergePath2.at(1)) ;
                    routeMergePath2.pop_front();
                    routeMoved[drop2+MAXM] = true ;
                    routeMoveSuc = true;
                }
            }
        }
    } else if(oper.opt==4){
        //Input
        int drop=oper.drop1;
        QPoint p=routeInPortList.at(routeInPortOfDrop[drop]-1);
        if(routeCheckPos(drop, p) && !nowDrop[p.x()][p.y()]){
            routePlaceDrop(drop, p);
            routeOperPoint++;
            routeMoved[drop+MAXM] = true ;
            routeMoveSuc = true;
        }
    } else if(oper.opt==5){
        //Output
        int drop=oper.drop1;
        routeCriticalDrop=drop;
        QPoint p=routeGetDropPos(drop), outPort=routeOutPortList.at(routeOutPortOfDrop[drop]-1);
        if(routeIsOutputing || routeGetOutputTarget(drop, p)){
            if(routeOutputPath.length()==1){
                nowDrop[outPort.x()][outPort.y()]=0 ;
                routeIsOutputing=false;
                routeOperPoint++;
                memset(routeWashBan,0,sizeof(routeWashBan)) ;
                routeMoved[drop+MAXM]=true;
                routeMoveSuc=true;
            } else{
                if(routeCheckPos(drop, routeOutputPath.at(1))){
                    routeMoveDrop(drop, routeOutputPath.at(0), routeOutputPath.at(1)) ;
                    routeOutputPath.pop_front();
                    routeMoved[drop+MAXM]=true;
                    routeMoveSuc=true;
                }
            }

        } else {
```

```
            debug("Move idle.") ;
            for(int x=1;x<=col;++x){
                for(int y=1;y<=row;++y){
                    if(nowDrop[x][y]>0 && nowDrop[x][y]!=drop && (calcChebyshevDis(QPoint(x,y),
p)<=4 || calcChebyshevDis(QPoint(x,y), outPort)<=4)){
                        int drop2=nowDrop[x][y] ;
                        debug(QString("Idle. drop2:%1 pos:(%2,%3)").arg(drop2).arg(x).arg(y)) ;
                        if(!routeMoved[drop2+MAXM]){
                            routeMoved[drop2+MAXM]=true;
                            int minValue = INF ;
                            QPoint nxt = QPoint(-1,-1);
                            for(int k=MOVEDIRNUM-1;k>=0;--k){
                                int tx=x+dir[k][0], ty=y+dir[k][1];
                                if(!outGridRange(QPoint(tx,ty)) && routeCheckPos(drop2,
QPoint(tx,ty))){
                                    int value = routeCalcBlockValue(QPoint(tx,ty)) - routeBlockK
* 2 * calcChebyshevDis(QPoint(tx,ty), p) ;
                                    if(value < minValue){
                                        minValue = value ;
                                        nxt = QPoint(tx,ty);
                                    }
                                }
                            }
                            if(nxt!=QPoint(-1,-1)){
                                routeMoveDrop(drop2, QPoint(x,y), nxt) ;
                                routeMoved[drop2+MAXM]=true;
                                //     routeMoveSuc = true;
                            }
                        }
                    }
                }
            }
        }
    } else if(oper.opt==6){
        //Mix
        int drop=oper.drop1, mixLen=oper.mixLen;
        routeCriticalDrop=drop;
        QPoint p=routeGetDropPos(drop) ;
        debug(QString("routeMixPath.length():%1
routeIsMixing:%2").arg(routeMixPath.length()).arg(routeIsMixing));
        if(routeIsMixing || routeGetMixTarget(drop, p, mixLen)){
            routeIsMixing = true;
            debug(QString("routeMixStart:(%1,%2)
routeMixTarget(%3,%4)").arg(routeMixStart.x()).arg(routeMixStart.y()).arg(routeMixTarget.x()).ar
g(routeMixTarget.y())) ;
            if(routeMixPath.length()>1)  debug(QString("Mix. Next Point:
(%1,%2)").arg(routeMixPath.at(1).x()).arg(routeMixPath.at(1).y()));
            if(routeMixPath.length()>1 && routeCheckPos(drop, routeMixPath.at(1))){
                routeMoveDrop(drop,routeMixPath.at(0),routeMixPath.at(1)) ;
                routeMixPath.pop_front();
                routeMoved[drop+MAXM] = true;
                routeMoveSuc = true;

            }
```

```
            if(routeMixPath.length()==1){
                routeIsMixing = false;
                memset(routeWashBan,0,sizeof(routeWashBan));
                routeOperPoint++;
            }
        }
    }
    bool washSuc = routeHandleWashDrop();
    if(!routeMoveSuc && !washSuc){
        for(int x=1;x<=col;++x){
            for(int y=1;y<=row;++y){
                if(nowDrop[x][y]){
                    int drop=nowDrop[x][y] ;
                    if(!routeMoved[drop+MAXM]){
                        routeMoved[drop+MAXM]=true;
                        int minValue = INF ;
                        QPoint nxt = QPoint(-1,-1);
                        for(int k=MOVEDIRNUM-1;k>=0;--k){
                            int tx=x+dir[k][0], ty=y+dir[k][1];
                            if(!outGridRange(QPoint(tx,ty)) && routeCheckPos(drop,
QPoint(tx,ty))){
                                int value = routeCalcBlockValue(QPoint(tx,ty)) ;
                                if(value < minValue){
                                    minValue = value ;
                                    nxt = QPoint(tx,ty);
                                }
                            }
                        }
                        if(nxt!=QPoint(-1,-1))
                            routeMoveDrop(drop, QPoint(x,y), nxt) ;
                    }
                }
            }
        }
    }
    ui->labelCurTime->setNum(++timeNow);
    if(!playingAll) ui->actionNextStep->setEnabled(true);
    update();
}
```

清洗液滴的寻路

注意清洁液滴用负数编号表示，依次为-1，-2等等。

每次清洗时，给被污染的格子进行一次排序，排序的权值**按照 (k*离最近的端口距离+离当前清洗液滴距离) 从小到大排序**，k暂时取3，每次向着第一个进行寻路。

当清洗液滴清洗满三个污染点之后判断生成下一个清洗液滴，注意需要满足可以在入口生成清洗液滴，且该液滴能够寻路走到一个污染格子处才生成。并且接下来对清洗满了的清洗液滴进行到出口的BFS寻路。

代码

这部分代码如下：

```
bool MainWindow::routeHandleWashDrop(){
    //处理清洁液滴，若清洁液滴有任何变动均返回true
    bool res=false, ret=false; //res为有无剩余容量，ret为返回值
    QList<int>::iterator it;
    for(it=routeWashDrops.begin();it!=routeWashDrops.end();++it){
        int drop = *it ;
        QPoint p = routeGetDropPos(drop) ;
       // debug(QString("wash. drop:%1 nowPos:(%2,%3)
cap:%4").arg(drop).arg(p.x()).arg(p.y()).arg(routeWashDropCap[drop+MAXM])) ;
        if(routeWashDropCap[drop+MAXM]==0){
            //应当到清洁液滴出口
            if(p == routeWashOutPort){

                nowDrop[routeWashOutPort.x()][routeWashOutPort.y()]=0;
                it = routeWashDrops.erase(it) ;
                -- it;
                //debug(QString("!!!!!!!!!!!!!!!DELETE drop:%1  p:
(%2,%3)").arg(drop).arg(p.x()).arg(p.y())) ;
                ret = true;
                continue ;
            } else{
                routeBFS(drop, p, true) ;
                //debug(QString("washDrop:%1(usedUp)
canGo:%2").arg(routeWashDrops.at(i)).arg(bfsDis[routeWashOutPort.x()][routeWashOutPort.y()])) ;
                if(bfsDis[routeWashOutPort.x()][routeWashOutPort.y()]){
                    QPoint nxt = routeGetNextPos(p, routeWashOutPort) ;
                    routeMoveDrop(drop, p, nxt) ;
                    ret = true;
                }
            }
        } else{
            res = true ;
            routeBFS(drop, p, false) ;
            int minValue = INF ;
            QPoint target=QPoint(-1,-1);
            debug(QString("routeMoveSuc:%1
routeCriticalDrop:%2").arg(routeMoveSuc).arg(routeCriticalDrop)) ;
            for(int x=1;x<=col;++x){
                for(int y=1;y<=row;++y){
                    if(bfsWashDis[x][y] && histDrop[x][y].size()>0){
                        int value = bfsWashDis[x][y]-1-routeCalcBlockValue(QPoint(x,y)) ;
                        if(!routeMoveSuc && routeCriticalDrop!=-1){
                            value += routeBlockK *
(calcChebyshevDis(routeGetDropPos(routeCriticalDrop), QPoint(x,y))) ;
                            if(routeOperations.at(routeOperPoint).opt==5)
                                value += routeBlockK *
calcChebyshevDis(routeOutPortList.at(routeOutPortOfDrop[routeCriticalDrop]-1), QPoint(x,y)) ;
                        }
                        if(value < minValue){
                            minValue = value;
                            target = QPoint(x,y) ;
                        }
                    }
```

```
                }
            }
            if(target!=QPoint(-1,-1)){
                QPoint nxt = routeGetNextPos(p, target) ;
          //      debug(QString("wash. drop:%1 nxtPos:(%2,%3) target:
(%4,%5)").arg(drop).arg(nxt.x()).arg(nxt.y()).arg(target.x()).arg(target.y())) ;
                routeMoveDrop(drop, p, nxt) ;
                if(routeWashDropCap[drop+MAXM]==0) res=false;
                ret = true;
            }
        }
    }
    //debug(QString("wash. res:%1 routeWashDropCnt:%2").arg(res).arg(routeWashDropCnt)) ;
    if(!res && routeCheckPos(-MAXM, routeWashInPort, true) && !nowDrop[routeWashInPort.x()]
[routeWashInPort.y()]){
        routeBFS(-MAXM, routeWashInPort) ;
        bool suc=false;
        for(int x=1;x<=col;++x){
            for(int y=1;y<=row;++y){
                if(bfsWashDis[x][y] && histDrop[x][y].size()>0){
                    suc=true;
                }
            }
        }
        if(suc){
            routeWashDrops.append(--routeWashDropCnt) ;
            routeWashDropCap[routeWashDropCnt+MAXM] = 3;
            routePlaceDrop(routeWashDropCnt, routeWashInPort) ;
            ret = true;
        }
    }
    return ret;
}
```

避免卡死的处理办法

增加变量moveSuc表示是否成功进行任何单步操作。若失败，则判断清洗液滴是否正常移动。若再次失败，则将液滴贪心地向着附近阻塞值最小的方向移动。

另外的加分项

关于颜色

为了颜色的丰富性，对于单滴液滴，都是通过随机来产生颜色的。不过为了美观，在分裂和合并时，采用RGB运算来得到合适的颜色。

在合并时比较简单，直接计算原来两颜色的中间值来生成新的颜色；而在分裂时通过随机一个步长step，将rgb三个中间值各自分裂成两个值，再随机分配给新的颜色。

另外当然为了更加严谨，在进行“上一步”回退之后，再“下一步”生成的液滴颜色并不会再次改变。

代码

代码如下：

```
void MainWindow::newDropColor(int type=0, QColor a=QColor(0,0,0), QColor b=QColor(0,0,0)){
    //type==0时为随机产生新颜色，type==1时为分裂a产生两种颜色，type==2时为合并a、b产生一种颜色
    //产生的新颜色放在dropColor最后
    if(type==1){
        int tmpRGB[3], newRGB1[3], newRGB2[3];
        tmpRGB[0]=a.red(); tmpRGB[1]=a.green(); tmpRGB[2]=a.blue();
        for(int i=0;i<3;++i){
            int tmp=tmpRGB[i];
            int rang=rd(0,std::min(tmp-20,230-tmp));
            newRGB1[i]=tmpRGB[i]-rang;
            newRGB2[i]=tmpRGB[i]+rang;
            if(rd(0,1)) std::swap(newRGB1[i],newRGB2[i]);
        }
        dropColor.append(QColor(newRGB1[0],newRGB1[1],newRGB1[2]));
        dropColor.append(QColor(newRGB2[0],newRGB2[1],newRGB2[2]));
    } else if(type==2){
        dropColor.append(QColor((a.red()+b.red())/2,(a.green()+b.green())/2,
(a.blue()+b.blue())/2));
    }
    else    dropColor.append(QColor(qrand()%200+25,qrand()%200+25,qrand()%200+25));
}
```

自动配置

依据输入文档，自动进行长、宽以及输入、输出的配置，可以非常方便，并且自动检测了其中输入、输出端口是否出现问题。

实现时通过判断能否通过边界端口支持该文档中的Input和Output，并且判断输出端口是否唯一。而长宽通过分别取指令中长宽的最大值来定。

这一部分代码如下：

