实验一 MiniDraw

ID: 58 陈文博

February 22, 2020

1 实验要求

* 写一个画图小程序 MiniDraw，要求画直线 (Line)，椭圆 (Ellipse)，矩形 (Rectangle)，多边形 (Polygon) 等图形元素（图元）
* 每种图元需用一个类（对象）来封装，如 ‘CLine‘，‘CEllipse‘，‘CRect‘，‘CPolygon‘，‘CFreehand‘
* 各种图元从一个父类来继承，如 ‘CFigure‘
* 学习类的继承和多态

2 功能描述

在实验基本要求基础上，我还增加了一下功能：

- 设置画笔颜色 (color)、宽度 (width)
- 闭合图形颜色填充 (fill)
- 绘制平滑曲线
- 撤销 (Undo) 绘图
- 保存 (Save) 画板
- 给工具栏、菜单栏和主窗口设置了图标

基本效果如下：

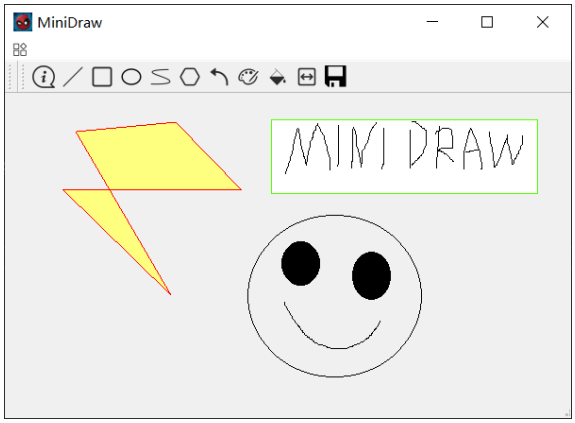

Figure 2.1: 实验效果

3 开发环境

IDE： Microsoft Visual Studio 2019 community

CMake： 3.16.3

Qt： 5.14.1

4 架构设计

4.1 文件结构

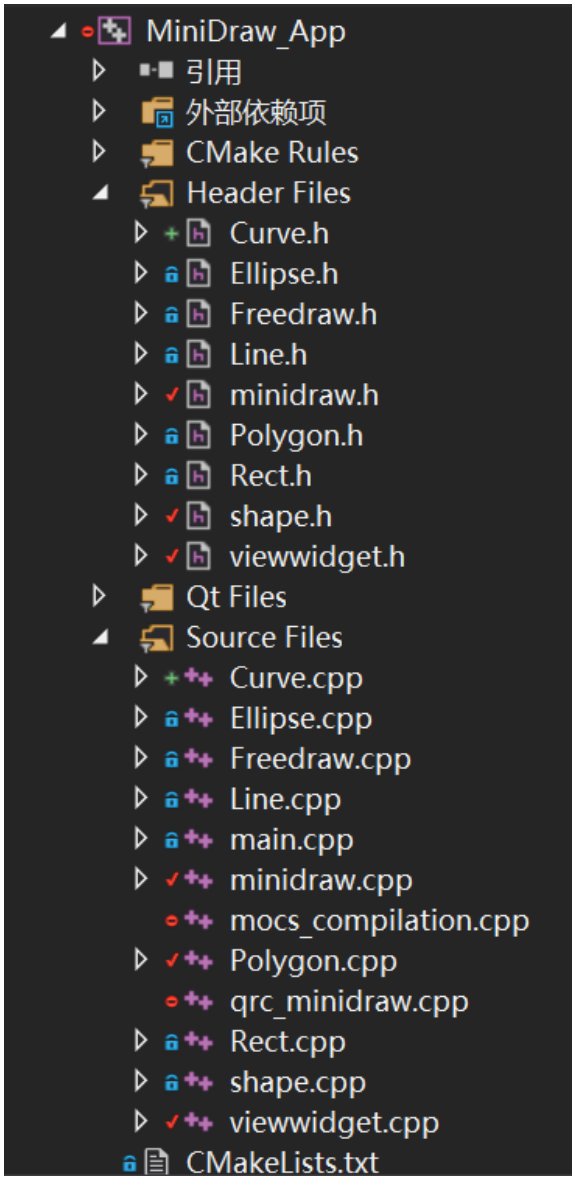

Figure 4.1: 文件结构

4.2 各个类的继承关系

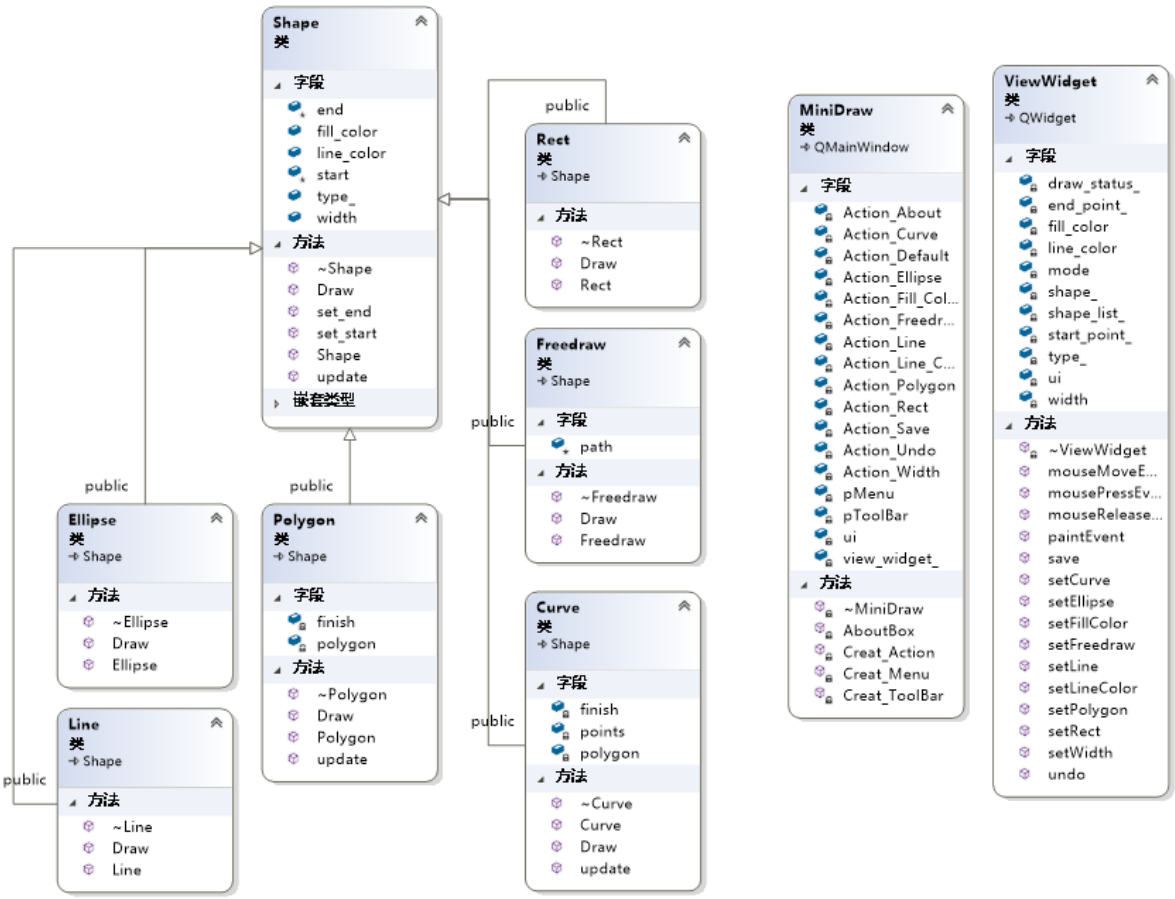

Figure 4.2: 各个类的继承关系

5 设计难点与解决

5.1 Freedraw 的实现

使用 Qt 自带的类 QPainterPath 存储 Freedraw 产生的轨迹点，在绘图刷新阶段使用 painter 的 drawPath 方法进行绘制。这种方法比循环调用 drawLine 方法效率要高得多。

