南京工程学院

实 训 报 告

课 程 名 称 多媒体编程基础

实训项目名称 实训 3： 简单绘图板

实训学生班级 数嵌 172

实训学生姓名 朱广锋

学 号 202170638

一、 实训目的

1.掌握 Qt Creator 的基本使用方法

2.理解界面设计的相关类

3.设计界面并编写一个简单的文本编辑程序，功能尽量完整

二、 实训环境及开发工具：

PC 机、Qt5.14（或其它版本）

三、 实训要求及内容：

1. 重点学习理解教材案例 CH602 和 CH603，以及提供补充材料和源代码。

2. 设计一个简单绘图软件，功能至少包括：**手绘矩形（或圆形）、手绘涂鸦，可以根据选定的颜色绘制**。自己选择界面风格。

3. 可以参考教材或提供的源代码，自己选择开发方法。

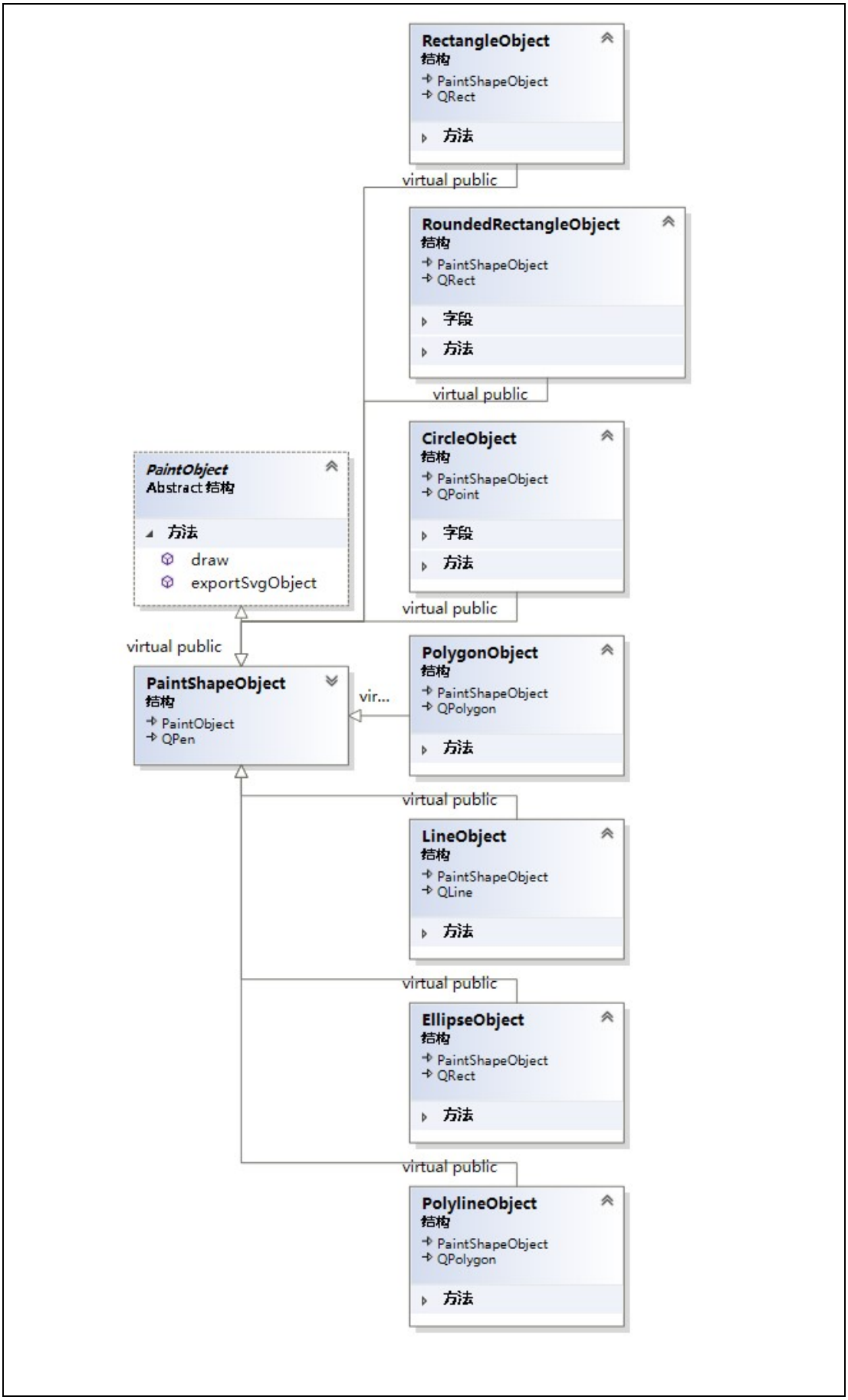
四、 程序设计思路（30 分）

主要利用了 C++的多态抽象出绘制和导出 svg 的方法，利用回调实现跨窗口函数调用，大多 Action 在构造窗口时使用 lamdba 作为槽函数。

界面上，在 ToolBar 上主要是切换绘画模式，侧边栏进行颜色和画笔粗细的调整，同时显示了当前画板保存的对象类型，可以看作图层。

图像可以导出为 svg 矢量图。

图像在 PaintWidget 绘制，在 PaintX 管理。


RectangleObject
结构
PaintShapeObject
QRect
方法
virtual public
RoundedRectangleObject
结构
PaintShapeObject
QRect
字段
方法
virtual public
CircleObject
结构
PaintShapeObject
QPoint
字段
方法
virtual public
PaintObject
Abstract 结构
方法
draw
exportSvgObject
virtual public
PaintShapeObject
结构
PaintObject
QPen
vir...
PolygonObject
结构
PaintShapeObject
QPolygon
方法
virtual public
LineObject
结构
PaintShapeObject
QLine
方法
virtual public
EllipseObject
结构
PaintShapeObject
QRect
方法
virtual public
PolylineObject
结构
PaintShapeObject
QPolygon
方法