```
void MainWindow::autoSet(){
    int maxCol=0, maxRow=0;
    for(int i=0;i<=timeLim;++i){
        for(int j=0;j<instructions[i].length();++j){
            Instruction inst = instructions[i].at(j);
            for(int k=0;k<6;k+=2)
                maxCol=std::max(maxCol, inst.arg[k]);
            for(int k=1;k<6;k+=2)
                maxRow=std::max(maxRow, inst.arg[k]);
        }
    }
    for(int i=0;i<=timeLim;++i){
        for(int j=0;j<instructions[i].length();++j){
            Instruction inst = instructions[i].at(j);
            if(inst.opt==4 || inst.opt==5){
                if(!isOnEdge(QPoint(inst.arg[0],inst.arg[1]),maxCol,maxRow)){
                    QMessageBox::critical(this, "自动配置失败", QString("端口无法设定在边界上")) ;
                    return ;
```

```
                }
            }
        }
    }
    col=maxCol; row=maxRow;
    inPortStr=outPortStr="";
    inPortList.clear();
    outPortList.clear();
    for(int i=0;i<=timeLim;++i){
        for(int j=0;j<instructions[i].length();++j){
            Instruction inst = instructions[i].at(j);
            if(inst.opt==4){
                QPoint p=QPoint(inst.arg[0],inst.arg[1]);
                if(inPortList.indexOf(p)==-1){
                    inPortList.append(p);
                    if(inPortList.length()>1) inPortStr=inPortStr+';';
                    inPortStr=inPortStr+QString("%1,%2").arg(p.x()).arg(p.y());
                }
            } else if(inst.opt==5){
                QPoint p=QPoint(inst.arg[0],inst.arg[1]);
                if(outPortList.indexOf(p)==-1){
                    if(outPortList.length()!=0){
                        QMessageBox::critical(this, "自动配置失败", QString("出现多个输出端口")) ;
                        return ;
                    }
                    outPortList.append(p);
                    outPortStr=QString("%1,%2").arg(p.x()).arg(p.y());
                }
            }
        }
    }
}
```

目录记忆

每次打开文件时，目录为上次打开文件处的文件夹。这样一来，对于需要频繁打开文件的用户便会更加友好。

调试经历

关于音频播放

一开始使用的QSound播放MP3格式，发现无法播放，经过测试发现可以播放其它wav格式的音频。于是最终将MP3格式音频转换为wav格式后即正常播放。

但是会出现卡顿，后来改成QMediaPlayer之后便不再卡顿了。

附代码

只附上关键的.cpp代码。

setdmfbdialog.cpp

```
#include "setdmfbdialog.h"
#include "ui_setdmfbdialog.h"
#include <QString>
#include "mainwindow.h"
#include <QMessageBox>

SetDMFBDialog::SetDMFBDialog(QWidget *parent, int col, int row, QString inPortStr, QString
outPortStr, MainWindow *mainWindow) :
    QDialog(parent),
    ui(new Ui::SetDFMBDialog)
{
    ui->setupUi(this);
    ui->spinBoxCol->setValue(col);
    ui->spinBoxRow->setValue(row);
    ui->lineEditInPort->setText(inPortStr);
    ui->lineEditOutPort->setText(outPortStr);
    this->mainWindow = mainWindow;
}

SetDMFBDialog::~SetDMFBDialog()
{
    delete ui;
}

bool isOnEdge(QPoint p, int col, int row)
{
    return p.x()==1||p.x()==col||p.y()==1||p.y()==row;
}

void SetDMFBDialog::on_buttonBox_accepted()
{
    int col=ui->spinBoxCol->value();
    int row=ui->spinBoxRow->value();
    if(col<=3 && row<=3){
        QMessageBox::critical(this, "参数错误", "行列不允许同时小于等于3");
        this->show();
        return;
    }
    int ret1 = mainWindow->parsePortStr(ui->lineEditInPort->text(), col, row);
    if(ret1==-1){
        QMessageBox::critical(this, "参数错误", "输入端口格式错误");
        this->show();
        return;
    }
    else if(ret1==-2){
        QMessageBox::critical(this, "参数错误", "输入端口坐标超出范围");
        this->show();
        return;
    }

    for(int i=0;i<ret1;++i){
```

```
        if(ret1!=0 && !isOnEdge(mainWindow->tmpList.at(i),col,row)){
            QMessageBox::critical(this, "参数错误", "输入端口不在边界上");
            this->show();
            return;
        }
    }
    QList<QPoint> inPortList=mainWindow->tmpList;
    int ret2 = mainWindow->parsePortStr(ui->lineEditOutPort->text(), col, row);
    if(ret2==-1){
        QMessageBox::critical(this, "参数错误", "输出端口格式错误");
        this->show();
        return;
    }
    else if(ret2==-2){
        QMessageBox::critical(this, "参数错误", "输出端口坐标超出范围");
        this->show();
        return;
    } else if(ret2>1){
        QMessageBox::critical(this, "参数错误", "输出端口不唯一");
        this->show();
        return;
    }
    if(!isOnEdge(mainWindow->tmpList.at(0),col,row)){
        QMessageBox::critical(this, "参数错误", "输出端口不在边界上");
        this->show();
        return;
    }
    mainWindow->setCol(col);
    mainWindow->setRow(row);
    mainWindow->setInPortStr(ui->lineEditInPort->text());
    mainWindow->setOutPortStr(ui->lineEditOutPort->text());
    if(ret1>0) mainWindow->inPortList  = inPortList;
    else mainWindow->inPortList.clear();
    if(ret2>0) mainWindow->outPortList = mainWindow->tmpList;
    else mainWindow->outPortList.clear();
    mainWindow->init();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "setdmfbdialog.h"
#include <QDebug>
#include <QPainter>
#include <QPoint>
#include <QPaintEvent>
#include <QSizePolicy>
#include <QColor>
#include <QFileDialog>
#include <QFile>
#include <QMessageBox>
```

```
#include <QTimer>
#include <cmath>
#include <QException>
#include <QSound>
#include <algorithm>
#include <QMediaPlayer>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    gridSize=40;
    tinySize=gridSize/5;
    filePath="./";
    leftUp=QPoint(60,135);
    timeLim=timeNow=0;
    col=row=5;
    debugOn=true;
    timerPlayAll=new QTimer(this);
    timerWash=new QTimer(this);
    connect(timerPlayAll, SIGNAL(timeout()), this, SLOT(on_actionNextStep_triggered()));
    connect(timerWash, SIGNAL(timeout()), this, SLOT(washNext()));
    debugPreLoad();
    init();
    update();
}

void MainWindow::debugPreLoad()
{
    if(debugOn){
        int T=2;
        if(T==0){
            filePath=tr("E:/作业及课件/大二小学期/作业/贵系程设/Week1/Week1/Input/testcase0.txt");
        }
        else if(T==1){
            filePath=tr("E:/作业及课件/大二小学期/作业/贵系程设/Week1/Week1/Input/testcase1.txt");
        } else if(T==2){
            filePath=tr("E:/作业及课件/大二小学期/作业/贵系程设/Week1/Week1/Input/testcase2.txt");
        } else if(T==3){
            filePath=tr("E:/作业及课件/大二小学期/作业/贵系程设/Week1/Week1/Input/testcase3.txt");
        } else if(T==4){
            filePath=tr("E:/作业及课件/大二小学期/作业/贵系程
设/Week1/Week1/Input/testcaseerror.txt");
        } else if(T==5){
            filePath=tr("E:/作业及课件/大二小学期/作业/贵系程
设/Week1/Week1/Input/testcasewash.txt");
        }
//        parsePortStr(inPortStr, col, row);
//        inPortList=tmpList;
//        parsePortStr(outPortStr, col, row);
//        outPortList=tmpList;

        openFileWithPath(filePath) ;
```

```
    }
}

void MainWindow::debug(QString s){
    if(debugOn){
        qDebug() << s;
    }
}

void MainWindow::debugDrop(int drop){
    if(debugOn){
        qDebug() << QString("drop: %1 RGB:(%2,%3,%4)").arg(drop).arg(dropColor.at(drop-1).red())
              .arg(dropColor.at(drop-1).green()).arg(dropColor.at(drop-1).blue());
    }
}

QPoint MainWindow::getPoint(int a, int b){
    //获得网格点的坐标，a=1且b=1时为左下角的网格点，a增大则向右移动，b增大则向上移动
    b=row+2-b; //坐标变换
    return QPoint((a-1)*gridSize+leftUp.x(), (b-1)*gridSize+leftUp.y());
}

QPoint MainWindow::getEdgeInd(QPoint p){
    //从边界的网格索引得到相邻的边界外处的网格索引
    int x=p.x();
    int y=p.y();
    if(y==row)
        return QPoint(x,y+1);
    else if(y==1)
        return QPoint(x,y-1);
    else if(x==1)
        return QPoint(x-1,y);
    else if(x==col)
        return QPoint(x+1,y);
    else {
        assert(0);
        return QPoint(-1,-1);
    }
}

int MainWindow::getCol()
{
    return col;
}

int MainWindow::getRow()
{
    return row;
}

QString MainWindow::getInPortStr()
{

    return inPortStr;
```

```
}

QString MainWindow::getOutPortStr()
{
    return outPortStr;
}

void MainWindow::setCol(int col)
{
    this->col = col;
}

void MainWindow::setRow(int row)
{
    this->row = row;
}

void MainWindow::setInPortStr(QString inPortStr)
{
    this->inPortStr = inPortStr;
}

void MainWindow::setOutPortStr(QString outPortStr)
{
    this->outPortStr = outPortStr;
}

int MainWindow::parsePortStr(QString portStr, int col, int row)
{
    // 解析表示端口的字符串，存储到tmpList这个成员变量中，类型为QList<QPoint>。返回-1表示解析出错，-2
表示数字范围出错，否则返回长度
    if(portStr==""){
        return 0;
    }
    QStringList strList = portStr.split(';');
    int len = strList.length();
    tmpList.clear();
    for(int i=0;i<len;++i){
        QString nowStr = strList.at(i) ;
        QStringList strList2 = nowStr.split(',');
        if(strList2.length()!=2) return -1;
        bool ok;
        int x=strList2.at(0).toInt(&ok);
        if(!ok) return -1;
        int y=strList2.at(1).toInt(&ok);
        if(!ok) return -1;
        if(x<1 || x>col || y<1 || y>row) return -2;
        tmpList.append(QPoint(x,y));
    }
    return len;
}


void MainWindow::paintEvent(QPaintEvent *event){
```

```
//坐标变换在绘图时实现，其它时候都是按照题目的坐标来做的
QPainter painter(this) ;
painter.setRenderHint(QPainter::Antialiasing, true); // 抗锯齿

//网格线
for(int i=1;i<=col+1;++i){
    for(int j=1;j<=row+1;++j){
        if(i!=col+1) painter.drawLine(getPoint(i,j),getPoint(i+1,j));
        if(j!=row+1) painter.drawLine(getPoint(i,j),getPoint(i,j+1));
    }
}
QPoint tmp = getPoint(col+1,1);
this->setMinimumSize(tmp.x()+60, tmp.y()+60);
this->setMaximumSize(tmp.x()+60, tmp.y()+60);

//输入和输出端口
painter.setBrush(QBrush(Qt::red,Qt::SolidPattern));
if(!ui->actionRoute->isChecked()){
    for(int i=0;i<inPortList.length();++i){
        tmp = inPortList.at(i);
        tmp = getEdgeInd(tmp);
        tmp = getPoint(tmp.x(),tmp.y()+1) ;
        painter.drawRoundRect(tmp.x(),tmp.y(),gridSize,gridSize);
    }
} else{
    for(int i=0;i<routeInPortList.length();++i){
        tmp = routeInPortList.at(i);
        tmp = getEdgeInd(tmp);
        tmp = getPoint(tmp.x(),tmp.y()+1) ;
        painter.drawRoundRect(tmp.x(),tmp.y(),gridSize,gridSize);
    }
}
painter.setBrush(QBrush(Qt::blue,Qt::SolidPattern));
if(!ui->actionRoute->isChecked()){
    for(int i=0;i<outPortList.length();++i){
        tmp = outPortList.at(i);
        tmp = getEdgeInd(tmp);
        tmp = getPoint(tmp.x(),tmp.y()+1) ;
        painter.drawRoundRect(tmp.x(),tmp.y(),gridSize,gridSize);
    }
} else{
    for(int i=0;i<routeOutPortList.length();++i){
        tmp = routeOutPortList.at(i);
        tmp = getEdgeInd(tmp);
        tmp = getPoint(tmp.x(),tmp.y()+1) ;
        painter.drawRoundRect(tmp.x(),tmp.y(),gridSize,gridSize);
    }
}