5.2 Polygon 的实现

和 Freedraw 类似，Polygon 也需要一个存储多边形各个顶点的结构，Qt 同样提供了 QPolygon 类实现该功能。由于 Polygon 涉及到两种鼠标操作：左击开始选取顶点（存储点）；右击连接闭合图形。为此引入类方法 update，在 shape 父类中添加虚函数 update，在 Polygon 子类中定义，目的是通过鼠标左击右击修改控制变量 mode 来切换绘点模式和连接闭合模式，实现功能要求。

5.3 Undo 的实现

画板为实现显示之前所有绘制的图像，维护了一个 vector 类来存储所有产生的 shape 子类，要实现 undo，只需要将 vector 中最后一个添加的元素删除即可，这里要注意在 pop_back 之前应该先 delete 最后一个元素，否则会发生内存泄漏。

```
void ViewWidget::undo()
{
    if (shape_list_.size() > 0)
    {
        delete shape_list_.back(); // 不加delete将造成内存泄漏
        shape_list_.pop_back();
    }
}
```

Figure 5.1: 程序截图

5.4 设置线宽、线色和颜色填充

绘图的线宽、线色和填充色可以通过对 painter 进行设置，为了能够保存并显示所有形状的颜色，在父类 shape 中添加相应的属性 line_color、fill_color 和 width，设置相关 Action 并添加到工具栏中。用对话框 qcolordialog 和 qinputdialog 进行颜色选择和线宽输入，在遍历 shape_list_ 时读取对象的属性修改 painter，达到相应的效果。

5.5 保存图片

利用 QPixmap 类获取图像和 QFileDialog 进行路径选择可以很容易的实现画板图像保存。

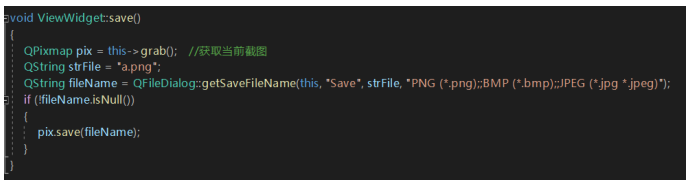

Figure 5.2: 程序截图

5.6 绘制曲线

在绘制多边形的基础上利用 QVector 类存储各个顶点，通过鼠标左键选择各个采样点，最后通过右键点击通过 QPainter 的 CubicTo 方法插值生成平滑曲线

5.7 图标设置

由于添加功能后工具栏文字很长，甚至一部分被自动隐藏，很不美观，故想到用图标代替各文字按钮。方法很简单，从网上下载相关图标

Figure 5.3: 下载好的图标

对各个 Action 的定义中添加 Icon：

```
Action_About = new QAction(tr("&About"), this);
Action_About->setIcon(QIcon(QString("../src/img/about.png"))); // 添加图标
connect(Action_About, &QAction::triggered, this, &MiniDraw::AboutBox);

Action_Line = new QAction(tr("&Line"), this);
Action_Line->setIcon(QIcon(QString("../src/img/line.png")));
connect(Action_Line, SIGNAL(triggered()), view_widget_, SLOT(setLine()));
```

