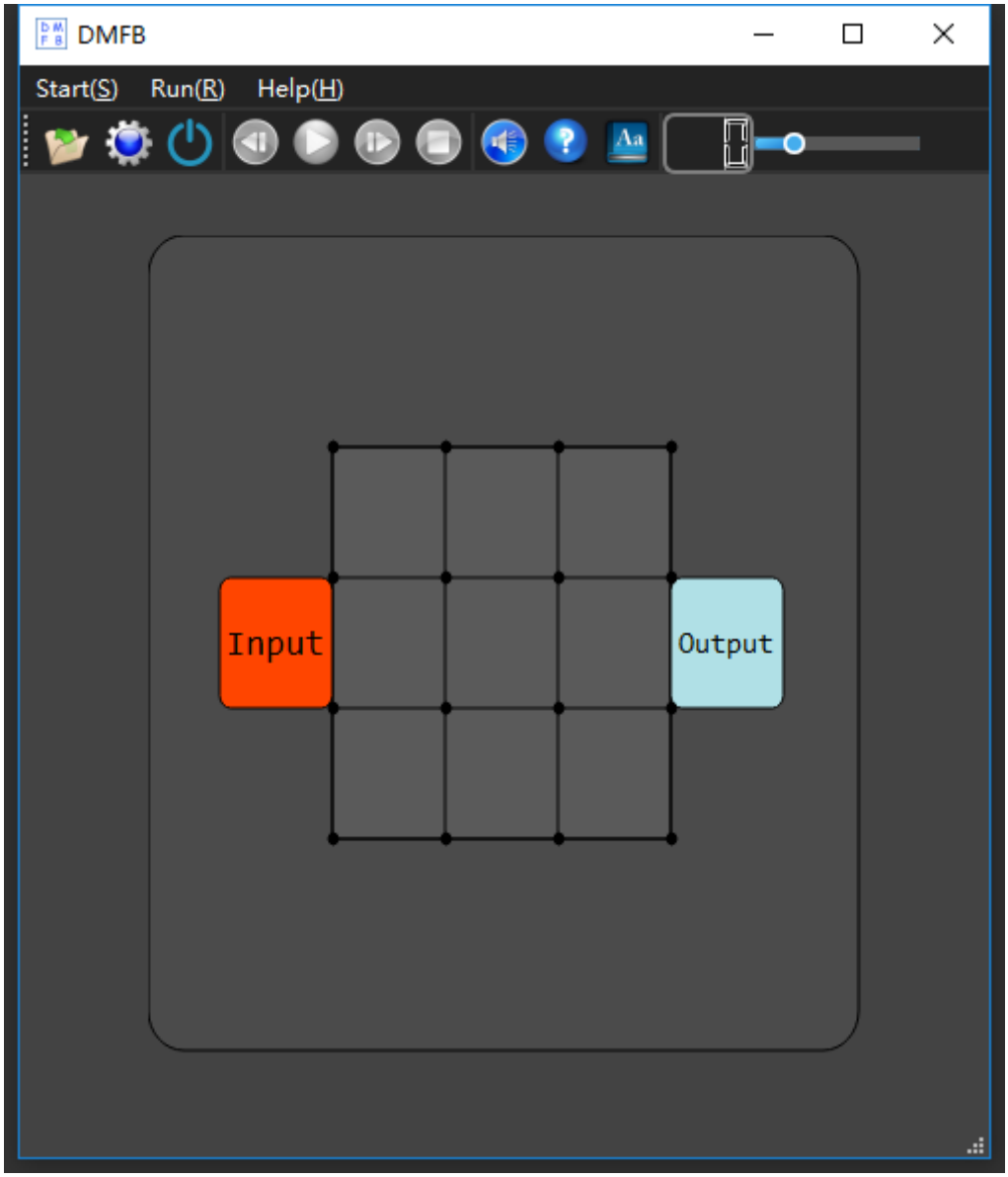
DMFB仿真程序

计83 饶淙元 2017011285

1.展示说明

1.1 主界面

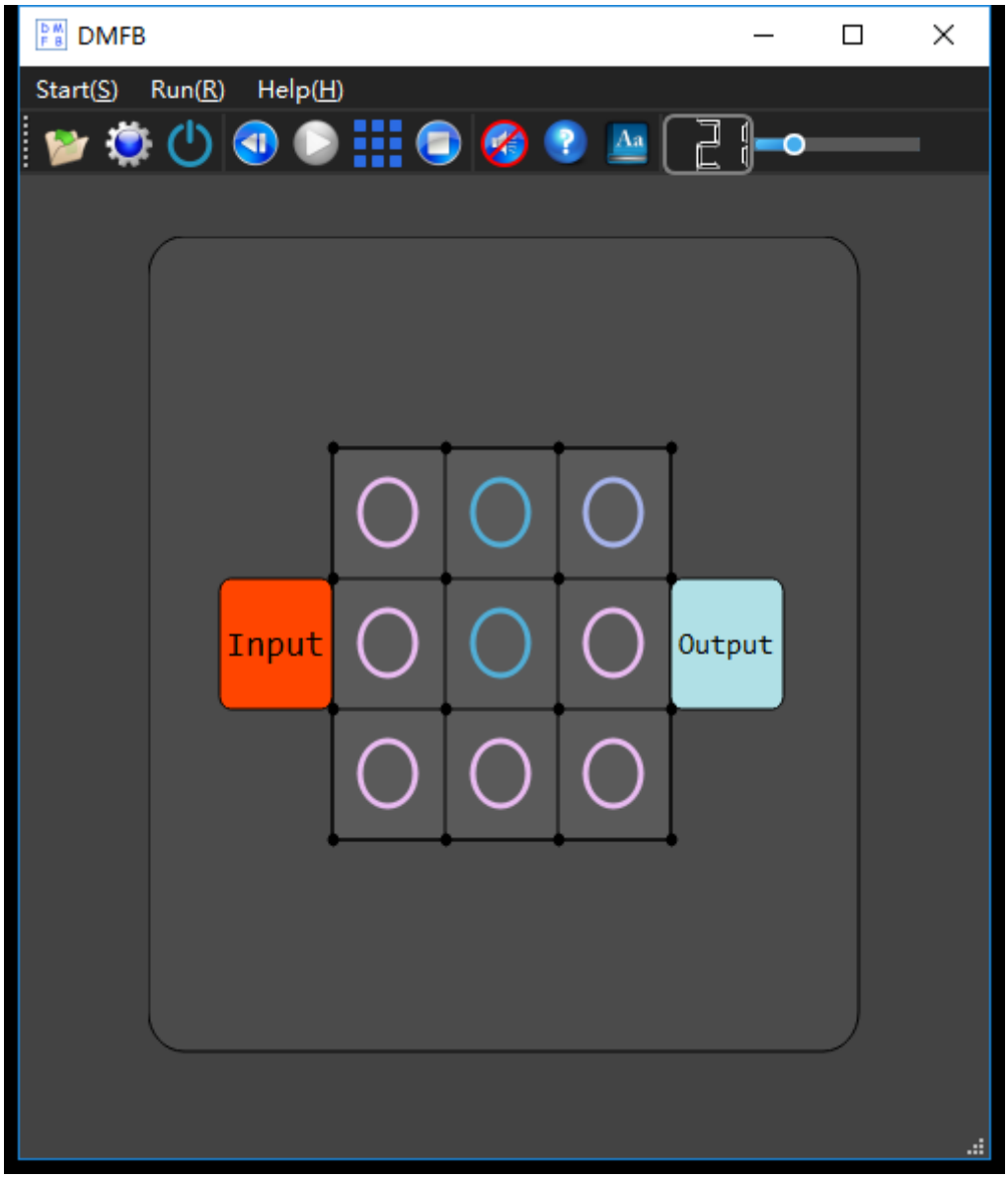


本着优化用户体验的原则，我最终将主界面设计成了如图的样子，其中菜单栏的按钮仅仅是为了迎合部分不习惯靠图标而习惯靠文字选择功能的用户，实际上仅靠工具栏的图标或者仅靠快捷键均可以完成所有功能操作。

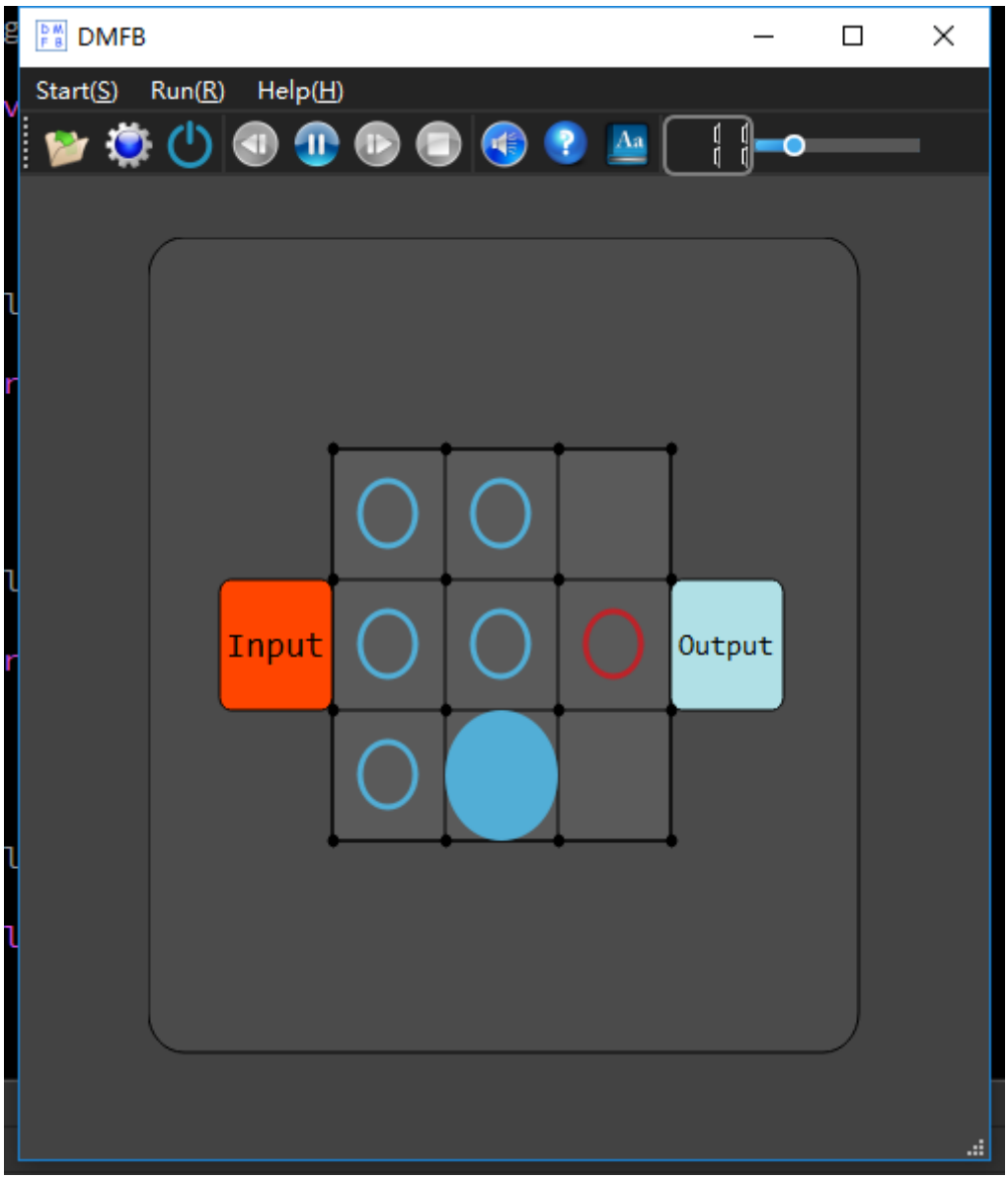
图中工具栏从左到右作用依次如下(括号内是快捷键)：

1. 打开文件(L)：点击后可以选择要加载的输入文件，快捷键。
2. 设置芯片(O)：可以设置电极行列、输入输出口、是否清洗。
3. 退出程序(ESC)：点击后可关闭程序。
4. 上一步(A | left)：返回上一时刻(播放或清洗状态下无法使用)。
5. 播放/暂停(P)：可以播放/暂停程序。
6. 下一步/污染计数(D | right)：点击跳到下一个时刻(播放状态下无法使用)或显示污染计数结果。
7. 停止(R)：将时间复位到0时刻。
8. 声音开关(V)：点击可开启/关闭声效。
9. 使用说明(I)：可以查看使用说明弹窗。
10. 程序相关：此处应有软件版权等信息，但是由于本人还没上软件工程，只作为一个占位弹窗。
11. 时间LCD：显示当前的时刻。
12. 速度条(W & S | up & down)：可以调整播放速度(包括清洗速度)，从1fps到20fps。