//清洗液滴的端口
if(ui->actionWash->isChecked()){
    painter.setBrush(QBrush(QColor(100,50,80),Qt::SolidPattern));

    if(!ui->actionRoute->isChecked()) tmp = QPoint(0,1);
```

```
        else tmp = getEdgeInd(routeWashInPort) ;
        tmp = getPoint(tmp.x(),tmp.y()+1);
        painter.drawRoundRect(tmp.x(),tmp.y(),gridSize,gridSize);

        painter.setBrush(QBrush(QColor(50,80,120),Qt::SolidPattern));
        if(!ui->actionRoute->isChecked()) tmp = QPoint(col+1,row);
        else tmp = getEdgeInd(routeWashOutPort) ;
        tmp = getPoint(tmp.x(),tmp.y()+1);
        painter.drawRoundRect(tmp.x(),tmp.y(),gridSize,gridSize);
    }

    if((!ui->actionRoute->isChecked() && timeNow==timeLim && timeNow!=0) || (ui->actionRoute-
>isChecked() && routeOperPoint>=routeOperations.length())){
        //最后的显示污染次数
        int maxn=0;
        for(int i=1;i<=col;++i) for(int j=1;j<=row;++j){
            maxn = std::max(maxn, histDrop[i][j].size()) ;
        }
        for(int i=1;i<=col;++i){
            for(int j=1;j<=row;++j){
                QFont font;
                if(maxn<=99) font.setPointSize(20);
                else font.setPointSize(13);
                painter.setFont(font);
                painter.drawText(QRect(getPoint(i,j+1), getPoint(i+1,j)), Qt::AlignCenter,
QString::number(histDrop[i][j].size())) ;
            }
        }

    } else{
        //液滴
        for(int i=1;i<=col;++i){
            for(int j=1;j<=row;++j){
                if(ban[i][j]){ //障碍
                    painter.setPen(Qt::red);
                    painter.drawLine(getPoint(i,j+1)+QPoint(1,1), getPoint(i+1,j)-QPoint(1,1));
                    painter.drawLine(getPoint(i+1,j+1)+QPoint(-1,1),
getPoint(i,j)+QPoint(1,-1));
                    painter.setPen(Qt::black);
                }
                if(ui->actionRoute->isChecked() && nowDrop[i][j]<0){
                    QColor color;
                    if(routeWashDropCap[nowDrop[i][j]+MAXM]==3)
                        color=QColor(58,159,177) ;
                    else if(routeWashDropCap[nowDrop[i][j]+MAXM]==2)
                        color=QColor(47,138,154) ;
                    else if(routeWashDropCap[nowDrop[i][j]+MAXM]==1)
                        color=QColor(28,110,125) ;
                    else if(routeWashDropCap[nowDrop[i][j]+MAXM]==0)
                        color=QColor(54,100,108) ;
                    painter.setBrush(QBrush(color,Qt::SolidPattern));
                    tmp = getPoint(i,j+1) ;

                    painter.drawEllipse(tmp.x(),tmp.y(),gridSize,gridSize) ;
```

```
                } else if(nowDrop[i][j]>0 && !notAlone[i][j] && !midState[i][j].x()){
                    painter.setBrush(QBrush(dropColor.at(nowDrop[i][j]-1),Qt::SolidPattern));
                    tmp = getPoint(i,j+1) ;
                    painter.drawEllipse(tmp.x(),tmp.y(),gridSize,gridSize) ;
                } else if(midState[i][j].x()){
                    painter.setBrush(QBrush(dropColor.at(nowDrop[i][j]-1),Qt::SolidPattern));
                    if(midState[i][j].y()==1){
                        tmp = getPoint(i-1,j+1) ;
                        painter.drawEllipse(tmp.x()+gridSize/2, tmp.y(), gridSize*2, gridSize);
                    } else{
                        tmp = getPoint(i,j+2) ;
                        painter.drawEllipse(tmp.x(), tmp.y()+gridSize/2, gridSize, gridSize*2);
                    }
                }
                if(!nowDrop[i][j] && !notAlone[i][j]){
                    QMapIterator<int, int> it(histDrop[i][j]);
                    while(it.hasNext()){
                        if(it.next().value()>0){
                            painter.setBrush(QBrush(dropColor.at(it.key()-1),Qt::SolidPattern))
;
                            tmp = getPoint(i,j+1) ;
                            QMap<int, QPoint>::iterator tmpIt = tinyPos[i][j].find(it.key());
                            int x,y;
                            if(tmpIt==tinyPos[i][j].end()){
                                x=rd(tmp.x(),tmp.x()+gridSize-tinySize);
                                y=rd(tmp.y(),tmp.y()+gridSize-tinySize);
                                tinyPos[i][j][it.key()] = QPoint(x,y) ;
                            } else {
                                x=(*tmpIt).x();
                                y=(*tmpIt).y();
                            }
                            painter.drawEllipse(x, y, tinySize, tinySize);
                        }
                    }
                }
            }
        }

        //清洁液滴
        if(isWashing && !ui->actionRoute->isChecked()){
            int x=washPath.first().x(), y=washPath.first().y();
            QColor color=QColor(58,159,177) ;
            painter.setBrush(QBrush(color,Qt::SolidPattern));
            tmp = getPoint(x,y+1) ;
            painter.drawEllipse(tmp.x(),tmp.y(),gridSize,gridSize) ;
        }
    }
}

int MainWindow::rd(int l,int r){
    return qrand()%(r-l+1)+l;
}
```

```
void MainWindow::newDropColor(int type=0, QColor a=QColor(0,0,0), QColor b=QColor(0,0,0)){
    //type==0时为随机产生新颜色，type==1时为分裂a产生两种颜色，type==2时为合并a、b产生一种颜色
    //产生的新颜色放在dropColor最后
    if(type==1){
        int tmpRGB[3], newRGB1[3], newRGB2[3];
        tmpRGB[0]=a.red(); tmpRGB[1]=a.green(); tmpRGB[2]=a.blue();
        for(int i=0;i<3;++i){
            int tmp=tmpRGB[i];
            int rang=rd(0,std::min(tmp-20,230-tmp));
            newRGB1[i]=tmpRGB[i]-rang;
            newRGB2[i]=tmpRGB[i]+rang;
            if(rd(0,1)) std::swap(newRGB1[i],newRGB2[i]);
        }
        dropColor.append(QColor(newRGB1[0],newRGB1[1],newRGB1[2]));
        dropColor.append(QColor(newRGB2[0],newRGB2[1],newRGB2[2]));
    } else if(type==2){
        dropColor.append(QColor((a.red()+b.red())/2,(a.green()+b.green())/2,
(a.blue()+b.blue())/2));
    }
    else    dropColor.append(QColor(qrand()%200+25,qrand()%200+25,qrand()%200+25));
}

void MainWindow::init(){
    //TODO!!!!!!!!!!!!!!!!做整个系统的初始化
    qsrand(time(NULL));
    if(!ui->actionRoute->isChecked()){
        dropColor.clear();
        for(int i=1;i<=MAXM-3;++i) newDropColor() ;
    }
    ui->labelCurTime->setNum(timeNow=0);
    routeIsOutputing=false;
    routeOutputPath.clear();
    routeSplitMid=false;
    routeSplitPath.clear();
    routeSplitTarget=QPoint(-1,-1);
    routeWashDrops.clear();
    routeWashDropCnt=0;
    routeIsMixing=false;
    routeMixPath.clear();
    routeMixTarget=routeMixStart=QPoint(-1,-1);
    routeMergeMid=false;
    routeMergePath1.clear();
    routeMergePath2.clear();
    routeMergeTarget1=routeMergeTarget2=QPoint(-1,-1);
    memset(routeBfsBan,0,sizeof(routeBfsBan));
    memset(routeLastDrop,0,sizeof(routeLastDrop));
    memset(routeDropPos,0,sizeof(routeDropPos));
    memset(routeWashBan,0,sizeof(routeWashBan));
    routeOperPoint=0;
    isWashing=false;
    playingAll=false;
    timerPlayAll->stop();

    timerWash->stop();
```

```
    memset(ban, 0, sizeof(ban)) ;
    ui->actionNextStep->setEnabled(true);
    on_actionPause_triggered();
    dropCnt=0;
    memset(notAlone,0,sizeof(notAlone));
    memset(midState,0,sizeof(midState));
    memset(nowDrop,0,sizeof(nowDrop));
    disapDropStack.clear();
    for(int i=1;i<=col;++i){
        for(int j=1;j<=row;++j){
            histDrop[i][j].clear();
            tinyPos[i][j].clear();
        }
    }
    update();
}

int  MainWindow::parseLine(QString str){
    //失败返回-1，否则返回该条指令的最后执行时刻，注意执行时刻大于MAXTIME时同样返回-1
    Instruction inst;
    QStringList argList = str.split(',') ;
    int time=-1, len=argList.length();
    bool ok;
    QString tmp = argList.at(0);
    time = tmp.right(tmp.length()-tmp.lastIndexOf(' ')).toInt(&ok) ;
    if(!ok) return -1;
    int ret=time+1;

    if(str.left(str.indexOf(' '))=="Move"){
        inst.opt=1;
    } else if(str.left(str.indexOf(' '))=="Split"){
        inst.opt=2;
        ret++;
    } else if(str.left(str.indexOf(' '))=="Merge"){
        inst.opt=3;
        ret++;
    } else if(str.left(str.indexOf(' '))=="Input"){
        inst.opt=4;
    } else if(str.left(str.indexOf(' '))=="Output"){
        inst.opt=5;
    } else if(str.left(str.indexOf(' '))=="Mix"){
        if(len<=1 || (len&1)==0) return -1;
        QPoint last;
        for(int i=1;i<len;i+=2){
            QPoint now;
            QString s = argList.at(i).simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            now.setX(s.toInt(&ok));
            if(!ok) return -1;

            s = argList.at(i+1).simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);

            now.setY(s.toInt(&ok));
```

```
            if(!ok) return -1;

            if(i!=1){
                inst.opt=1;
                inst.arg[0]=last.x();
                inst.arg[1]=last.y();
                inst.arg[2]=now.x();
                inst.arg[3]=now.y();
                if(time+i/2>MAXTIME) return -1;
                instructions[time+i/2-1].append(inst) ;
                lastVis[now.x()][now.y()]=std::max(lastVis[now.x()][now.y()],time+i/2);
            }
            last=now;
        }
        ret=ret+(len-2)/2;
        inst.opt=0;
    }

    if(inst.opt!=0){
        if((inst.opt==1 && len!=5) || (inst.opt==2 && len!=7) || (inst.opt==3 && len!=5)
                || (inst.opt==4 && len!=3) || (inst.opt==5 && len!=3))
            return -1; //操作数个数不对应
        for(int i=1;i<len;++i){
            QString s = argList.at(i);
            s = s.simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            inst.arg[i-1] = s.toInt(&ok);
            if(!ok) return -1;
        }
        if(time>MAXTIME) return -1;
        instructions[time].append(inst) ;

        //处理lastVis
        if(inst.opt==1){
            lastVis[inst.arg[2]][inst.arg[3]] = std::max(lastVis[inst.arg[2]][inst.arg[3]],
time+1);
        } else if(inst.opt==2){
            lastVis[inst.arg[0]][inst.arg[1]] = std::max(lastVis[inst.arg[0]][inst.arg[1]],
time+1);
            lastVis[inst.arg[2]][inst.arg[3]] = std::max(lastVis[inst.arg[2]][inst.arg[3]],
time+2);
            lastVis[inst.arg[4]][inst.arg[5]] = std::max(lastVis[inst.arg[4]][inst.arg[5]],
time+2);
        } else if(inst.opt==3){
            lastVis[inst.arg[0]][inst.arg[1]] = std::max(lastVis[inst.arg[0]][inst.arg[1]],
time+1);
            lastVis[inst.arg[2]][inst.arg[3]] = std::max(lastVis[inst.arg[2]][inst.arg[3]],
time+1);
            int x=(inst.arg[0]+inst.arg[2])/2, y=(inst.arg[1]+inst.arg[3])/2;
            lastVis[x][y] = std::max(lastVis[x][y], time+2);
        } else if(inst.opt==4){


            lastVis[inst.arg[0]][inst.arg[1]] = std::max(lastVis[inst.arg[0]][inst.arg[1]],
```