Figure 5.4: 程序截图

生成运行得到的效果：

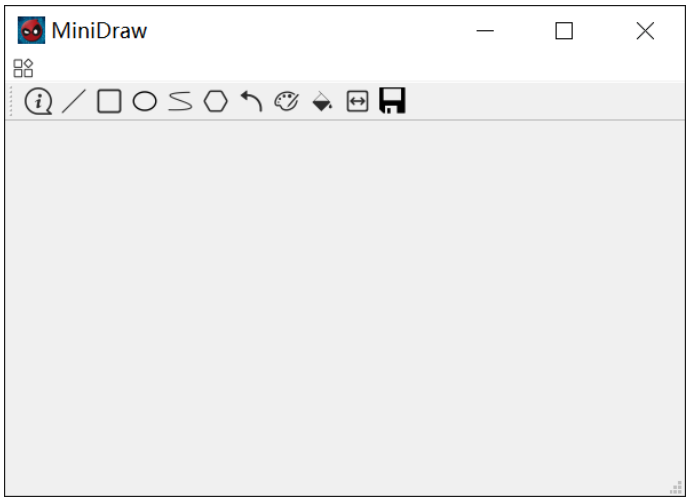

Figure 5.5: 实现效果

5.8 关于两个工具栏问题

如下图所示，按照原框架生成的 GUI 会出现两个工具栏问题，这是由于在 minidraw.ui 中已经定义了一个默认的工具栏"mainToolBar"，程序中使用 addToolBar 函数将会新建另外一个工具栏。

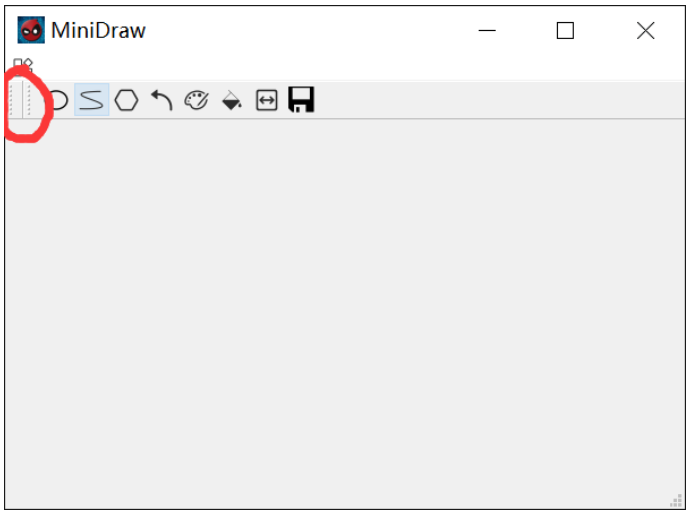

Figure 5.6: 两个工具栏现象

解决该问题的方法是在 minidraw.ui 中删去 ToolBar 一项，在 minidraw.h 中包含头文件 qtoolbar.h，即可解决问题，效果如下：