五、设计方法及代码（30 分）

绘制对象类定义

```
#pragma once
#include <QPainter>

namespace thatboy
{
    namespace  qt
    {
        // 保留类的默认构造与赋值函数
#define DEFAULT_CONSTRUCT(_class) \
        _class() = default; \
        _class(const _class&) = default; \
        _class(_class&&) = default; \
        _class& operator=(const _class&) = default;

        /// <summary>
        /// 绘制对象类
        /// </summary>
        struct PaintObject
        {
            virtual void draw(QPainter&) = 0;
            virtual QString exportSvgObject() = 0;
        };

        /// <summary>
        /// 形状类
        /// </summary>
        struct PaintShapeObject
            : virtual public PaintObject
            , virtual public QPen
        {
        };

        /// <summary>
        /// 线条
        /// </summary>
        struct LineObject
            : virtual public PaintShapeObject
            , virtual public QLine

        {
            DEFAULT_CONSTRUCT(LineObject)
```

```
                using QLine::QLine;
            LineObject(const QLine& l) :QLine(l) {}

            virtual void draw(QPainter& painter)
            {
                painter.setPen(*this);
                painter.drawLine(*this);
            }

            virtual QString exportSvgObject()
            {
                return QString().sprintf(R"(<line x1="%d" y1="%d" x2="%d" y2="%d"
stroke="#%02X%02X%02X" stroke-width="%d" fill="none"/>)"
                    , p1().x(), p1().y(), p2().x(), p2().y(), QPen::color().red(),
QPen::color().green(), QPen::color().blue(), QPen::width());
            }
        };

        /// <summary>
        /// 矩形
        /// </summary>
        struct RectangleObject
            : virtual public PaintShapeObject
            , virtual public QRect
        {
            DEFAULT_CONSTRUCT(RectangleObject)
                using QRect::QRect;
            RectangleObject(const QRect& r) :QRect(r) {}

            virtual void draw(QPainter& painter)
            {
                painter.setPen(*this);
                painter.drawRect(*this);
            }

            virtual QString exportSvgObject()
            {
                return QString().sprintf(R"(<rect x="%d" y="%d" width="%d"
height="%d" stroke="#%02X%02X%02X" stroke-width="%d" fill="none"/>)"
                    , normalized().x(), normalized().y(), normalized().width(),
normalized().height(), QPen::color().red(), QPen::color().green(),
QPen::color().blue(), QPen::width());
            }
        };
```

```
        /// <summary>
        /// 圆形
        /// </summary>
        struct CircleObject
            : virtual public PaintShapeObject
            , virtual public QPoint
        {
            DEFAULT_CONSTRUCT(CircleObject)
                int r{ 0 };
            void setR(int _r) { r = _r; }

            using QPoint::QPoint;
            CircleObject(const QPoint& p) :QPoint(p) {}

            virtual void draw(QPainter& painter)
            {
                painter.setPen(*this);
                painter.drawEllipse(*this, r, r);
            }

            virtual QString exportSvgObject()
            {
                return QString().sprintf(R"(<circle cx="%d" cy="%d" r="%d"
stroke="#%02X%02X%02X" stroke-width="%d" fill="none"/>)"
                    , x(), y(), r, QPen::color().red(), QPen::color().green(),
QPen::color().blue(), QPen::width());
            }
        };

        /// <summary>
        /// 椭圆
        /// </summary>
        struct EllipseObject
            : virtual public PaintShapeObject
            , virtual public QRect
        {
            DEFAULT_CONSTRUCT(EllipseObject)
                using QRect::QRect;
            EllipseObject(const QRect& r) :QRect(r) {}

            virtual void draw(QPainter& painter)