```
time+1);
        }
    }
    return ret;
}

int MainWindow::parseFile(){
    QFile file(filePath);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QMessageBox::critical(this, "错误", "打开文件失败");
        return -1;
    }
    //因为重新打开了文件，记得初始化!!!!
    memset(lastVis,0,sizeof(lastVis));
    for(int i=0;i<=timeLim;++i)
        instructions[i].clear();
    timeLim=0;
    QTextStream in(&file);
    QString line = in.readLine(); ;
    int cnt=0;
    while(!line.isNull()){
        ++cnt;
        if(line==""){
            line = in.readLine();
            continue;
        }
        int ret = parseLine(line) ;
        if(ret==-1){
            QMessageBox::critical(this, "错误", QString("第%1行指令出错").arg(cnt));
            return -1;
        }
        if(ret>timeLim) timeLim=ret;
        line = in.readLine();
    }
    file.close();
    return 0;
}

MainWindow::~MainWindow()
{
    delete ui;
}

bool isOnEdge(QPoint p, int col, int row);

void MainWindow::autoSet(){
    int maxCol=0, maxRow=0;
    for(int i=0;i<=timeLim;++i){
        for(int j=0;j<instructions[i].length();++j){
            Instruction inst = instructions[i].at(j);
            for(int k=0;k<6;k+=2)
                maxCol=std::max(maxCol, inst.arg[k]);

            for(int k=1;k<6;k+=2)
```

```
                maxRow=std::max(maxRow, inst.arg[k]);
            }
        }
        for(int i=0;i<=timeLim;++i){
            for(int j=0;j<instructions[i].length();++j){
                Instruction inst = instructions[i].at(j);
                if(inst.opt==4 || inst.opt==5){
                    if(!isOnEdge(QPoint(inst.arg[0],inst.arg[1]),maxCol,maxRow)){
                        QMessageBox::critical(this, "自动配置失败", QString("端口无法设定在边界上")) ;
                        return ;
                    }
                }
            }
        }
        col=maxCol; row=maxRow;
        inPortStr=outPortStr="";
        inPortList.clear();
        outPortList.clear();
        for(int i=0;i<=timeLim;++i){
            for(int j=0;j<instructions[i].length();++j){
                Instruction inst = instructions[i].at(j);
                if(inst.opt==4){
                    QPoint p=QPoint(inst.arg[0],inst.arg[1]);
                    if(inPortList.indexOf(p)==-1){
                        inPortList.append(p);
                        if(inPortList.length()>1) inPortStr=inPortStr+';';
                        inPortStr=inPortStr+QString("%1,%2").arg(p.x()).arg(p.y());
                    }
                } else if(inst.opt==5){
                    QPoint p=QPoint(inst.arg[0],inst.arg[1]);
                    if(outPortList.indexOf(p)==-1){
                        if(outPortList.length()!=0){
                            QMessageBox::critical(this, "自动配置失败", QString("出现多个输出端口")) ;
                            return ;
                        }
                        outPortList.append(p);
                        outPortStr=QString("%1,%2").arg(p.x()).arg(p.y());
                    }
                }
            }
        }
    }

    void MainWindow::on_actionSetDFMB_triggered()
    {
        setdfmbdialog = new SetDMFBDialog(this, col, row, inPortStr, outPortStr, this);
        setdfmbdialog -> show();
    }

    void MainWindow::openFileWithPath(QString path){
        ui->labelFileName->setText(path.mid(path.lastIndexOf('/')+1,path.length()));
        int ret=parseFile();

        if(ui->actionAutoSet->isChecked()){
```

```
        autoSet();
    }
    if(ret!=-1){
        routeParseFile();
        if(ui->actionRoute->isChecked()) routeInit();
    }
    init();
}

void MainWindow::on_actionOpenFile_triggered()
{
    filePath = QFileDialog::getOpenFileName(this, "打开文件", filePath.mid(0,
filePath.lastIndexOf('/')+1), "All files (*.*)");
    openFileWithPath(filePath);
}

int MainWindow::getMidState(int x1, int y1, int x2, int y2){
    //输入x1,y1和x2,y2，保证这两个点其中有一个是中间点，判断为水平(1)还是竖直(2)
    if(abs(x1-x2)==1){
        return 1;
    } else return 2;
}

void MainWindow::playMoveSound(){
    if(ui->actionSound->isChecked()) {
        QMediaPlayer *player = new QMediaPlayer(this) ;
        player->setMedia(QUrl::fromLocalFile(soundMovePath)) ;
        player->play();
    }
}

void MainWindow::playSplit1Sound(){
    if(ui->actionSound->isChecked()) {
        QMediaPlayer *player = new QMediaPlayer(this) ;
        player->setMedia(QUrl::fromLocalFile(soundSplit1Path)) ;
        player->play();
    }
}

void MainWindow::playSplit2Sound(){
    if(ui->actionSound->isChecked()) {
        QMediaPlayer *player = new QMediaPlayer(this) ;
        player->setMedia(QUrl::fromLocalFile(soundSplit2Path)) ;
        player->play();
    }
}

void MainWindow::playMergeSound(){
    if(ui->actionSound->isChecked()) {
        QMediaPlayer *player = new QMediaPlayer(this) ;
        player->setMedia(QUrl::fromLocalFile(soundMergePath)) ;
        player->play();

    }
```

```
}

void MainWindow::instMove(int x1, int y1,int x2, int y2, bool rev){
    //从x1,y1移动到x2,y2，rev为true时表明撤销移动
    playMoveSound();
    if(!rev) {
        int drop=nowDrop[x1][y1];
        nowDrop[x2][y2] = drop ;
        nowDrop[x1][y1] = 0;
        histDrop[x2][y2][drop] = histDrop[x2][y2][drop]+1;
    }
    else {
        int drop=nowDrop[x2][y2];
        histDrop[x2][y2][drop] = histDrop[x2][y2][drop]-1;
        nowDrop[x1][y1] = drop;
        nowDrop[x2][y2] = 0;
    }
}

void MainWindow::instSplit1(int x1, int y1,int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1分裂到x2,y2和x3,y3，rev为true时表明撤销分裂的首步
    playSplit1Sound();
    if(!rev){
        nowDrop[x2][y2] = ++dropCnt ;
        nowDrop[x3][y3] = ++dropCnt ;
        histDrop[x2][y2][nowDrop[x2][y2]] = histDrop[x2][y2][nowDrop[x2][y2]]+1;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]+1;
        notAlone[x2][y2] = notAlone[x3][y3] = true;
        midState[x1][y1] = QPoint(1, getMidState(x1,y1,x2,y2));
    } else{
        midState[x1][y1] = QPoint(0,0);
        notAlone[x2][y2] = notAlone[x3][y3] = false;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]-1;
        histDrop[x2][y2][nowDrop[x2][y2]] = histDrop[x2][y2][nowDrop[x2][y2]]-1;
        nowDrop[x2][y2] = nowDrop[x3][y3] = 0;
        dropCnt-=2;
    }
}

void MainWindow::instMerge1(int x1, int y1, int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1和x2,y2合并到x3,y3，rev为true是表示撤销合并的首步
    if(!rev){
        nowDrop[x3][y3] = ++dropCnt ;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]+1;
        notAlone[x1][y1] = notAlone[x2][y2] = true;
        midState[x3][y3] = QPoint(2,getMidState(x2,y2,x3,y3));
    } else{
        midState[x3][y3] = QPoint(0,0);
        notAlone[x1][y1] = notAlone[x2][y2] = false;
        histDrop[x3][y3][nowDrop[x3][y3]] = histDrop[x3][y3][nowDrop[x3][y3]]-1;
        nowDrop[x3][y3] = 0;
        dropCnt--;

    }
```

```
}

void MainWindow::instSplit2(int x1, int y1,int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1分裂到x2,y2和x3,y3，rev为true时表明撤销分裂的第二步
    playSplit2Sound();
    if(!rev){
        disapDropStack.push(nowDrop[x1][y1]);
        nowDrop[x1][y1]=0;
        notAlone[x2][y2]=notAlone[x3][y3]=false;
        midState[x1][y1]=QPoint(0,0);
    } else {
        midState[x1][y1]=QPoint(1,getMidState(x1,y1,x2,y2));
        notAlone[x2][y2]=notAlone[x3][y3]=true;
        nowDrop[x1][y1]=disapDropStack.top();
        disapDropStack.pop();
    }
}

void MainWindow::instMerge2(int x1, int y1, int x2, int y2, int x3, int y3, bool rev){
    //从x1,y1和x2,y2合并到x3,y3，rev为true是表示撤销合并的第二步
    playMergeSound();
    if(!rev){
        disapDropStack.push(nowDrop[x1][y1]);
        disapDropStack.push(nowDrop[x2][y2]);
        nowDrop[x1][y1]=nowDrop[x2][y2]=0;
        notAlone[x1][y1]=notAlone[x2][y2]=false;
        midState[x3][y3]=QPoint(0,0);
    } else {
        midState[x3][y3]=QPoint(2,getMidState(x2,y2,x3,y3));
        notAlone[x1][y1]=notAlone[x2][y2]=true;
        nowDrop[x2][y2]=disapDropStack.top();
        disapDropStack.pop();
        nowDrop[x1][y1]=disapDropStack.top();
        disapDropStack.pop();
    }
}

int MainWindow::instInput(int x1, int y1, bool rev){
    //输入到x1,y1，rev为true表示撤销输入
    if(inPortList.indexOf(QPoint(x1,y1))==-1)
        return -1;
    if(!rev){
        nowDrop[x1][y1] = ++dropCnt;
        histDrop[x1][y1][nowDrop[x1][y1]] = histDrop[x1][y1][nowDrop[x1][y1]]+1;
    } else {
        histDrop[x1][y1][nowDrop[x1][y1]] = histDrop[x1][y1][nowDrop[x1][y1]]-1;
        nowDrop[x1][y1] = 0;
        --dropCnt;
    }
    return 0;
}


int MainWindow::instOutput(int x1, int y1, bool rev){
```

```
    //从x1,y1输出，rev为true表示撤销输出
    if(outPortList.indexOf(QPoint(x1,y1))==-1)
        return -1;
    if(!rev){
        disapDropStack.push(nowDrop[x1][y1]);
        nowDrop[x1][y1] = 0;
    } else{
        nowDrop[x1][y1] = disapDropStack.top();
        disapDropStack.pop();
    }
    return 0;
}

void MainWindow::handleInst(Instruction inst, bool rev){
    //处理指令，rev为true表示撤销指令
    if(inst.opt==1){
        int x1=inst.arg[0], y1=inst.arg[1], x2=inst.arg[2], y2=inst.arg[3];
        instMove(x1,y1,x2,y2,rev);
    } else if(inst.opt==2){
        int x1=inst.arg[0], y1=inst.arg[1], x2=inst.arg[2], y2=inst.arg[3],
                x3=inst.arg[4], y3=inst.arg[5];
        instSplit1(x1,y1,x2,y2,x3,y3,rev);
    } else if(inst.opt==3){
        int x1=inst.arg[0], y1=inst.arg[1], x2=inst.arg[2], y2=inst.arg[3],
             x3=(x1+x2)/2, y3=(y1+y2)/2;
        instMerge1(x1,y1,x2,y2,x3,y3,rev);
    } else if(inst.opt==4){
        int x1=inst.arg[0], y1=inst.arg[1];
        if(instInput(x1,y1,rev)==-1)
            throw 1;
    } else if(inst.opt==5){
        int x1=inst.arg[0], y1=inst.arg[1];
        if(instOutput(x1,y1,rev)==-1)
            throw 2;
    } else if(inst.opt==6){
        int x1=inst.arg[0], y1=inst.arg[1], x2=inst.arg[2], y2=inst.arg[3],
                x3=inst.arg[4], y3=inst.arg[5];
        instSplit2(x1,y1,x2,y2,x3,y3,rev);
    } else if(inst.opt==7){
        int x1=inst.arg[0], y1=inst.arg[1], x2=inst.arg[2], y2=inst.arg[3],
             x3=(x1+x2)/2, y3=(y1+y2)/2;
        instMerge2(x1,y1,x2,y2,x3,y3,rev);
    }
}

void MainWindow::handleMid(bool rev){
    //处理中间态，当rev为true时逆向处理中间态（即生成中间态）
    if(timeNow==0) return;
    if(!rev){
        for(int i=0;i<instructions[timeNow-1].length();++i){
            Instruction inst = instructions[timeNow-1].at(i);
            if(inst.opt==2){

                inst.opt=6;
```

```
                handleInst(inst, rev);
            } else if(inst.opt==3){
                inst.opt=7;
                handleInst(inst, rev);
            }
        }
    } else{
        for(int i=instructions[timeNow-1].length()-1;i>=0;--i){
            Instruction inst = instructions[timeNow-1].at(i);
            if(inst.opt==2){
                inst.opt=6;
                handleInst(inst, rev);
            } else if(inst.opt==3){
                inst.opt=7;
                handleInst(inst, rev);
            }
        }
    }
}

bool MainWindow::outGridRange(QPoint a){
    //判断a格子是否出界
    int x=a.x(), y=a.y();
    return x<1||y<1||x>col||y>row ;
}

bool MainWindow::washCheckPoint(QPoint a){
    //清洗模式下判断能否走a格子
    int x=a.x(), y=a.y();
    if(outGridRange(a)||ban[x][y]) return false;
    for(int i=0;i<BANDIRNUM;++i){
        int tx=x+dir[i][0], ty=y+dir[i][1];
        if(!outGridRange(QPoint(tx,ty)) && nowDrop[tx][ty]) return false;
    }
    return true;
}

void MainWindow::washAddPath(QPoint s, QPoint t){
    //已经以s为起点BFS过了，现在将s到t的路径加入到washPath
    if(s==t) return ;
    washAddPath(s, bfsPre[t.x()][t.y()]);
    washPath.push_back(t);
    washFlag[t.x()][t.y()]=true;
}

bool MainWindow::washCheckNeed(QPoint s){
    //判断s是否为没有被加入washPath的必要清洗格子
    int x=s.x(), y=s.y();
    return lastVis[x][y]>timeNow && !washFlag[x][y] && histDrop[x][y].size()>0 ;
}