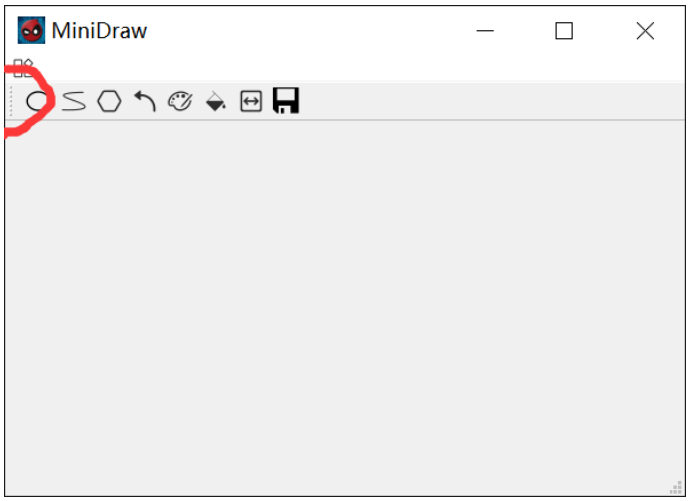

Figure 5.7: 修正效果

6 实验效果

6.1 绘制直线

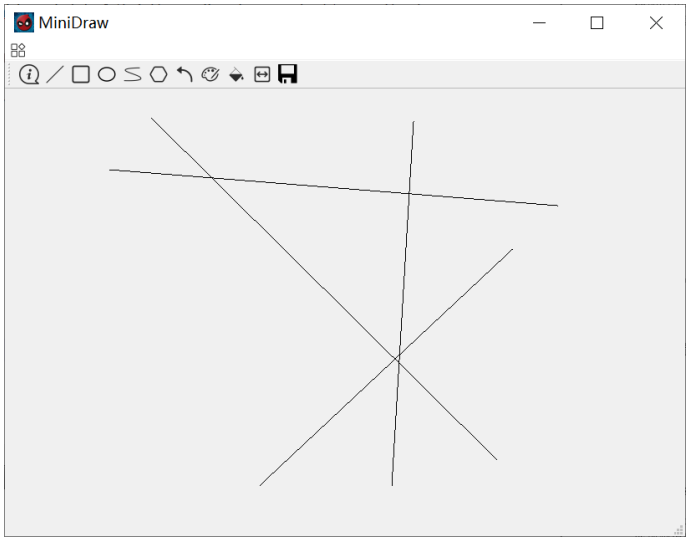

Figure 6.1: 绘制直线

6.2 绘制矩形

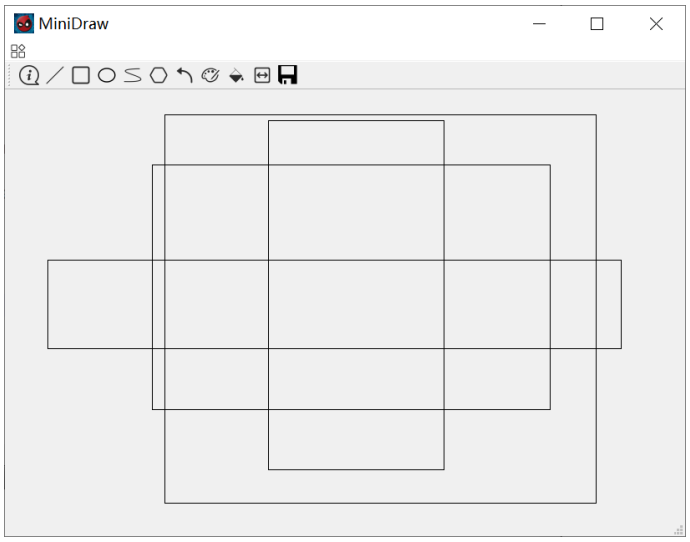

Figure 6.2: 绘制矩形

6.3 绘制椭圆

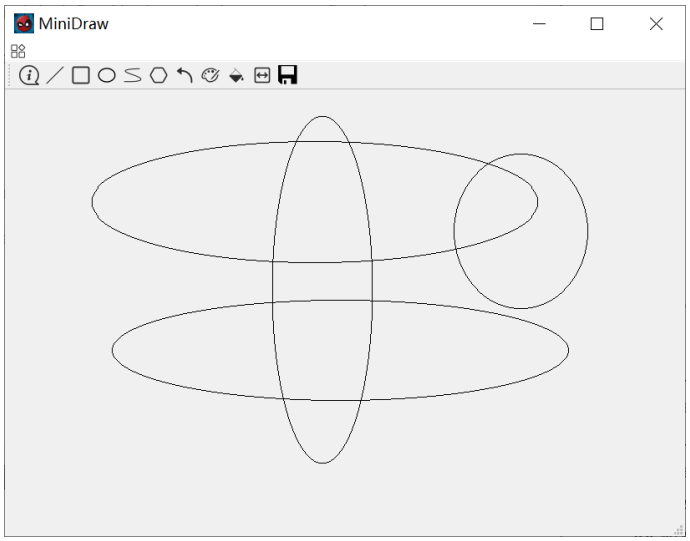