其中，播放按钮在点击后会变成暂停按钮，下一步按钮在播放完毕后会变成显示结果按钮，声音按钮在点击后出于关闭声音状态时图标也会改变，以下是这三个图标变化时的样图：



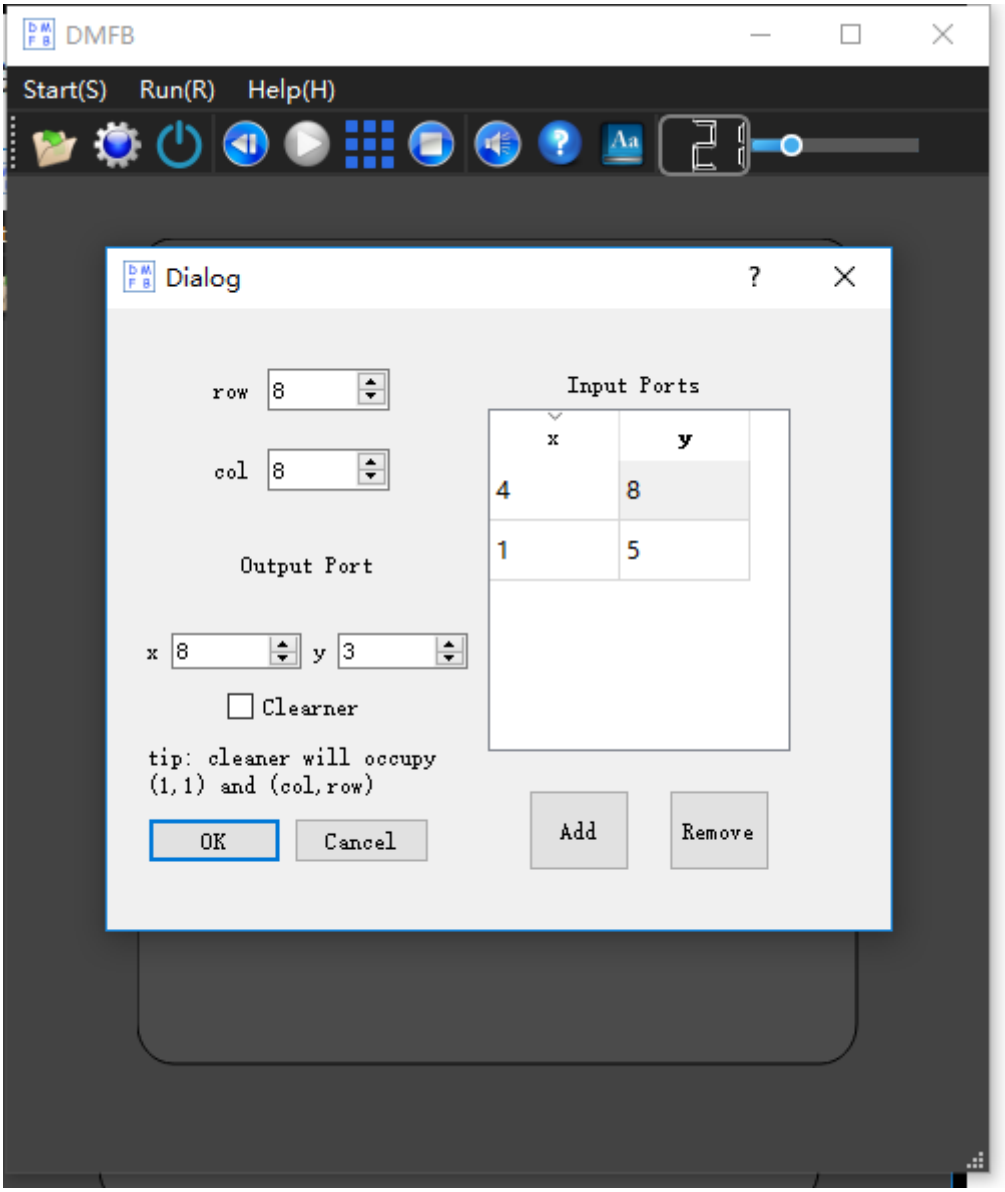
上图为最后一阵时下一步按钮变化报告污染状态按钮，并且已经将声音关闭。



上图为播放过程中上一步、下一步、复位均禁用，播放按钮变暂停按钮。

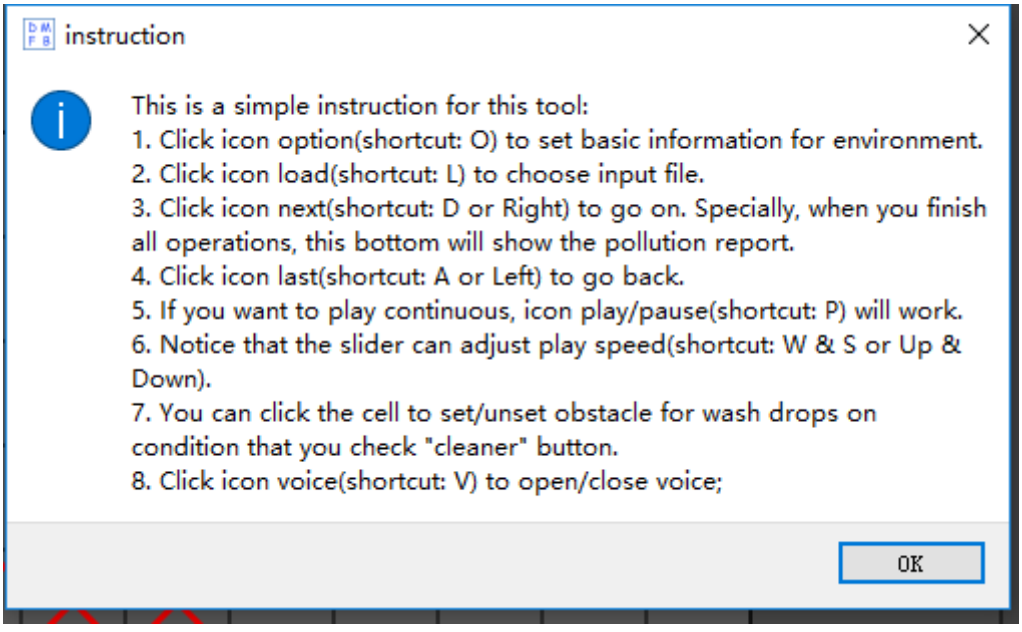
1.2 其他窗口

时间仓促，设置界面的制作较为简陋：



如图可设置行列、唯一的输出端口、是否使用清洗液滴以及一个用来添加输入端口的表格，表格的add、remove按钮分别可以添加、删除一个输入端口，设置完后点击OK或按回车键即可完成设置。