QPoint MainWindow::washBFS(QPoint s, bool *ok){

    //从s点开始BFS，返回最靠近的必要清洗格子或是(col,row)，注意若不能到达(col,row)则返回QPoint(-1,-1)
```

```
    //ok为true表示找到了必要清洗格子（这是为了处理(col,row)恰好是必要清洗格子的情况）
    bfsQue.clear();
    bfsQue.push_back(s);
    memset(bfsDis,0,sizeof(bfsDis));
    bfsDis[s.x()][s.y()]=1;
    QPoint ret=QPoint(-1,-1);
    *ok = false;
    while(!bfsQue.empty()){
        QPoint a=bfsQue.first();
        bfsQue.pop_front();
        for(int i=0;i<MOVEDIRNUM;++i){
            int x=a.x()+dir[i][0], y=a.y()+dir[i][1];
            if(!washCheckPoint(QPoint(x,y))) continue ;
            if(!bfsDis[x][y]){
                bfsDis[x][y]=bfsDis[a.x()][a.y()]+1;
                bfsPre[x][y]=a;
                bfsQue.push_back(QPoint(x,y));
                if(washCheckNeed(QPoint(x,y)) && ret==QPoint(-1,-1)){
                    ret = QPoint(x,y);
                    *ok = true;
                }
            }
        }
    }
    if(bfsDis[col][row] && ret==QPoint(-1,-1)) ret = QPoint(col,row);
    if(ret!=QPoint(-1,-1)) washAddPath(s,ret);
    return ret;
}

bool MainWindow::wash(){
    //用于做单步下的清洗，返回值表示这一步是否进行清洗
    washPath.clear();
    memset(washFlag,false,sizeof(washFlag));
    washPath.push_back(QPoint(1,1));
    if(washCheckPoint(QPoint(1,1))==false) return false;
    QPoint now=QPoint(1,1);
    bool ok=true;
    bool has=false;
    while(ok){
        now = washBFS(now, &ok);
        if(ok) has=true;
        if(now==QPoint(-1,-1)) return false;
    }
    return has;
}

void MainWindow::washNext(){
    washPath.pop_front();
    if(washPath.empty()){
        ui->actionNextStep->setEnabled(true);
        timerWash->stop();
        isWashing=false;

    } else{
```

```
        QPoint a=washPath.first();
        histDrop[a.x()][a.y()].clear();
    }
    update();
}

void MainWindow::mousePressEvent(QMouseEvent *event){
    if(event->button()==Qt::RightButton && !ui->actionRoute->isChecked()){
        int x=event->x(), y=event->y();
        x = (x-leftUp.x())/gridSize+1;
        y = (y-leftUp.y())/gridSize+1;
        y = row-y+1;
        if(x>=1 && x<=col && y>=1 && y<=row && event->x()>leftUp.x() && event->y()>leftUp.y()){
            ban[x][y]^=1;
        }
        update();
    }
}

RouteOperation MainWindow::routeGenOutputOperation(int pt){
    RouteOperation oper;
    oper.opt=5;
    oper.drop1=pt;
    return oper;
}

void MainWindow::routeInit(){
    //TODO 用于route模式下，预处理每个液滴和端口编号，以及预处理routeOperations
    //另外还要处理dropColor
    dropColor.clear();
    memset(routeNowDrop,0,sizeof(routeNowDrop));
    memset(routeInPortOfDrop,0,sizeof(routeInPortOfDrop));
    memset(routeOutPortOfDrop,0,sizeof(routeOutPortOfDrop));
    memset(routeCanOutput,0,sizeof(routeCanOutput));
    memset(routeLastMixTime,0,sizeof(routeLastMixTime));
    int tmpNowDrop[MAXN][MAXN] ;
    routeOperations.clear();
    routeDropNum=0;
    routeInPortList.clear();
    routeOutPortList.clear();
    routeWashInPort  = QPoint(1,row/2)   ;
    routeWashOutPort = QPoint(col,row/2) ;
    for(int i=0;i<=timeLim;++i){
        memcpy(tmpNowDrop, routeNowDrop, sizeof(tmpNowDrop)) ;
        for(int j=0;j<routeInstructions[i].length();++j){
            RouteInstruction inst = routeInstructions[i].at(j) ;
            RouteOperation oper;
            oper.opt = inst.opt;
            oper.oriTime = i;
            if(inst.opt==1){
                //Move
//                int drop = tmpNowDrop[inst.args.at(0)][inst.args.at(1)];

//                routeNowDrop[inst.args.at(2)][inst.args.at(3)] = drop;
```

```
//                if(routeNowDrop[inst.args.at(0)][inst.args.at(1)] == drop)
routeNowDrop[inst.args.at(0)][inst.args.at(1)] = 0 ;
                std::swap(routeNowDrop[inst.args.at(0)][inst.args.at(1)],
routeNowDrop[inst.args.at(2)][inst.args.at(3)]);
            } else if(inst.opt==2){
                //Split
                oper.drop1 = routeNowDrop[inst.args.at(0)][inst.args.at(1)];
                oper.drop2 = routeDropNum+1;
                oper.drop3 = routeDropNum+2;
                routeOperations.append(oper);
                newDropColor(1, dropColor.at(oper.drop1-1)) ;

                routeNowDrop[inst.args.at(0)][inst.args.at(1)]=0;
                routeNowDrop[inst.args.at(2)][inst.args.at(3)]=++routeDropNum;
                routeNowDrop[inst.args.at(4)][inst.args.at(5)]=++routeDropNum;
            } else if(inst.opt==3){
                //Merge
                oper.drop1 = routeNowDrop[inst.args.at(0)][inst.args.at(1)];
                oper.drop2 = routeNowDrop[inst.args.at(2)][inst.args.at(3)];
                oper.drop3 = routeDropNum+1;
                routeOperations.append(oper);
                newDropColor(2, dropColor.at(oper.drop1-1), dropColor.at(oper.drop2-1));

                int x3=(inst.args.at(0)+inst.args.at(2))/2 , y3=
(inst.args.at(1)+inst.args.at(3))/2 ;
                routeNowDrop[inst.args.at(0)][inst.args.at(1)]=0;
                routeNowDrop[inst.args.at(2)][inst.args.at(3)]=0;
                routeNowDrop[x3][y3]=++routeDropNum;
            } else if(inst.opt==4){
                //Input
                int x=inst.args.at(0), y=inst.args.at(1), drop=++routeDropNum;
                routeNowDrop[x][y] = drop;
                if(routeInPortList.indexOf(QPoint(x,y))==-1)
routeInPortList.append(QPoint(x,y));
                int port = routeInPortList.indexOf(QPoint(x,y))+1;
                routeInPortOfDrop[drop] = port;
                newDropColor();

                oper.drop1 = drop;
                routeOperations.append(oper);
            } else if(inst.opt==5){
                //Output
                int x=inst.args.at(0), y=inst.args.at(1), drop=routeNowDrop[x][y];
                routeNowDrop[x][y] = 0 ;
                if(routeOutPortList.indexOf(QPoint(x,y))==-1)
routeOutPortList.append(QPoint(x,y));
                int port = routeOutPortList.indexOf(QPoint(x,y))+1;
                routeOutPortOfDrop[drop] = port;
                routeCanOutput[drop] = true;
            } else if(inst.opt==6){
                //Mix


                int len=inst.args.length(), x=inst.args.at(0), y=inst.args.at(1), drop =
```

```
routeNowDrop[x][y];
                routeLastMixTime[drop] = i;

                oper.drop1 = drop;
                oper.mixLen = len/4;
                routeOperations.append(oper);
            }
        }
    }
    for(int i=0;i<routeOperations.length();++i){
        RouteOperation oper = routeOperations.at(i);
        if(oper.opt==2){
            int pt = oper.drop2;
            if(routeCanOutput[pt] && routeLastMixTime[pt]<=oper.oriTime)
routeOperations.insert(i+1, routeGenOutputOperation(pt)) ;
            pt = oper.drop3;
            if(routeCanOutput[pt] && routeLastMixTime[pt]<=oper.oriTime)
routeOperations.insert(i+1, routeGenOutputOperation(pt)) ;
        } else if(oper.opt==3){
            int pt = oper.drop3;
            if(routeCanOutput[pt] && routeLastMixTime[pt]<=oper.oriTime)
routeOperations.insert(i+1, routeGenOutputOperation(pt)) ;
        } else if(oper.opt==4){
            int pt = oper.drop1;
            if(routeCanOutput[pt] && routeLastMixTime[pt]<=oper.oriTime)
routeOperations.insert(i+1, routeGenOutputOperation(pt)) ;
        } else if(oper.opt==6){
            int pt = oper.drop1;
            if(routeCanOutput[pt] && routeLastMixTime[pt]<=oper.oriTime)
routeOperations.insert(i+1, routeGenOutputOperation(pt)) ;
        }
    }
    routeOutPortList.replace(0,QPoint(col/2,1));
    int tmp = col/routeInPortList.length();
    for(int i=0;i<routeInPortList.length();++i){
        routeInPortList.replace(i, QPoint(i*tmp+1, row));
    }
}

void MainWindow::routeParseLine(QString str){
    //route模式下对str这一行指令进行解析，保证能够解析成功（因为已经调用过parseFile()）
    RouteInstruction inst;
    QStringList argList = str.split(',') ;
    int time=-1, len=argList.length();
    bool ok;
    QString tmp = argList.at(0);
    time = tmp.right(tmp.length()-tmp.lastIndexOf(' ')).toInt(&ok) ;

    if(str.left(str.indexOf(' '))=="Move"){
        inst.opt=1;
    } else if(str.left(str.indexOf(' '))=="Split"){
        inst.opt=2;

    } else if(str.left(str.indexOf(' '))=="Merge"){
```

```
        inst.opt=3;
    } else if(str.left(str.indexOf(' '))=="Input"){
        inst.opt=4;
    } else if(str.left(str.indexOf(' '))=="Output"){
        inst.opt=5;
    } else if(str.left(str.indexOf(' '))=="Mix"){
        inst.opt=6;
    }
    //放入这条指令到routeInstructions
    for(int i=1;i<len;++i){
        QString s = argList.at(i);
        s = s.simplified();
        if(s.endsWith(';')) s = s.left(s.length()-1);
        inst.args.append(s.toInt(&ok));
    }
    routeInstructions[time].append(inst) ;

    //放入拆分Mix的Move指令
    if(inst.opt==6){
        QPoint last;
        RouteInstruction tmpInst;
        tmpInst.opt=1;
        for(int i=1;i<len;i+=2){
            //Mix要拆成Move（便于预处理移动），但也要整体放入（便于预处理Mix），为了保证顺序，先放入整
体的Mix
            QPoint now;
            QString s = argList.at(i).simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            now.setX(s.toInt(&ok));
            s = argList.at(i+1).simplified();
            if(s.endsWith(';')) s = s.left(s.length()-1);
            now.setY(s.toInt(&ok));
            if(i!=1){
                tmpInst.args.append(last.x());
                tmpInst.args.append(last.y());
                tmpInst.args.append(now.x());
                tmpInst.args.append(now.x());
                routeInstructions[time+i/2-1].append(tmpInst) ;
            }
            last=now;
        }
    }
}

void MainWindow::routeParseFile(){
    QFile file(filePath);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QMessageBox::critical(this, "错误", "打开文件失败");
        return ;
    }