            {
                painter.setPen(*this);
```

```
                painter.drawEllipse(*this);
            }

            virtual QString exportSvgObject()
            {
                return QString().sprintf(R"(<ellipse cx="%d" cy="%d" rx="%d"
ry="%d" stroke="#%02X%02X%02X" stroke-width="%d" fill="none"/>)"
                    , normalized().center().x(), normalized().center().y(),
normalized().width() / 2, normalized().height() / 2, QPen::color().red(),
QPen::color().green(), QPen::color().blue(), QPen::width());
            }
        };

        /// <summary>
        /// 圆角矩形
        /// </summary>
        struct RoundedRectangleObject
            : virtual public PaintShapeObject
            , virtual public QRect
        {
            DEFAULT_CONSTRUCT(RoundedRectangleObject)
                int xR{ 15 };
            int yR{ 15 };
            void setR(int _xr, int _yr) { xR = _xr; yR = _yr; }

            using QRect::QRect;
            RoundedRectangleObject(const QRect& r) :QRect(r) {}

            virtual void draw(QPainter& painter)
            {
                painter.setPen(*this);
                painter.drawRoundRect(*this, xR, yR);
            }

            virtual QString exportSvgObject()
            {
                return QString().sprintf(R"(<rect x="%d" y="%d" width="%d"
height="%d" rx="%d" ry="%d" stroke="#%02X%02X%02X" stroke-width="%d" fill="none"/>)"
                    , normalized().x(), normalized().y(), normalized().width(),
normalized().height(), xR, yR, QPen::color().red(), QPen::color().green(),
QPen::color().blue(), QPen::width());
            }
        };
```

```
        /// <summary>
        /// 多边形
        /// </summary>
        struct PolygonObject
            : virtual public  PaintShapeObject
            , virtual public  QPolygon
        {
            DEFAULT_CONSTRUCT(PolygonObject)
                using QPolygon::QPolygon;
            PolygonObject(const QPolygon& p) :QPolygon(p) {}

            virtual void draw(QPainter& painter)
            {
                painter.setPen(*this);
                painter.drawPolygon(*this);
            }

            virtual QString exportSvgObject()
            {
                QString svg;
                svg += R"(<polygon points=")";
                for (auto& pt : *this)
                    svg += QString().sprintf("%d,%d ", pt.x(), pt.y());
                svg += QString().sprintf(R"(" stroke="#%02X%02X%02X"
stroke-width="%d" fill="none"/>)"
                    , QPen::color().red(), QPen::color().green(),
QPen::color().blue(), QPen::width());
                return svg;
            }
        };

        /// <summary>
        /// 连续线段
        /// </summary>
        struct PolylineObject
            : virtual public  PaintShapeObject
            , virtual public  QPolygon
        {
            DEFAULT_CONSTRUCT(PolylineObject)
                using QPolygon::QPolygon;
            PolylineObject(const QPolygon& p) :QPolygon(p) {}

            virtual void draw(QPainter& painter)
            {
```

```
                painter.setPen(*this);
                painter.drawPolyline(*this);
            }

            virtual QString exportSvgObject()
            {
                QString svg;
                svg += R"(<polyline points=")";
                for (auto& pt : *this)
                    svg += QString().sprintf("%d,%d ", pt.x(), pt.y());
                svg += QString().sprintf(R"(" stroke="#%02X%02X%02X"
stroke-width="%d" fill="none"/>)"
                    , QPen::color().red(), QPen::color().green(),
QPen::color().blue(), QPen::width());
                return svg;
            }
        };
#undef DEFAULT_CONSTRUCT
    }
}
```