Figure 6.3: 绘制直线

6.4 自由绘图

Figure 6.4: 自由绘图

6.5 绘制多边形

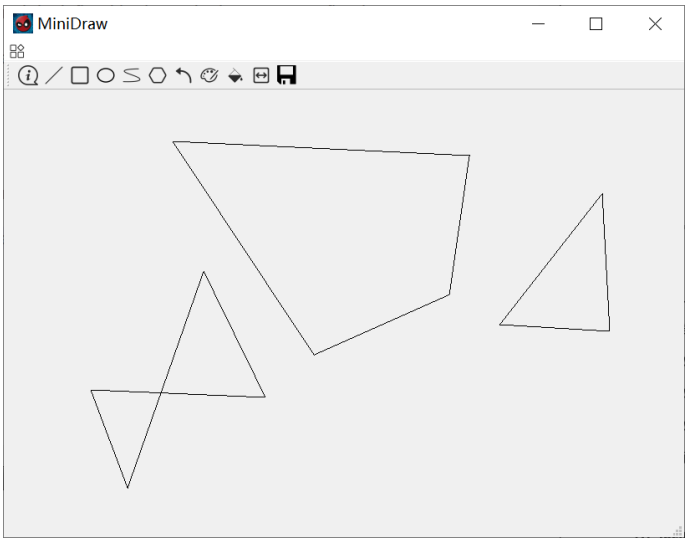

Figure 6.5: 绘制多边形

6.6 撤销操作

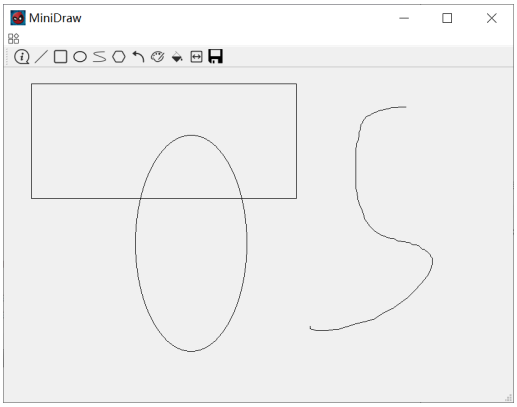

Figure 6.6: 撤销前

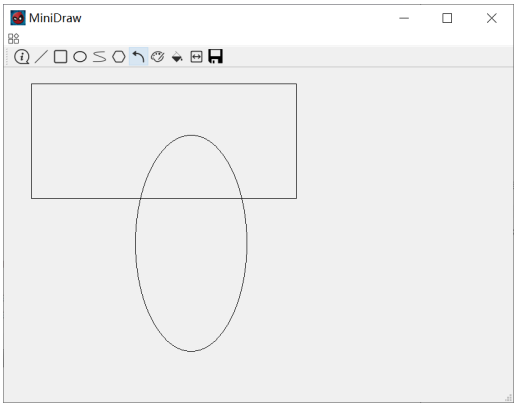

Figure 6.7: 撤销后

6.7 修改线色