    //因为重新打开了文件，记得初始化!!!!

    for(int i=0;i<=MAXTIME;++i)
```

```
        routeInstructions[i].clear();
    QTextStream in(&file);
    QString line = in.readLine(); ;
    while(!line.isNull()){
        if(line==""){
            line = in.readLine();
            continue;
        }
        routeParseLine(line) ;
        line = in.readLine();
    }
    file.close();
}

bool MainWindow::routeCheckConstraint(int drop, QPoint p){
    //检查静态\动态约束
    for(int i=0;i<BANDIRNUM;++i){
        int x=p.x()+dir[i][0], y=p.y()+dir[i][1];
        if(!outGridRange(QPoint(x,y)) && ((nowDrop[x][y]&&nowDrop[x][y]!=drop) ||
(routeLastDrop[x][y]&&routeLastDrop[x][y]!=drop))) return false;
    }
    return true;
}

bool MainWindow::routeCheckPos(int drop, QPoint p, bool washUsedUp=false){
    //返回drop能否走p点。washUsedUp表明为朝出口移动的清洁液滴，此时应无视污染；否则应避免清洁液滴误清洗
    int x=p.x(), y=p.y();
    return routeCheckConstraint(drop, p) && (washUsedUp || histDrop[x][y].size()==0 ||
histDrop[x][y].find(drop)!=histDrop[x][y].end());
}

QPoint MainWindow::routeGetDropPos(int drop){
    return routeDropPos[drop+MAXM];
}

void MainWindow::routeSetDropPos(int drop, QPoint p){
    routeDropPos[drop+MAXM]=p;
}

void MainWindow::routePlaceDrop(int drop, QPoint p){
    //将液滴放在p点，注意若drop<0则为清洗液滴
    int x=p.x(), y=p.y();
    nowDrop[x][y]=drop;
    routeSetDropPos(drop, p);
    if(drop<0){
        if(histDrop[x][y].size()>0 && routeWashDropCap[drop+MAXM]>0){
            routeWashDropCap[drop+MAXM]--;
            histDrop[x][y].clear();
        }
    } else histDrop[x][y][drop] = histDrop[x][y][drop] + 1;
}


void MainWindow::routeMoveDrop(int drop, QPoint p1, QPoint p2){
```

```
    //从p1移动到p2
    routePlaceDrop(drop, p2);
    nowDrop[p1.x()][p1.y()]=0;
    playMoveSound();
}

void MainWindow::routeBFS(int drop, QPoint s, bool washUsedUp=false){
    //从drop所在位置s开始BFS
    bfsQue.clear();
    bfsQue.push_back(s);
    memset(bfsWashDis,0,sizeof(bfsWashDis)) ;
    memset(bfsDis,0,sizeof(bfsDis));
    bfsDis[s.x()][s.y()]=bfsWashDis[s.x()][s.y()]=1;
    while(!bfsQue.empty()){
        QPoint a=bfsQue.first();
        bfsQue.pop_front();
        for(int i=0;i<MOVEDIRNUM;++i){
            int x=a.x()+dir[i][0], y=a.y()+dir[i][1];
            if(!outGridRange(QPoint(x,y)) && routeCheckConstraint(drop, QPoint(x,y)) &&
!routeWashBan[x][y]) {
                if(!bfsWashDis[x][y]){
                    bfsWashDis[x][y]=bfsDis[a.x()][a.y()]+1;
                    bfsPre[x][y]=a;
                }
            }
            if(outGridRange(QPoint(x,y)) || !routeCheckPos(drop, QPoint(x,y), washUsedUp) ||
routeBfsBan[x][y]) continue;
            if(drop<0 && routeWashBan[x][y]) continue; //禁止清洗液滴进入
            if(!bfsDis[x][y]){
                bfsDis[x][y]=bfsDis[a.x()][a.y()]+1;
                bfsPre[x][y]=a;
                bfsQue.push_back(QPoint(x,y));
            }
        }
    }
}

QPoint MainWindow::routeGetNextPos(QPoint s, QPoint t, bool flag=false){
    //已经从s调用BFS过，想求得走到t的路径上下一个点，flag为true时记录路径
    if(s==t){
        // t就等于s，特殊情况
        if(flag) routeBfsPath.append(s);
        return s;
    }
    else if(bfsPre[t.x()][t.y()]==s) {
        if(flag) {routeBfsPath.append(s); routeBfsPath.append(t);}
        return t;
    }
    QPoint ret = routeGetNextPos(s, bfsPre[t.x()][t.y()], true) ;
    if(flag) routeBfsPath.append(t) ;
    return ret;
}
```

```
void MainWindow::routeGetPath(QPoint s, QPoint t){
    //已经从s调用BFS过，想得到s到t的路径，存储在routeBfsPath中
    routeBfsPath.clear();
    routeGetNextPos(s, t, true);
}

void MainWindow::debugOper(RouteOperation oper){
    debug(QString("opt:%1, drop1:%2, drop2:%3, drop3:%4, oriTime:%5,
mixLen:%6").arg(oper.opt).arg(oper.drop1).arg(oper.drop2).arg(oper.drop3).arg(oper.oriTime).arg(
oper.mixLen)) ;
}

int MainWindow::calcChebyshevDis(QPoint a, QPoint b){
    return std::abs(a.x()-b.x()) + std::abs(a.y()-b.y()) ;
}

int MainWindow::routeCalcBlockValue(QPoint p){
    //计算p点的阻塞值，即到最近端口的切比雪夫距离的相反数乘上routeBlockK，即求一个最大值，注意到清洗液滴
出入口要再除以系数2
    int ret=-INF;
    for(int i=0;i<routeInPortList.length();++i){
        QPoint a=routeInPortList.at(i) ;
        ret = std::max(ret, -calcChebyshevDis(a, p) * routeBlockK) ;
    }
    for(int i=0;i<routeOutPortList.length();++i){
        QPoint a=routeOutPortList.at(i) ;
        ret = std::max(ret, -calcChebyshevDis(a, p) * routeBlockK) ;
    }
    ret = std::max(ret, -calcChebyshevDis(routeWashInPort, p) * routeBlockK / 2) ;
    ret = std::max(ret, -calcChebyshevDis(routeWashOutPort, p) * routeBlockK / 2) ;
    return ret;
}

bool MainWindow::routeGetMergeTarget(int drop1, QPoint p1, int drop2, QPoint p2){
    routeMergeTarget1=routeMergeTarget2=QPoint(-1,-1);
    QList<QPoint> tmpPath1, tmpPath2;
    int minValue = INF ;
    for(int x=2;x<col;++x){      //Merge的中间点显然不在左右边界
        for(int y=2;y<=row;++y){ //不在最下面一排进行Merge，且只进行左右Merge
            if(!histDrop[x][y].size() && !histDrop[x][y-1].size()){ //保证下面的点可以走

                memset(routeBfsBan,0,sizeof(routeBfsBan));
                routeBfsBan[x][y-1]=routeBfsBan[x][y]=true;
             //   debug(QString("TTT1, (%1,%2)").arg(x).arg(y)) ;
                routeBFS(drop1, p1);
                if(bfsDis[x-1][y]){
                    routeGetPath(p1, QPoint(x-1,y)) ;
                    tmpPath1 = routeBfsPath ;
                    for(int i=0;i<routeBfsPath.length();++i){
                        QPoint t = routeBfsPath.at(i);
                        routeBfsBan[t.x()][t.y()]=true;
                    }

                    routeBFS(drop2, p2) ;
```

```
                    if(bfsDis[x+1][y]){
                        routeGetPath(p2, QPoint(x+1,y)) ;
                        tmpPath2 = routeBfsPath ;
                        int value = (int)std::sqrt((tmpPath1.length()-1)*(tmpPath1.length()-1)+
(tmpPath2.length()-1)*(tmpPath2.length()-1))+routeCalcBlockValue(QPoint(x,y)) ;
                //        debug(QString("TTT2, value:%1").arg(value)) ;
                        if(value<minValue){
                            minValue=value;
                            routeMergeTarget1 = QPoint(x-1,y) ;
                            routeMergeTarget2 = QPoint(x+1,y) ;
                            routeMergePath1 = tmpPath1 ;
                            routeMergePath2 = tmpPath2 ;
                            //debug("updated");
                        }
                    }
                }

                memset(routeBfsBan,0,sizeof(routeBfsBan));
                routeBfsBan[x][y-1]=routeBfsBan[x][y]=true;
                routeBFS(drop1, p1);
                if(bfsDis[x+1][y]){
                    routeGetPath(p1, QPoint(x+1,y)) ;
                    tmpPath1 = routeBfsPath ;
                    for(int i=0;i<routeBfsPath.length();++i){
                        QPoint t = routeBfsPath.at(i);
                        routeBfsBan[t.x()][t.y()]=true;
                    }
                    routeBFS(drop2, p2) ;
                    if(bfsDis[x-1][y]){
                        routeGetPath(p2, QPoint(x-1,y)) ;
                        tmpPath2 = routeBfsPath ;
                        int value = (int)std::sqrt((tmpPath1.length()-1)*(tmpPath1.length()-1)+
(tmpPath2.length()-1)*(tmpPath2.length()-1))+routeCalcBlockValue(QPoint(x,y)) ;
                  //      debug(QString("TTT3, value:%1").arg(value)) ;
                        if(value<minValue){
                            minValue=value;
                            routeMergeTarget1 = QPoint(x+1,y) ;
                            routeMergeTarget2 = QPoint(x-1,y) ;
                            routeMergePath1 = tmpPath1 ;
                            routeMergePath2 = tmpPath2 ;
                  //          debug("updated");
                        }
                    }
                }
                memset(routeBfsBan,0,sizeof(routeBfsBan));
            }
        }
    }
    if(routeMergeTarget1!=QPoint(-1,-1)){
        //TODO!!!!!!!!!!!!!!!!!!!!!!!!!!!!!记得重置routeWashBan
        for(int i=0;i<BANDIRNUM;++i){
            int x=routeMergeTarget1.x()+dir[i][0], y=routeMergeTarget1.y()+dir[i][1];

            if(!outGridRange(QPoint(x,y))) routeWashBan[x][y]=true;
```

```
            x=routeMergeTarget2.x()+dir[i][0]; y=routeMergeTarget2.y()+dir[i][1];
            if(!outGridRange(QPoint(x,y))) routeWashBan[x][y]=true;
        }
        return true;
    } else return false;
}

bool MainWindow::routeGetMixTarget(int drop, QPoint p, int mixLen){
    routeMixTarget = QPoint(-1,-1); //注意该Target为左下角，且可能横放或竖放
    routeMixStart = QPoint(-1,-1);  //进入环的起始点
    int minValue = INF;
    routeBFS(drop, p);
    for(int i=1;i<=col;++i){
        for(int j=1;j<=row;++j){
            //枚举Target
            //横放
            bool fail=false;
            int value=0, minDis=INF;
            QPoint tmpStart;
            for(int t=1;t<=mixLen;++t){
                if(outGridRange(QPoint(i+t-1,j)) || outGridRange(QPoint(i+t-1,j+1)) ||
!bfsDis[i+t-1][j] || !bfsDis[i+t-1][j+1]){
                    fail=true ;
                    break;
                }
                value += routeCalcBlockValue(QPoint(i+t-1,j)) + routeCalcBlockValue(QPoint(i+t-
1,j+1)) ;
                if(bfsDis[i+t-1][j]<minDis){
                    minDis=bfsDis[i+t-1][j];
                    tmpStart=QPoint(i+t-1,j);
                }
                if(bfsDis[i+t-1][j+1]<minDis){
                    minDis=bfsDis[i+t-1][j+1];
                    tmpStart=QPoint(i+t-1,j+1);
                }
            }
            if(!fail){
                value += minDis*2*mixLen;
                if(value < minValue){
                    minValue = value;
                    routeMixPath.clear();
                    routeMixStart=tmpStart;
                    routeMixTarget=QPoint(i,j);
                    routeGetPath(p, routeMixStart) ;
                    routeMixPath = routeBfsPath ;
                    int x = routeMixStart.x() , y = routeMixStart.y() , oy = ((y==j) ? (j+1) :
(j)) ;
                    for(int tx=x+1;tx<=i+mixLen-1;++tx) routeMixPath.append(QPoint(tx, y));
                    for(int tx=i+mixLen-1;tx>=i;--tx)   routeMixPath.append(QPoint(tx, oy));
                    for(int tx=i;tx<=x;++tx)            routeMixPath.append(QPoint(tx, y));
                }
            }
```