画板操作，在 PaintWidget 类完成。

绘制模式

```
enum PaintMode : int
    {
        PAINT_NULL
        , PAINT_PENCIL            // 铅笔
        , PAINT_LINE              // 直线
        , PAINT_CIRCLE            // 圆形
        , PAINT_ELLIPSE           // 椭圆
        , PAINT_RECTANGLE         // 矩形
        , PAINT_ROUNDEDRECTANGLE  // 圆角矩形
        , PAINT_POLYGON           // 多边形
    };
```

鼠标响应

```
    void PaintWidget::mousePressEvent(QMouseEvent* event)
{
    switch (event->button())
    {
    case Qt::LeftButton:
        switch (paintMode)
        {
```

```
        case PaintWidget::PAINT_NULL:
            break;
        case PaintWidget::PAINT_PENCIL:
            paintObjList.push_back(new thatboy::qt::PolylineObject());
            if (onPaintObjectCreate)
                onPaintObjectCreate(PAINT_PENCIL, paintObjList.back());

    dynamic_cast<thatboy::qt::PolylineObject*>(paintObjList.back())->append(event
->pos());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);
            break;
        case PaintWidget::PAINT_LINE:
            paintObjList.push_back(new
thatboy::qt::LineObject(QLine(event->pos(), event->pos())));
            if (onPaintObjectCreate)
                onPaintObjectCreate(PAINT_LINE, paintObjList.back());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);
            break;
        case PaintWidget::PAINT_CIRCLE:
            paintObjList.push_back(new
thatboy::qt::CircleObject(QPoint(event->pos())));
            if (onPaintObjectCreate)
                onPaintObjectCreate(PAINT_CIRCLE, paintObjList.back());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);
            break;
        case PaintWidget::PAINT_ELLIPSE:
            paintObjList.push_back(new
thatboy::qt::EllipseObject(QRect(event->pos(), event->pos())));
            if (onPaintObjectCreate)
                onPaintObjectCreate(PAINT_ELLIPSE, paintObjList.back());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);
            break;
        case PaintWidget::PAINT_RECTANGLE:
            paintObjList.push_back(new
thatboy::qt::RectangleObject(QRect(event->pos(), event->pos())));
            if (onPaintObjectCreate)
```

```
                onPaintObjectCreate(PAINT_RECTANGLE, paintObjList.back());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);
            break;
        case PaintWidget::PAINT_ROUNDEDRECTANGLE:
            paintObjList.push_back(new
thatboy::qt::RoundedRectangleObject(QRect(event->pos(), event->pos())));
            if (onPaintObjectCreate)
                onPaintObjectCreate(PAINT_ROUNDEDRECTANGLE,
paintObjList.back());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);
            break;
        case PaintWidget::PAINT_POLYGON:

            if (continusStatus)
            {

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);

    dynamic_cast<thatboy::qt::PolygonObject*>(paintObjList.back())->append(event-
>pos());
            }
            else
            {
                continusStatus = true;
                paintObjList.push_back(new thatboy::qt::PolygonObject());
                if (onPaintObjectCreate)
                    onPaintObjectCreate(PAINT_POLYGON, paintObjList.back());

    dynamic_cast<thatboy::qt::PaintShapeObject*>(paintObjList.back())->QPen::oper
ator=(thisPen);

    dynamic_cast<thatboy::qt::PolygonObject*>(paintObjList.back())->append(event-
>pos());

    dynamic_cast<thatboy::qt::PolygonObject*>(paintObjList.back())->append(event-
>pos());
            }
            break;
        default:
```

```
            break;
        }
        break;

    case Qt::RightButton:
        continusStatus = false;
        break;
    default:
        break;
    }
}

void PaintWidget::mouseMoveEvent(QMouseEvent* event)
{
    if (event->buttons() & Qt::LeftButton)
    {
        switch (paintMode)
        {
        case PaintWidget::PAINT_NULL:
            break;
        case PaintWidget::PAINT_PENCIL:

    dynamic_cast<thatboy::qt::PolylineObject*>(paintObjList.back())->append(event
->pos());
            break;
        case PaintWidget::PAINT_LINE:

    dynamic_cast<thatboy::qt::LineObject*>(paintObjList.back())->setP2(event->pos
());
            break;
        case PaintWidget::PAINT_CIRCLE:
        {
            auto& circle =
*dynamic_cast<thatboy::qt::CircleObject*>(paintObjList.back());
            circle.setR(sqrt((event->pos().x() - circle.x()) * (event->pos().x() -
circle.x())
                + (event->pos().y() - circle.y()) * (event->pos().y() -
circle.y())));
        }
        break;
        case PaintWidget::PAINT_ELLIPSE:

    dynamic_cast<thatboy::qt::EllipseObject*>(paintObjList.back())->setBottomRigh
t(event->pos());
```

```
            break;
        case PaintWidget::PAINT_RECTANGLE:

    dynamic_cast<thatboy::qt::RectangleObject*>(paintObjList.back())->setBottomRi
ght(event->pos());
            break;
        case PaintWidget::PAINT_ROUNDEDRECTANGLE:

    dynamic_cast<thatboy::qt::RoundedRectangleObject*>(paintObjList.back())->setB
ottomRight(event->pos());
            break;
        case PaintWidget::PAINT_POLYGON:

    dynamic_cast<thatboy::qt::PolygonObject*>(paintObjList.back())->back() =
event->pos();
            break;
        default:
            break;
        }
        update();
    }
}
```