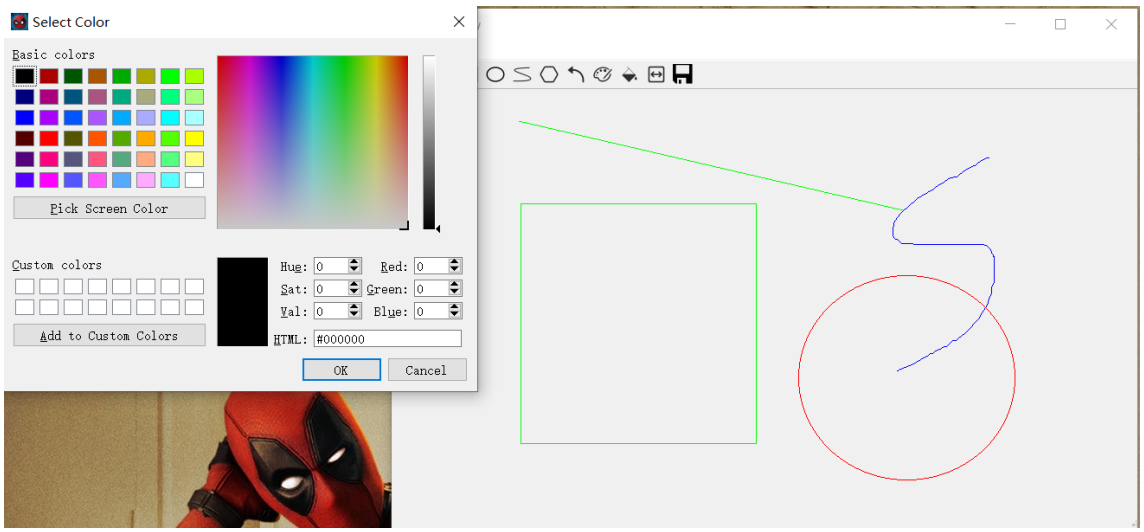

Figure 6.8: 修改线色

6.8 修改填充色

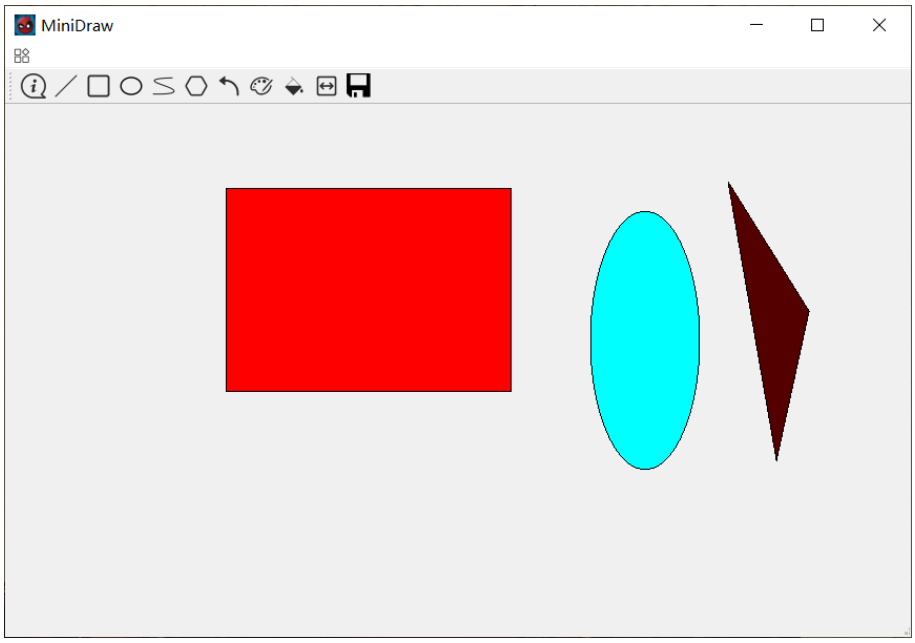

Figure 6.9: 修改填充色

6.9 修改线宽

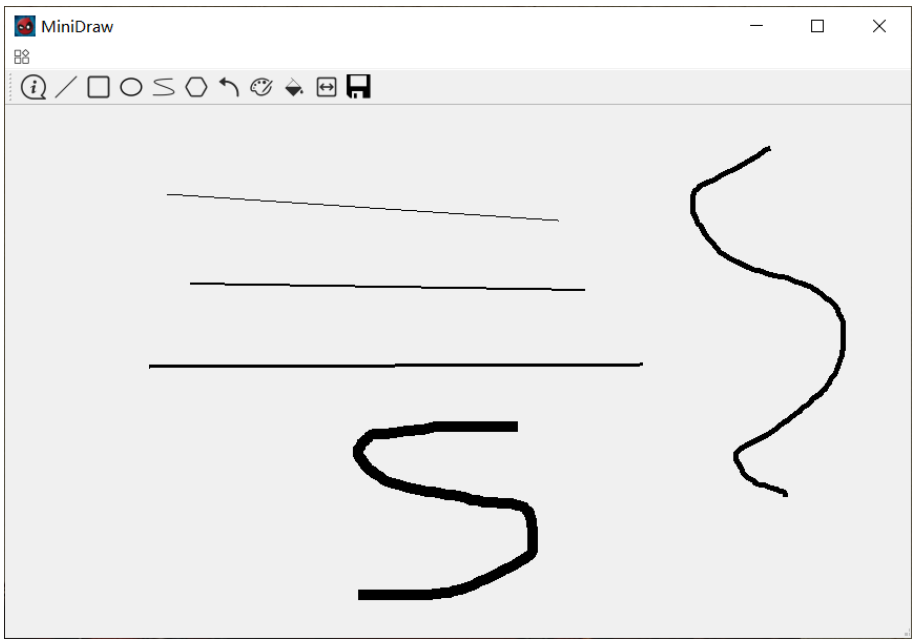

Figure 6.10: 修改线宽

6.10 保存图片

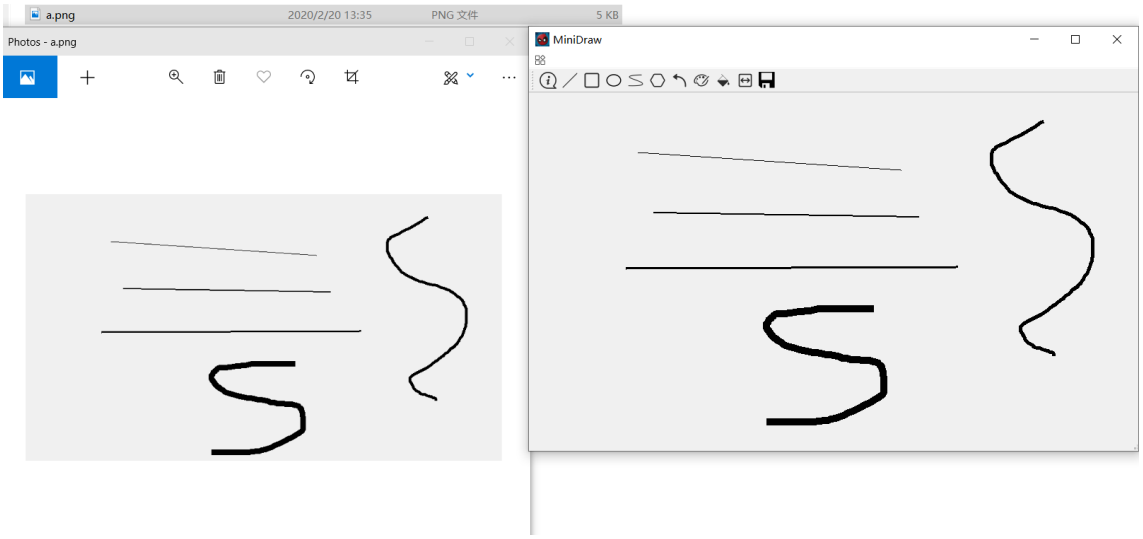

Figure 6.11: 保存图片