另外启动程序时会先跳出说明窗口，在主界面按I也可以弹出这个窗口：

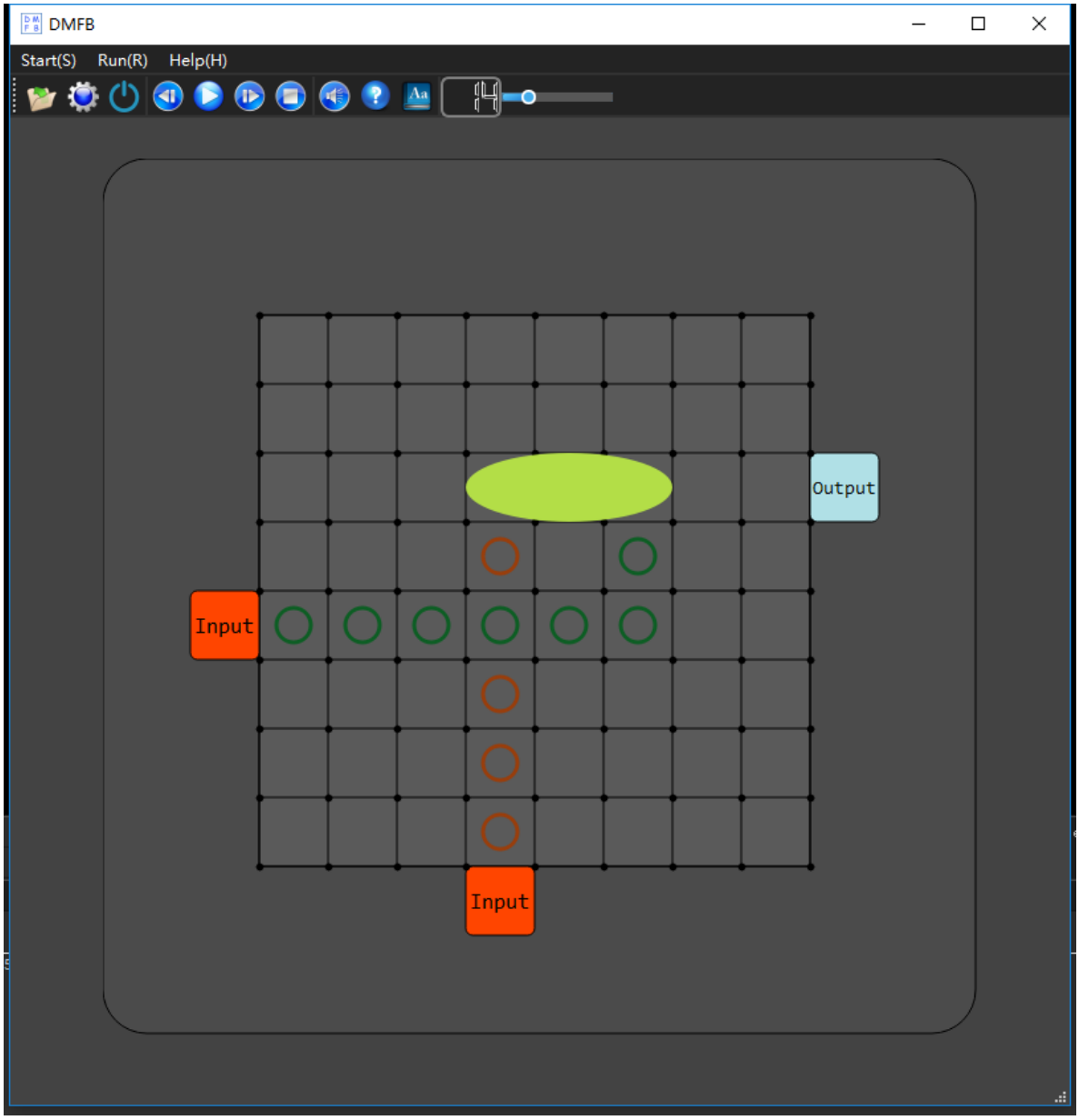


此外还有一个~~其实没什么用的~~about窗口：

1.3 运行效果

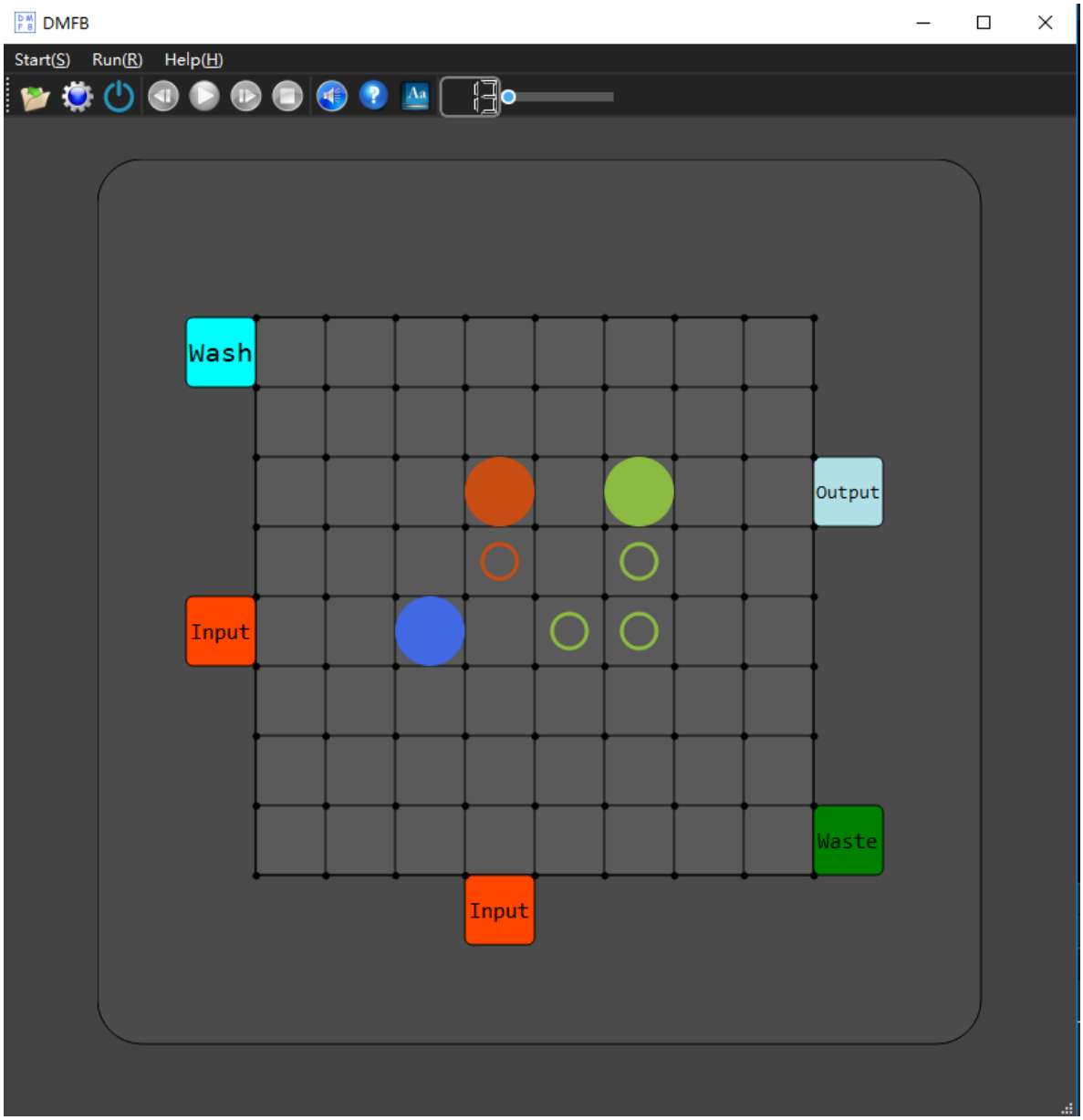
1.3.1 非清洗模式

运行时我用简单的椭圆绘制液滴，对于不同的液滴用不同颜色区分，清洗液滴颜色给定不变，液滴留下的污痕采用同颜色的椭圆环表示，并且在非清洗模式下会后覆盖前。



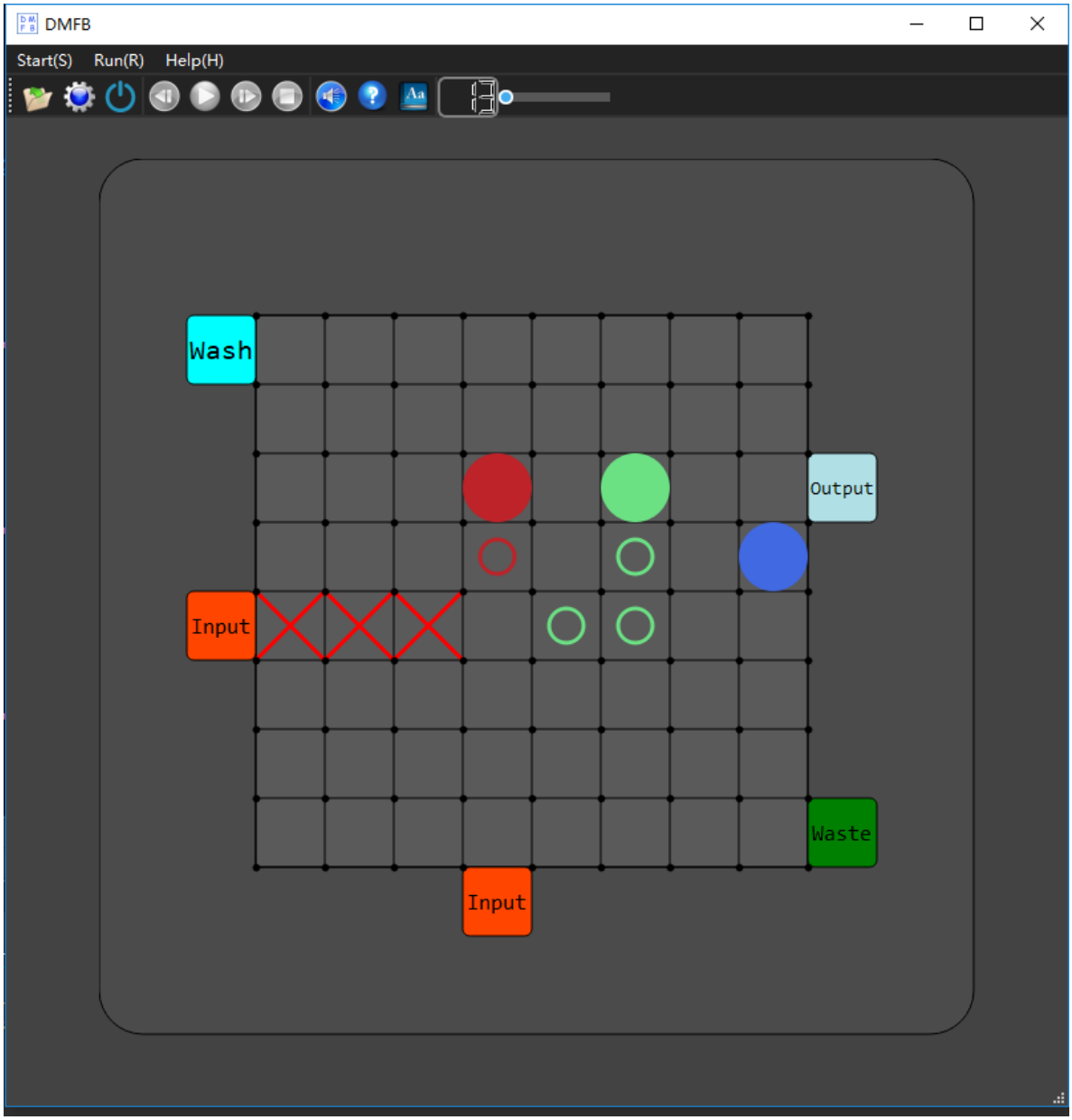
上图是不加清洗液滴的情况下，输入两个液滴正在经过一段移动后正在融合的场面。

1.3.2 清洗模式



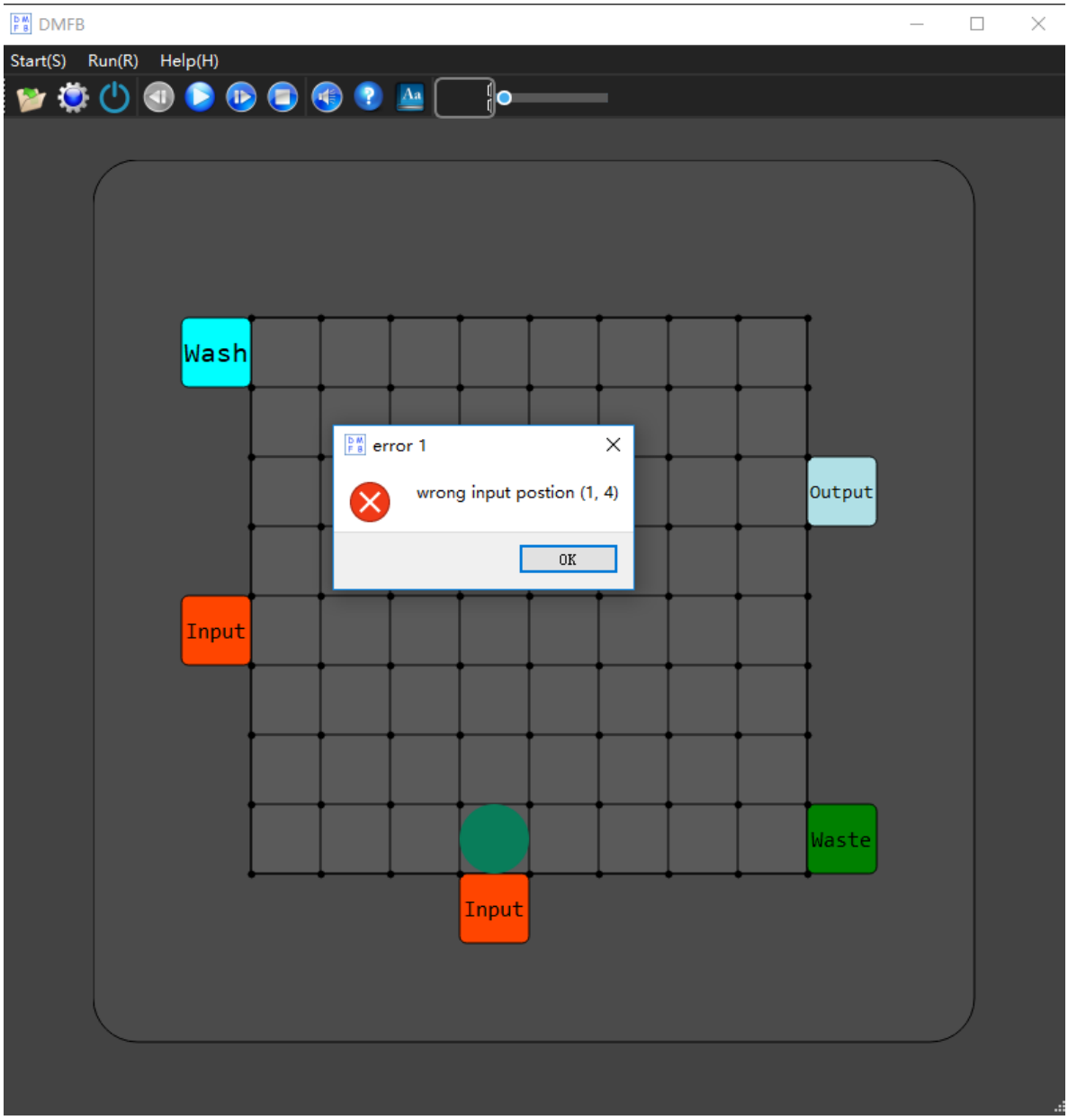
上图是清洗液滴(图中蓝色液滴)正在行为的场景，清洗液滴会在每过一帧后清洗掉所有能清洗的污迹，图中它刚从左上角到达中央区域，准备清洗掉右边两处绿色污迹。