图像绘制

```
void PaintWidget::paintEvent(QPaintEvent* event)
{
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
    painter.fillRect(rect(), backColor);
    for (auto& obj : paintObjList)
        obj->draw(painter);

}
```

SVG 导出

```
QString PaintWidget::exportSvg() const
{
    QString svg;
    svg += R"(<!DOCTYPE html>
<html>
<body>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="800" height="800">
)";
    for (auto obj : paintObjList)
```

```
        svg += obj->exportSvgObject() + "\n";
    svg += R"(
</svg>

</body>
</html>)";
    return svg;

}
```

主窗口，在构造函数中完成初始化，连接槽函数

```
PaintX::PaintX(QWidget *parent)
    : QMainWindow(parent)
{
    ui.setupUi(this);

    auto sketchpadWidget = new PaintWidget(ui.sketchpad);

    // 导出
    connect(ui.actionExport, &QAction::triggered, [sketchpadWidget]
        {
            QFile file(QFileDialog::getSaveFileName(sketchpadWidget, "Export", "/",
"SVG Files(*.svg);;"), sketchpadWidget);
            file.open(QIODevice::WriteOnly| QIODevice::Text);
            if (file.isOpen())
            {
                file.write(sketchpadWidget->exportSvg().toStdString().c_str());
                file.close();
            }
        });

    // 颜色按钮
    ui.foreColorBtn->setAutoFillBackground(true);
    ui.backColorBtn->setAutoFillBackground(true);
    ui.foreColorBtn->setFlat(true);
    ui.backColorBtn->setFlat(true);
    ui.foreColorBtn->setPalette(sketchpadWidget->getForeColor());
    ui.backColorBtn->setPalette(sketchpadWidget->getBackColor());

    // 颜色选择
    connect(ui.foreColorBtn, &QPushButton::clicked, [=]
        {
            auto c =
QColorDialog::getColor(ui.foreColorBtn->palette().color(QPalette::Button));
```

```
            if (c.isValid())
            {
                ui.foreColorBtn->setPalette(c);
                sketchpadWidget->setForeColor(c);
            }
        });
    connect(ui.backColorBtn, &QPushButton::clicked, [=]
        {
            auto c =
QColorDialog::getColor(ui.backColorBtn->palette().color(QPalette::Button));
            if (c.isValid())
            {
                ui.backColorBtn->setPalette(c);
                sketchpadWidget->setBackColor(c);
            }
        });

    // 图层创建回调
    sketchpadWidget->setPaintObjectCreateCallBack([=](PaintWidget::PaintMode mode,
const thatboy::qt::PaintObject*)
        {
            switch (mode)
            {
            case PaintWidget::PAINT_NULL:
                break;
            case PaintWidget::PAINT_PENCIL:
                ui.paintObjectList->addItem("Penceil");
                break;
            case PaintWidget::PAINT_LINE:
                ui.paintObjectList->addItem("Line");
                break;
            case PaintWidget::PAINT_CIRCLE:
                ui.paintObjectList->addItem("Circle");
                break;
            case PaintWidget::PAINT_ELLIPSE:
                ui.paintObjectList->addItem("Ellipse");
                break;
            case PaintWidget::PAINT_RECTANGLE:
                ui.paintObjectList->addItem("Rectangle");
                break;
            case PaintWidget::PAINT_ROUNDEDRECTANGLE:
                ui.paintObjectList->addItem("RoundRectangle");
                break;
            case PaintWidget::PAINT_POLYGON:
```

```
                ui.paintObjectList->addItem("Polygon");
                break;
            default:
                break;
            }
        });

    // 画笔粗细
    ui.penWidthSlider->setMinimum(1);
    ui.penWidthSlider->setMaximum(50);
    ui.penWidthSpinBox->setMinimum(1);
    ui.penWidthSpinBox->setMaximum(50);
    ui.penWidthSlider->setSingleStep(1);
    ui.penWidthSpinBox->setSingleStep(1);
    // 同步SpinBox和Slider
    connect(ui.penWidthSpinBox,
static_cast<void(QSpinBox::*)(int)>(&QSpinBox::valueChanged), ui.penWidthSlider,
&QSlider::setValue);
    connect(ui.penWidthSlider, &QSlider::valueChanged, ui.penWidthSpinBox,
&QSpinBox::setValue);
    connect(ui.penWidthSlider, &QSlider::valueChanged, sketchpadWidget,
&PaintWidget::setPenWidth);

    // 取消其他按钮选中
    auto unCheckOtherActions = [=](QAction* showAction) {
        ui.actionPenceil->setChecked(false);
        ui.actionLine->setChecked(false);
        ui.actionRectangle->setChecked(false);
        ui.actionRoundRectangle->setChecked(false);
        ui.actionCircle->setChecked(false);
        ui.actionEllipse->setChecked(false);
        ui.actionPolgon->setChecked(false);
        if (showAction)
            showAction->setChecked(true);
    };

    // 按钮
    connect(ui.actionClear, &QAction::triggered, sketchpadWidget,
&PaintWidget::clearPaint);
    connect(ui.actionClear, &QAction::triggered, ui.paintObjectList,
&QListWidget::clear);
    connect(ui.actionArrow, &QAction::triggered, [=]
        {
            unCheckOtherActions(nullptr);
```