```
            //竖放
            fail=false;
            value=0; minDis=INF;
            tmpStart=QPoint(-1,-1);
            for(int t=1;t<=mixLen;++t){
                if(outGridRange(QPoint(i,j+t-1)) || outGridRange(QPoint(i+1,j+t-1)) ||
!bfsDis[i][j+t-1] || !bfsDis[i+1][j+t-1]){
                    fail=true ;
                    break;
                }
                value += routeCalcBlockValue(QPoint(i,j+t-1)) +
routeCalcBlockValue(QPoint(i+1,j+t-1)) ;
                if(bfsDis[i][j+t-1]<minDis){
                    minDis=bfsDis[i][j+t-1];
                    tmpStart=QPoint(i,j+t-1);
                }
                if(bfsDis[i+1][j+t-1]<minDis){
                    minDis=bfsDis[i+1][j+t-1];
                    tmpStart=QPoint(i+1,j+t-1);
                }
            }
            if(!fail){
                value += minDis*2*mixLen;
                if(value < minValue){
                    minValue = value;
                    routeMixPath.clear();
                    routeMixStart=tmpStart;
                    routeMixTarget=QPoint(i,j);
                //    debug(QString("routeMix. 竖放 value:%1 routeMixStart:(%2,%3)
routeMixTarget:(%4,%5)").arg(value).arg(routeMixStart.x())
                 //
.arg(routeMixStart.y()).arg(routeMixTarget.x()).arg(routeMixTarget.y())) ;
                    routeGetPath(p, routeMixStart) ;
                    routeMixPath = routeBfsPath ;
                    int x = routeMixStart.x() , y = routeMixStart.y() , ox = ((x==i) ? (i+1) :
(i)) ;
                    for(int ty=y+1;ty<=j+mixLen-1;++ty) routeMixPath.append(QPoint(x,ty));
                    for(int ty=j+mixLen-1;ty>=j;--ty)   routeMixPath.append(QPoint(ox,ty));
                    for(int ty=j;ty<=y;++ty)            routeMixPath.append(QPoint(x,ty));
                }
            }
        }
    }
    if(routeMixTarget==QPoint(-1,-1)) return false;
    else return true;
}

bool MainWindow::routeHandleWashDrop(){
    //处理清洁液滴，若清洁液滴有任何变动均返回true
    bool res=false, ret=false; //res为有无剩余容量，ret为返回值
    QList<int>::iterator it;
    for(it=routeWashDrops.begin();it!=routeWashDrops.end();++it){

        int drop = *it ;
```

```
        QPoint p = routeGetDropPos(drop) ;
       // debug(QString("wash. drop:%1 nowPos:(%2,%3)
cap:%4").arg(drop).arg(p.x()).arg(p.y()).arg(routeWashDropCap[drop+MAXM])) ;
        if(routeWashDropCap[drop+MAXM]==0){
            //应当到清洁液滴出口
            if(p == routeWashOutPort){

                nowDrop[routeWashOutPort.x()][routeWashOutPort.y()]=0;
                it = routeWashDrops.erase(it) ;
                -- it;
                //debug(QString("!!!!!!!!!!!!!!!DELETE drop:%1  p:
(%2,%3)").arg(drop).arg(p.x()).arg(p.y())) ;
                ret = true;
                continue ;
            } else{
                routeBFS(drop, p, true) ;
                //debug(QString("washDrop:%1(usedUp)
canGo:%2").arg(routeWashDrops.at(i)).arg(bfsDis[routeWashOutPort.x()][routeWashOutPort.y()])) ;
                if(bfsDis[routeWashOutPort.x()][routeWashOutPort.y()]){
                    QPoint nxt = routeGetNextPos(p, routeWashOutPort) ;
                    routeMoveDrop(drop, p, nxt) ;
                    ret = true;
                }
            }
        } else{
            res = true ;
            routeBFS(drop, p, false) ;
            int minValue = INF ;
            QPoint target=QPoint(-1,-1);
            debug(QString("routeMoveSuc:%1
routeCriticalDrop:%2").arg(routeMoveSuc).arg(routeCriticalDrop)) ;
            for(int x=1;x<=col;++x){
                for(int y=1;y<=row;++y){
                    if(bfsWashDis[x][y] && histDrop[x][y].size()>0){
                        int value = bfsWashDis[x][y]-1-routeCalcBlockValue(QPoint(x,y)) ;
                        if(!routeMoveSuc && routeCriticalDrop!=-1){
                            value += routeBlockK *
(calcChebyshevDis(routeGetDropPos(routeCriticalDrop), QPoint(x,y))) ;
                            if(routeOperations.at(routeOperPoint).opt==5)
                                value += routeBlockK *
calcChebyshevDis(routeOutPortList.at(routeOutPortOfDrop[routeCriticalDrop]-1), QPoint(x,y)) ;
                        }
                        if(value < minValue){
                            minValue = value;
                            target = QPoint(x,y) ;
                        }
                    }
                }
            }
            if(target!=QPoint(-1,-1)){
                QPoint nxt = routeGetNextPos(p, target) ;


          //      debug(QString("wash. drop:%1 nxtPos:(%2,%3) target:
```

```
(%4,%5)").arg(drop).arg(nxt.x()).arg(nxt.y()).arg(target.x()).arg(target.y())) ;
                routeMoveDrop(drop, p, nxt) ;
                if(routeWashDropCap[drop+MAXM]==0) res=false;
                ret = true;
            }
        }
    }
    //debug(QString("wash. res:%1 routeWashDropCnt:%2").arg(res).arg(routeWashDropCnt)) ;
    if(!res && routeCheckPos(-MAXM, routeWashInPort, true) && !nowDrop[routeWashInPort.x()]
[routeWashInPort.y()]){
        routeBFS(-MAXM, routeWashInPort) ;
        bool suc=false;
        for(int x=1;x<=col;++x){
            for(int y=1;y<=row;++y){
                if(bfsWashDis[x][y] && histDrop[x][y].size()>0){
                    suc=true;
                }
            }
        }
        if(suc){
            routeWashDrops.append(--routeWashDropCnt) ;
            routeWashDropCap[routeWashDropCnt+MAXM] = 3;
            routePlaceDrop(routeWashDropCnt, routeWashInPort) ;
            ret = true;
        }
    }
    return ret;
}

bool MainWindow::routeGetSplitTarget(int drop, QPoint p){
    int minValue = INF;
    routeSplitTarget = QPoint(-1,-1);
    routeSplitPath.clear() ;
    for(int x=2;x<col;++x){
        for(int y=2;y<=row;++y){
            if(histDrop[x-1][y].size()>0 || histDrop[x+1][y].size()>0 || histDrop[x-1][y-
1].size()>0 || histDrop[x+1][y-1].size()>0)
                continue ;
            memset(routeBfsBan,0,sizeof(routeBfsBan)) ;
            routeBfsBan[x-1][y]=routeBfsBan[x+1][y]=routeBfsBan[x-1][y-1]=routeBfsBan[x+1][y-
1]=true ;
            routeBFS(drop, p);
            if(bfsDis[x][y]){
                int value = bfsDis[x][y]-1+routeCalcBlockValue(QPoint(x,y)) ;
                if(value < minValue){
                    minValue = value ;
                    routeSplitTarget = QPoint(x,y) ;
                }
            }
            memset(routeBfsBan,0,sizeof(routeBfsBan));
        }
    }

    if(routeSplitTarget!=QPoint(-1,-1)){
```

```
        for(int k=0;k<BANDIRNUM;++k){
            int x=routeSplitTarget.x()+dir[k][0], y=routeSplitTarget.y()+dir[k][1];
            routeWashBan[x][y]=true;
        }
        routeGetPath(p, routeSplitTarget) ;
        routeSplitPath = routeBfsPath ;
        return true;
    }
    return false ;
}

bool MainWindow::routeGetOutputTarget(int drop, QPoint p){
    routeBFS(drop, p);
    routeOutputPath.clear();
    QPoint outPort = routeOutPortList.at(routeOutPortOfDrop[drop]-1) ;
    if(bfsDis[outPort.x()][outPort.y()]){
        routeGetPath(p, outPort) ;
        routeOutputPath = routeBfsPath ;
        for(int i=0;i<routeOutputPath.length();++i){
            QPoint p=routeOutputPath.at(i) ;
            for(int k=0;k<BANDIRNUM;++k){
                int x=p.x()+dir[k][0], y=p.y()+dir[k][1] ;
                if(!outGridRange(QPoint(x,y)))
                    routeWashBan[x][y]=true ;
            }
        }
        return true;
    }
    return false;
    //TODO记得清空 routeWashBan
}

void MainWindow::routeNextStep(){
    if(routeOperPoint >= routeOperations.length()) return ;

    ui->actionNextStep->setEnabled(false);
    memcpy(routeLastDrop, nowDrop, sizeof(nowDrop)) ;
    memset(routeMoved,0,sizeof(routeMoved));
    //debug("test1");
    RouteOperation oper = routeOperations.at(routeOperPoint);
    routeCriticalDrop=-1;
    debugOper(oper);
    //debug("test2");
    routeMoveSuc = false;

    if(oper.opt==2){
        //Split
        int drop1=oper.drop1, drop2=oper.drop2, drop3=oper.drop3;
        routeCriticalDrop=drop1;
        QPoint p = routeGetDropPos(drop1) ;
        debug(QString("Split. drop:%1 routeSplitTarget:
(%2,%3)").arg(drop1).arg(routeSplitTarget.x()).arg(routeSplitTarget.y())) ;

        if(routeSplitTarget!=QPoint(-1,-1) || routeGetSplitTarget(drop1, p)){
```

```
            QPoint p1 = routeSplitTarget , p2 = routeSplitTarget + QPoint(-1,0) , p3 =
routeSplitTarget + QPoint(1,0) ;
            if(routeSplitMid){
                notAlone[p2.x()][p2.y()] = notAlone[p3.x()][p3.y()] = false ;
                midState[p1.x()][p1.y()] = QPoint(0, 0) ;
                nowDrop[p1.x()][p1.y()] = 0 ;
                routeMoved[drop1] = routeMoved[drop2] = routeMoved[drop3] = true ;
                routeSplitMid=false;
                routeOperPoint++;
                memset(routeWashBan,0,sizeof(routeWashBan)) ;
                routeSplitTarget=QPoint(-1,-1);
                routeSplitPath.clear();
                playSplit2Sound();
                routeMoveSuc = true;
            } else if(routeSplitPath.length()==1){
                if(routeCheckPos(drop1, p2) && routeCheckPos(drop1, p3)){
                    routeSplitMid=true;
                    routePlaceDrop(drop2, routeSplitTarget+QPoint(-1,0)) ;
                    routePlaceDrop(drop3, routeSplitTarget+QPoint(1,0)) ;
                    routeMoved[drop1] = routeMoved[drop2] = routeMoved[drop3] = true ;
                    notAlone[p2.x()][p2.y()] = notAlone[p3.x()][p3.y()] = true ;
                    midState[p1.x()][p1.y()] = QPoint(1, 1) ;
                    playSplit1Sound();
                    routeMoveSuc = true;
                }
            } else if(routeSplitPath.length()>1 && routeCheckPos(drop1, routeSplitPath.at(1))){
                routeMoveDrop(drop1, routeSplitPath.at(0), routeSplitPath.at(1)) ;
                routeSplitPath.pop_front() ;
                routeMoved[drop1] = true ;
                routeMoveSuc = true;
            }
        }
    } else if(oper.opt==3){
        //Merge
        int drop1=oper.drop1, drop2=oper.drop2, drop3=oper.drop3;
        routeCriticalDrop=drop1;
        if(routeMergeTarget1!=QPoint(-1,-1) || routeGetMergeTarget(drop1,
routeGetDropPos(drop1), drop2, routeGetDropPos(drop2))){
            QPoint p1=routeMergeTarget1, p2=routeMergeTarget2, p3=(p1+p2)/2 ;
            //有Target
            debug(QString("Merge. Target1:(%1,%2)  Target2:(%3,%4)  Len1:%5
Len2:%6").arg(p1.x()).arg(p1.y()).arg(p2.x()).arg(p2.y())
                  .arg(routeMergePath1.length()).arg(routeMergePath2.length()));
            if(routeMergeMid){
          //      debug("test3");
                routeMergeMid=false;
                nowDrop[p1.x()][p1.y()]=nowDrop[p2.x()][p2.y()]=0;
                midState[p3.x()][p3.y()] = QPoint(0, 0) ;
                notAlone[p1.x()][p1.y()]=notAlone[p2.x()][p2.y()]=false ;
                memset(routeWashBan,0,sizeof(routeWashBan));
                routeMergePath1.clear();
                routeMergePath2.clear();