此外清洗模式下暂停时可以点击电极设置/取消清洗障碍，阻止清洗液滴移动到这些位置，例如上图所示帧如果我在合适位置加上障碍，清洗液滴将走不同的路径：



如上图，由于左边路被堵住，以及静态约束的影响，清洗液滴只能绕道走右边前往清洗目标。需要说明的是如果清洗液滴无法在可到达区域找到清洗目标，那么这一帧结束时将不会触发清洗。

1.3.3 指令错误



对于各种错误情况均有弹窗报错，例如图中所示情况即是输入文件在下一个时刻需要在(1,4)位置输入一个液滴，但是(1,4)并非输入端口，因此报错，此外报错还包括清洗模式下发生污染、输出端口错误、各种指令错误——包括指令输入坐标不相邻，或者该移动的液滴实际不存在等，这些错误均不会导致程序崩溃，只会序运行到这一步弹窗报错并停止播放，这之后可以回放(非清洗模式)或复位，不能继续下一步。

1.4 声效

程序中共有八种音效，分别是输入、输出、移动、融合第一阶段、融合第二阶段、分裂第一阶段、分裂第二阶段、清洗。其中清洗时其他指令会停止，所以清洗音效单独存在，其他音效可以在同一帧里同时播放。

当快速播放时，后一帧的相同音效会覆盖前一帧音效，不会出现声音过于嘈杂的情况。

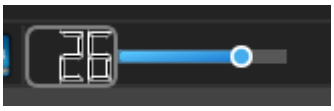
2. 软件亮点

2.1 变速播放

播放时帧率过慢可能并不满足某些情况的实际需求，尤其是当用户只想验证一个输入是否能顺利运行完时——而与此相反的，如果用户很希望看清楚每一个液滴的行走情况，那么慢速播放就很重要。因此我设计了一个速度调节器，可以从最慢1fps到最快20fps之间进行切换(这个功能只对播放有效，长按D或右键时播放速度由键盘响应速度决定)，另外液滴清洗进行了适应的加速，其帧率是正常播放的四倍。

2.2 qss样式

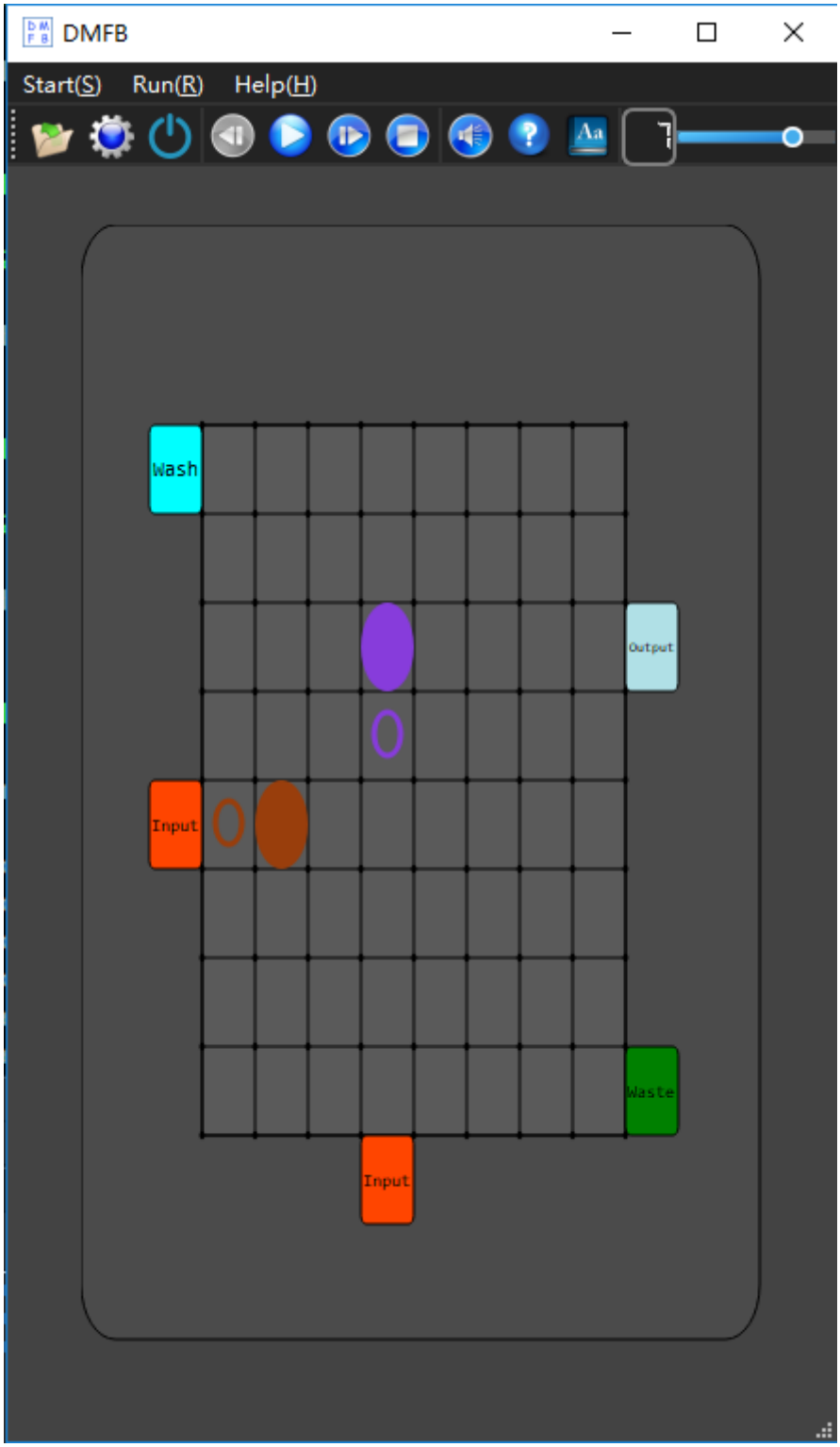
为了让界面更美观，我除了采用圆角替代方角外，还用了一些qss进行美化，包括界面的所有颜色设定，以及改变LCD和速度条的样式以符合界面风格。