6.11 绘制曲线

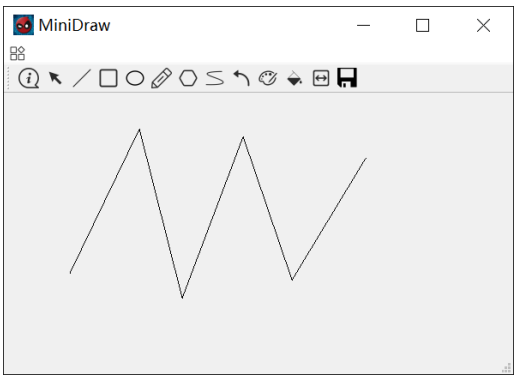

Figure 6.12: 选点

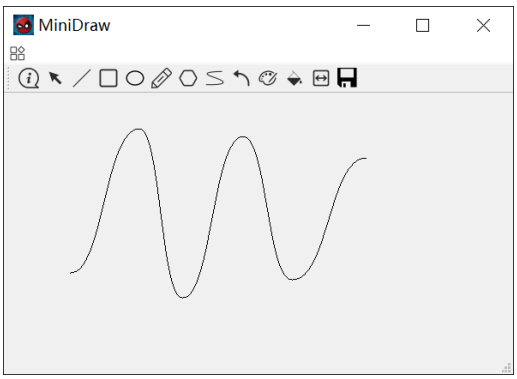

Figure 6.13: 画线

6.12 总结

这次实验总体上难度不算大，因为一些电脑原先环境的原因，前期比较多的时间花在环境配置上，虽然之前没接触过 Qt，对 C++ 也不是很熟，但在理清框架的连接关系之后还是不禁赞叹面向对象的强大，原理理清之后各种功能无非就是拼凑衔接而已，期待后续实验。