                routeMergeTarget1=routeMergeTarget2=QPoint(-1,-1);
```

```
                routeOperPoint++;
                routeMoved[drop1+MAXM] = routeMoved[drop2+MAXM] = routeMoved[drop3+MAXM] = true
;
                routeMoveSuc = true;
                playMergeSound();
            } else if(routeMergePath1.length()==1 && routeMergePath2.length()==1){
          //      debug("test4");
                routeMergeMid=true;
                routePlaceDrop(drop3, p3);
                midState[p3.x()][p3.y()] = QPoint(2, 1) ;
                notAlone[p1.x()][p1.y()]=notAlone[p2.x()][p2.y()]=true ;
                routeMoved[drop1+MAXM] = routeMoved[drop2+MAXM] = routeMoved[drop3+MAXM] = true
;
                routeMoveSuc = true;
            }
            else{
          //      debug("test5");
                if(routeMergePath1.length()>1 && routeCheckPos(drop1, routeMergePath1.at(1))){
                    routeMoveDrop(drop1,routeMergePath1.at(0),routeMergePath1.at(1)) ;
                    routeMergePath1.pop_front();
                    routeMoved[drop1+MAXM] = true ;
                    routeMoveSuc = true;
                }
                if(routeMergePath2.length()>1 && routeCheckPos(drop2, routeMergePath2.at(1))){
                    routeMoveDrop(drop2,routeMergePath2.at(0),routeMergePath2.at(1)) ;
                    routeMergePath2.pop_front();
                    routeMoved[drop2+MAXM] = true ;
                    routeMoveSuc = true;
                }
            }
        }
    } else if(oper.opt==4){
        //Input
        int drop=oper.drop1;
        QPoint p=routeInPortList.at(routeInPortOfDrop[drop]-1);
        if(routeCheckPos(drop, p) && !nowDrop[p.x()][p.y()]){
            routePlaceDrop(drop, p);
            routeOperPoint++;
            routeMoved[drop+MAXM] = true ;
            routeMoveSuc = true;
        }
    } else if(oper.opt==5){
        //Output
        int drop=oper.drop1;
        routeCriticalDrop=drop;
        QPoint p=routeGetDropPos(drop), outPort=routeOutPortList.at(routeOutPortOfDrop[drop]-1);
        if(routeIsOutputing || routeGetOutputTarget(drop, p)){
            if(routeOutputPath.length()==1){
                nowDrop[outPort.x()][outPort.y()]=0 ;
                routeIsOutputing=false;
                routeOperPoint++;
                memset(routeWashBan,0,sizeof(routeWashBan)) ;

                routeMoved[drop+MAXM]=true;
```

```
                routeMoveSuc=true;
            } else{
                if(routeCheckPos(drop, routeOutputPath.at(1))){
                    routeMoveDrop(drop, routeOutputPath.at(0), routeOutputPath.at(1)) ;
                    routeOutputPath.pop_front();
                    routeMoved[drop+MAXM]=true;
                    routeMoveSuc=true;
                }
            }
        } else {
            debug("Move idle.") ;
            for(int x=1;x<=col;++x){
                for(int y=1;y<=row;++y){
                    if(nowDrop[x][y]>0 && nowDrop[x][y]!=drop && (calcChebyshevDis(QPoint(x,y),
p)<=4 || calcChebyshevDis(QPoint(x,y), outPort)<=4)){
                        int drop2=nowDrop[x][y] ;
                        debug(QString("Idle. drop2:%1 pos:(%2,%3)").arg(drop2).arg(x).arg(y)) ;
                        if(!routeMoved[drop2+MAXM]){
                            routeMoved[drop2+MAXM]=true;
                            int minValue = INF ;
                            QPoint nxt = QPoint(-1,-1);
                            for(int k=MOVEDIRNUM-1;k>=0;--k){
                                int tx=x+dir[k][0], ty=y+dir[k][1];
                                if(!outGridRange(QPoint(tx,ty)) && routeCheckPos(drop2,
QPoint(tx,ty))){
                                    int value = routeCalcBlockValue(QPoint(tx,ty)) - routeBlockK
* 2 * calcChebyshevDis(QPoint(tx,ty), p) ;
                                    if(value < minValue){
                                        minValue = value ;
                                        nxt = QPoint(tx,ty);
                                    }
                                }
                            }
                            if(nxt!=QPoint(-1,-1)){
                                routeMoveDrop(drop2, QPoint(x,y), nxt) ;
                                routeMoved[drop2+MAXM]=true;
                                //     routeMoveSuc = true;
                            }
                        }
                    }
                }
            }
        }
    } else if(oper.opt==6){
        //Mix
        int drop=oper.drop1, mixLen=oper.mixLen;
        routeCriticalDrop=drop;
        QPoint p=routeGetDropPos(drop) ;
        debug(QString("routeMixPath.length():%1
routeIsMixing:%2").arg(routeMixPath.length()).arg(routeIsMixing));
        if(routeIsMixing || routeGetMixTarget(drop, p, mixLen)){
            routeIsMixing = true;

            debug(QString("routeMixStart:(%1,%2)
```

```
routeMixTarget(%3,%4)").arg(routeMixStart.x()).arg(routeMixStart.y()).arg(routeMixTarget.x()).ar
g(routeMixTarget.y())) ;
            if(routeMixPath.length()>1)  debug(QString("Mix. Next Point:
(%1,%2)").arg(routeMixPath.at(1).x()).arg(routeMixPath.at(1).y()));
            if(routeMixPath.length()>1 && routeCheckPos(drop, routeMixPath.at(1))){
                routeMoveDrop(drop,routeMixPath.at(0),routeMixPath.at(1)) ;
                routeMixPath.pop_front();
                routeMoved[drop+MAXM] = true;
                routeMoveSuc = true;
            }
            if(routeMixPath.length()==1){
                routeIsMixing = false;
                memset(routeWashBan,0,sizeof(routeWashBan));
                routeOperPoint++;
            }
        }
    }
    bool washSuc = routeHandleWashDrop();
    if(!routeMoveSuc && !washSuc){
        for(int x=1;x<=col;++x){
            for(int y=1;y<=row;++y){
                if(nowDrop[x][y]){
                    int drop=nowDrop[x][y] ;
                    if(!routeMoved[drop+MAXM]){
                        routeMoved[drop+MAXM]=true;
                        int minValue = INF ;
                        QPoint nxt = QPoint(-1,-1);
                        for(int k=MOVEDIRNUM-1;k>=0;--k){
                            int tx=x+dir[k][0], ty=y+dir[k][1];
                            if(!outGridRange(QPoint(tx,ty)) && routeCheckPos(drop,
QPoint(tx,ty))){
                                int value = routeCalcBlockValue(QPoint(tx,ty)) ;
                                if(value < minValue){
                                    minValue = value ;
                                    nxt = QPoint(tx,ty);
                                }
                            }
                        }
                        if(nxt!=QPoint(-1,-1))
                            routeMoveDrop(drop, QPoint(x,y), nxt) ;
                    }
                }
            }
        }
    }
    ui->labelCurTime->setNum(++timeNow);
    if(!playingAll) ui->actionNextStep->setEnabled(true);
    update();
}

void MainWindow::on_actionNextStep_triggered()
{

    if(!playingAll && ui->actionNextStep->isEnabled()==false) return;
```

```
    if((!ui->actionRoute->isChecked() && timeNow==timeLim) || (ui->actionRoute->isChecked() &&
routeOperPoint>=routeOperations.length())){
        on_actionPause_triggered();
        return;
    }

    if(ui->actionRoute->isChecked()){
        routeNextStep();
        return ;
    }

    ui->actionNextStep->setEnabled(false);
    int tmpDrop[MAXN][MAXN];
    QPoint tmpMidState[MAXN][MAXN];
    memcpy(tmpDrop, nowDrop, sizeof(tmpDrop));
    memcpy(tmpMidState, midState, sizeof(tmpMidState));
    Instruction inst;
    int failCnt=0;
    for(int i=1;i<=col;++i){
        for(int j=1;j<=row;++j){
            if(histDrop[i][j].size()>1){
                failCnt++;
            }
        }
    }
    //处理midState（实际上一个更优的做法是直接把合并和分裂都拆成两条来做）
    handleMid(false);

    //处理指令
    int now;
    try {
        for(now=0;now<instructions[timeNow].length();++now){
            handleInst(instructions[timeNow].at(now), false);
        }
        for(int x1=1;x1<=col;++x1){
            for(int y1=1;y1<=row;++y1){
                if(nowDrop[x1][y1] && !midState[x1][y1].x()){
                    for(int d=0;d<BANDIRNUM;++d){
                        int x2=x1+dir[d][0], y2=y1+dir[d][1];
                        if(x2<1||y2<1||x2>col||y2>row||midState[x2][y2].x()) continue;
                        if(nowDrop[x2][y2] && nowDrop[x2][y2]!=nowDrop[x1][y1]) {
                            qDebug() << x1 << ' ' << y1 << ' ' << x2 << ' ' << y2 << '\n';
                            throw 3;
                        }
                        if(tmpDrop[x2][y2] && tmpDrop[x2][y2]!=nowDrop[x1][y1] &&
!tmpMidState[x2][y2].x() && tmpMidState[x1][y1].y()!=d/2+1) {
                            //qDebug() << x1 << ' ' << y1 << ' ' << x2 << ' ' << y2 << '\n';
                            //qDebug() << tmpMidState[x1][y1].x() << ' ' << d << ' ' <<
tmpMidState[x2][y2].x() << '\n' ;
                            throw 4;
                        }
                    }

                }
```

```
            }
        }
    } catch (int a) {
        if(a==1){
            QMessageBox::critical(this, "错误", "输入不在端口相邻处");
        } else if(a==2){
            QMessageBox::critical(this, "错误", "输出不在端口相邻处");
        } else if(a==3){
            QMessageBox::critical(this, "错误", "不满足静态约束");
        } else if(a==4){
            QMessageBox::critical(this, "错误", "不满足动态约束");
        }
        for(now=now-1;now>=0;--now){
            handleInst(instructions[timeNow].at(now), true);
        }
        handleMid(true);
        update();
        on_actionPause_triggered();
        return;
    }

    ui->labelCurTime->setNum(++timeNow);
    update();

    for(int i=1;i<=col;++i){
        for(int j=1;j<=row;++j){
            if(histDrop[i][j].size()>1){
                failCnt--;
            }
        }
    }
    if(failCnt<0 && ui->actionWash->isChecked()) QMessageBox::warning(this, "警告", "清洗失败，液滴出现污染", "忽略");

    //清洗
    if(ui->actionWash->isChecked()){
        if(wash()){
            timerWash->start(40);
            isWashing=true;
            histDrop[1][1].clear();
            return;
        }
    }

    if(!playingAll) ui->actionNextStep->setEnabled(true);
}

void MainWindow::on_actionPreviousStep_triggered()
{
    if(timeNow==0)
        return;
    Instruction inst;

    ui->labelCurTime->setNum(--timeNow);
```

```
    //处理指令
    int now;
    for(now=instructions[timeNow].length()-1;now>=0;--now){
        handleInst(instructions[timeNow].at(now), true);
    }

    //处理midState
    handleMid(true);
    update();
}

void MainWindow::on_actionReset_triggered()
{
    init();
}

void MainWindow::on_actionPlayAll_triggered()
{
    playingAll=true;
    timerPlayAll->start(1000);
    ui->actionNextStep->setEnabled(false);
    ui->actionPreviousStep->setEnabled(false);
}

void MainWindow::on_actionPause_triggered()
{
    playingAll=false;
    timerPlayAll->stop();
    ui->actionNextStep->setEnabled(true);
    if(!ui->actionWash->isChecked()) ui->actionPreviousStep->setEnabled(true);
}

void MainWindow::on_actionWash_triggered()
{
    init();
    if(!ui->actionWash->isChecked()){
        ui->actionRoute->setChecked(false);
        ui->actionPreviousStep->setEnabled(true);
    } else{
        ui->actionPreviousStep->setEnabled(false);
    }
    update();
}

void MainWindow::on_actionRoute_triggered()
{
    if(ui->actionRoute->isChecked()){
        ui->actionWash->setChecked(true);
        ui->actionPreviousStep->setEnabled(false);
        routeInit();
        //TODO

    } else{
```

```
        ui->actionPreviousStep->setEnabled(true);
    }
    init();
}
```