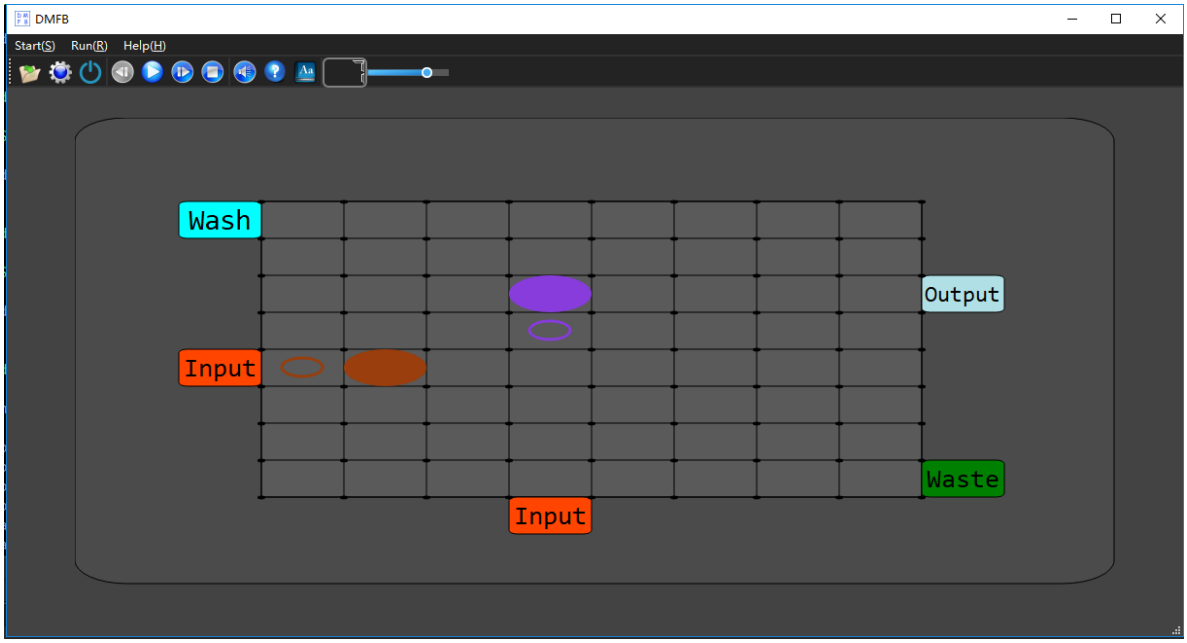
如图，缩放条的颜色是符合按钮整体风格的蓝色，并且加上了线性渐变色以达阴影效果。

2.3 缩放适应

界面的一大特点是可以自行适应窗口大小以刷新绘图状态，例如同样是8行8列的情况，对窗口进行拉伸后可以得到以下情形：



上图为拉高时的情形。



上图为拉宽时的情形。

可以看作，无论是布局还是液滴，甚至是端口上的文字，都可以根据窗口大小自适应地调整，这可以作为程序的亮点之一。

3. 程序设计

3.1 文件结构

Qt工程中主要包含以下文本文件：

- dmfb.h：定义了主要数据类DMFB，并声明了其中的接口函数。主要包括初始化、读取指令、下一帧、上一帧等功能。
- dmfb.cpp：实现了DMFB类的各个函数，在函数内部进行了一定程度的错误信息管理以保证程序正常运行。
- drawwidget.h：定义了继承自QWidget的派生类DrawWidget，用于绘制芯片界面。
- drawwidget.cpp：实现了DrawWidget类的各个函数，留出了更新大小、更新背景、更新前景等接口，并重载了鼠标点击事件。
- dropitem.h：定义了所有液滴的类，包括抽象类DropItem以及它的派生类Drop、DropMark、LargeDrop、CleanerDrop。
- dropitem.cpp：实现类上述液滴类的内容，留出了接口供DMFB与DrawWidget使用。
- functions.h：定义了一些通用接口(实际上只定义了一个)。
- functions.cpp：实现了functions.h中接口的内容。
- instruction.h：定义了用于存储指令数据的指令类，包括基类Instruction与派生类InputData，OutputData，MergeData，SpliteData，MoveData，MixData。
- instruction.cpp：实现了指令类的内容，主要是构造函数的编写。
- mainwindow.h：定义了继承自QMainWindow的主窗口类MainWindow，主要用于显示界面与简单的按钮禁用逻辑，也包括后面加的声音管理等。
- mainwindow.cpp：实现了MainWindow类的函数，里面会new一个DMFB实例作为数据来源，UI中包括一个DrawWidget实例用于展现芯片界面。
- option.h：定义了继承自QDialog的设置窗口类Option，供主窗口调用。
- option.cpp：实现了QDialog类的内容。
- mainwindow.ui：定义了主窗口UI的内容，包括工具栏、菜单栏的内容，进度条，LCD显示，绘图区域以及布局。

- option.ui：定义了设置窗口UI的内容，包括清洗检查框、行列数字设置、QTableView充当的输入端口的等。
- png.qrc：所有图片资源。
- sound.qrc：所有音效资源。
- txt.qrc：指引窗口的文本资源。

3.2 设计思路

3.2.1 整体设计

在我看来，本次基础要求中除去第七点引入清洗液滴之外，**前六点要求本质上是要做一个播放器**，因此我的设计理念比较接近我心中播放器的模式（包括按钮其实采用的播放器的按钮）：

主要的绘图窗口仅仅展示内容，根据获取到的数据进行绘制，实际数据存储在数据类(DMFB)中，数据类中将完整过程的数据都记录下来，如同一个视频文件一般——如果数据太大可能需要压缩，但实际上估算可以知道，最多100帧，每帧最多不到200液滴与污迹，乘起来只有20K液滴信息，而一个液滴的信息只需要几十字节，所以存下所有帧的信息所需要的空间不超过2MB。所以一个可行的办法是在输入指令后先在后台计算出整个运行过程每一帧的状态，前端切换、播放的操作只是对数据类的时刻变量进行操控。

实际上我担心在一开始计算花的时间会过长，并且考虑到加入人机交互后不能一次得到所有结果，因此在进入下一帧时我选择现场计算，但是回退前面的帧时只是对时刻变量进行改变，读取内存，而回退后继续往前时也采用了fast forward的策略，改变所处时刻。

加入清洗液滴后整个程序从本质上发生了变化：从原本的播放器进化成了一个小游戏，需要人机交互，每回合用户添加的障碍都会对后面的情形造成影响，并且清洗液滴的加入本身还是一种AI设计，这使得程序难度大涨。

好在助教提出清洗模式下不必返回上一步，因此之前播放器模式的结构可以继续沿用，只需要插入清洗算法相关的内容，并且在这种模式下需要处理污染报错，其他没有太多需要添加的内容。

3.2.2 清洗算法

我自知不懂算法，果断放弃了加分项要求，使用暴力贪心的方法每帧刷新后停止其他液滴，让清洗液滴单独行动，基于BFS和局部最优的方法每次将距离清洗液滴最近的污迹清掉，全部清理掉后从出口离开。如果不能完成从入口到出口的道路或者没有能清理的污迹，清洗液滴将不出动。

3.2.3 内存管理

对于播放器模式，内存管理非常重要，否则很难保证我所期望的低内存消耗同时还内存安全，除此之外我还希望回退时液滴颜色和原来播放时是相同的。为了实现这些效果，我对液滴进行了特殊的内存管理，由DropItem提供静态接口统一管理，每次有新液滴登场(new)时需要在这里统一注册、获得ID，存到一个key为ID，value为液滴指针的map中，然后这些液滴将被保留，直到DMFB调用接口重置液滴数据时。

由于我的液滴颜色是作为液滴成员变量的，在这个模式下，回退时会记录每个位置的液滴指针，通过指针找到对应液滴，知道它的类型、颜色等信息，以供前端进行显示。从实际运行来看，加入清洗液滴之前，我的程序运行的内存消耗只有6MB左右，最终把清洗液滴相关的内容加进去后也只有25MB，对一个窗口程序来说应该不算大。

3.2.4 指令化简

一个需要跨多帧的指令在我的架构下处理起来是有点麻烦的，为了处理分裂和融合，我专门设定了`LargeDrop`来记录中间态，并且每帧会记录需要分裂与融合的液滴指针，它们只需要两帧即可完成也不算特别麻烦。而混合指令就比较麻烦了，延时可以非常大。

而恰好助教表示混合的声效用移动的声效，我不知道这算不算一个暗示，我从中受到启发，将所有的混合指令读入后拆解成了若干个移动指令，大大简化了指令实现难度。后来我想到其实分裂融合也可以这般简化指令，不过想到这件事情的时候分裂融合部分已经写好了，也就没有这么干。

3.2.5 绘图策略

绘图我希望尽量低消耗完成，为此我绘制了一张`QPixmap`，再在`DrawWidget`上显示出来，这样一来`paintEvent`里仅仅是将这张图片进行显示，图片的绘制只在后端数据更新时进行，可以大大降低消耗，因此我也敢于将播放速度调得比较快。

绘图的时候我还采用了前后景分离的模式，背景是在设置完后已经生成的(特别的，用户设置障碍时会重新绘制背景，但这个过程也不需要连续播放，不会影响效率)，而前景则是所有的液滴，这样一分工绘图效率大大增加。

3.3 采坑记录

实际上写程序的过程不可能一帆风顺，不可避免地经历了不少坑，但在扫坑的过程中也增长了不少知识，包括Qt的使用以及我之前并没有熟练的OOP内容。

1. 抽象类的使用：之前我并不清楚抽象类为何要存在，以及哪些函数不能被定义为纯虚函数。这次写液滴类时我突然明白了，某些函数确实需要所有派生类都用到，我希望能用基类指针直接调用，但是对于这个基类又没办法定义清楚实现，就可以用纯虚函数。而构造函数和析构函数不能写成纯虚函数，因为我构造液滴时一定要先构造抽象液滴，析构时亦是如此，必须要被实现。在经历了一通与Qt无关的编译报错后，终于认清了这些问题.....
2. update与repaint：起初课件上说update的合并问题时我并没有在意，因为我实际操作感觉用update并没有问题。直到加入`QSound`的处理后，发现界面更新异常时我才想起repaint的事，赶紧将update改成repaint，颇有一种“不听老人言，吃亏在眼前”的感觉。
3. 错误处理：起初我只按照作业要求处理了部分数据错误，但是在调试过程中由于我的程序第一次写好必然有debug，非常需要知道一些数据处理错误，而这些错误在我没有预判的情况下会直接由内存错误引起程序崩溃，导致调试非常低效。后来我将各种可能出现的指令错误全部处理后，总算是能及时知道异常的存在与表现，顺利debug。
4. 字号处理：在调整文字的自适应窗口大小时，我已经用了`QFontMetrics::size`接口算好了文字框，计算出了需要放大的倍率，并用`QFont::setPointSizeF`接口调整大小，最后却发现得到的文字非常之小，经过多次验证后才意识到我设定新的字号时，需要用倍率乘以原来的字号，由于我之前是直接将倍率当参数传过去的，自然就很小。
5. 指令读取：读取一行指令后我会先按空格split一次，前一项作为指令类型，后一项再对逗号split得到各个数字，但是刚开始调试时发现后面总是会读到一个0，不知道为什么，最后单步调试时才发现是因为最后一个数字后面带上了分号，Qt的字符串转整数接口对于非法字符串会自动输出0，这着实是我处理的失误，把分号remove掉后就OK了。（不过我也想吐槽下为什么这个指令一定要加个分号，风格已经这么像汇编了，学着汇编不加分号不好吗？）