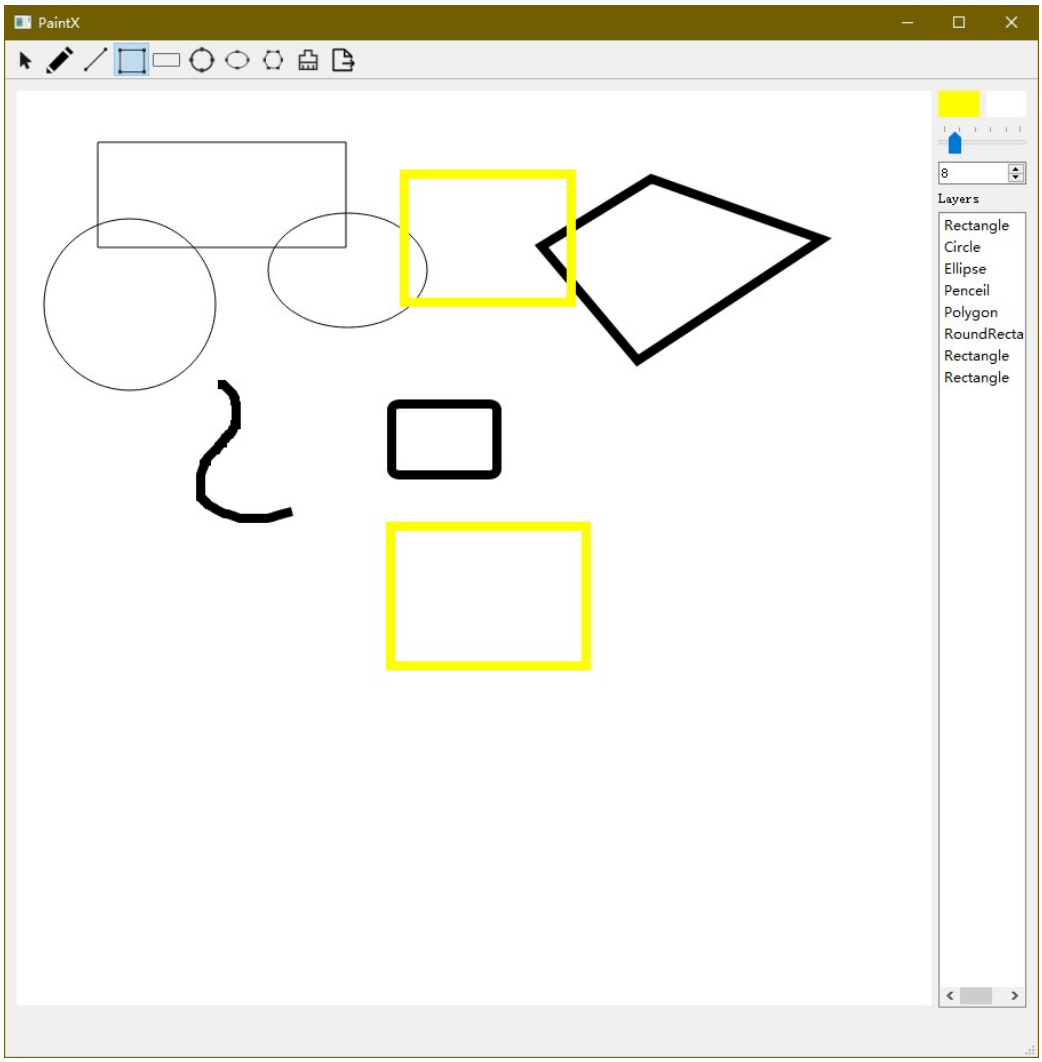
```
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_NULL);
        });
    connect(ui.actionPenceil, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionPenceil);
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_PENCIL);
        });
    connect(ui.actionLine, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionLine);
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_LINE);
        });
    connect(ui.actionRectangle, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionRectangle);
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_RECTANGLE);
        });
    connect(ui.actionRoundRectangle, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionRoundRectangle);

sketchpadWidget->switchPaintMode(PaintWidget::PAINT_ROUNDEDRECTANGLE);
        });
    connect(ui.actionCircle, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionCircle);
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_CIRCLE);
        });
    connect(ui.actionEllipse, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionEllipse);
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_ELLIPSE);
        });
    connect(ui.actionPolgon, &QAction::triggered, [=]
        {
            unCheckOtherActions(ui.actionPolgon);
            sketchpadWidget->switchPaintMode(PaintWidget::PAINT_POLYGON);
        });

}
```

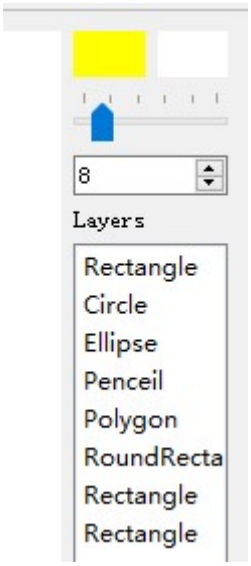
六、实训结果及说明（30 分）

1. 主界面内容如下，上为工具栏，右为侧边栏，内容区域为画板。

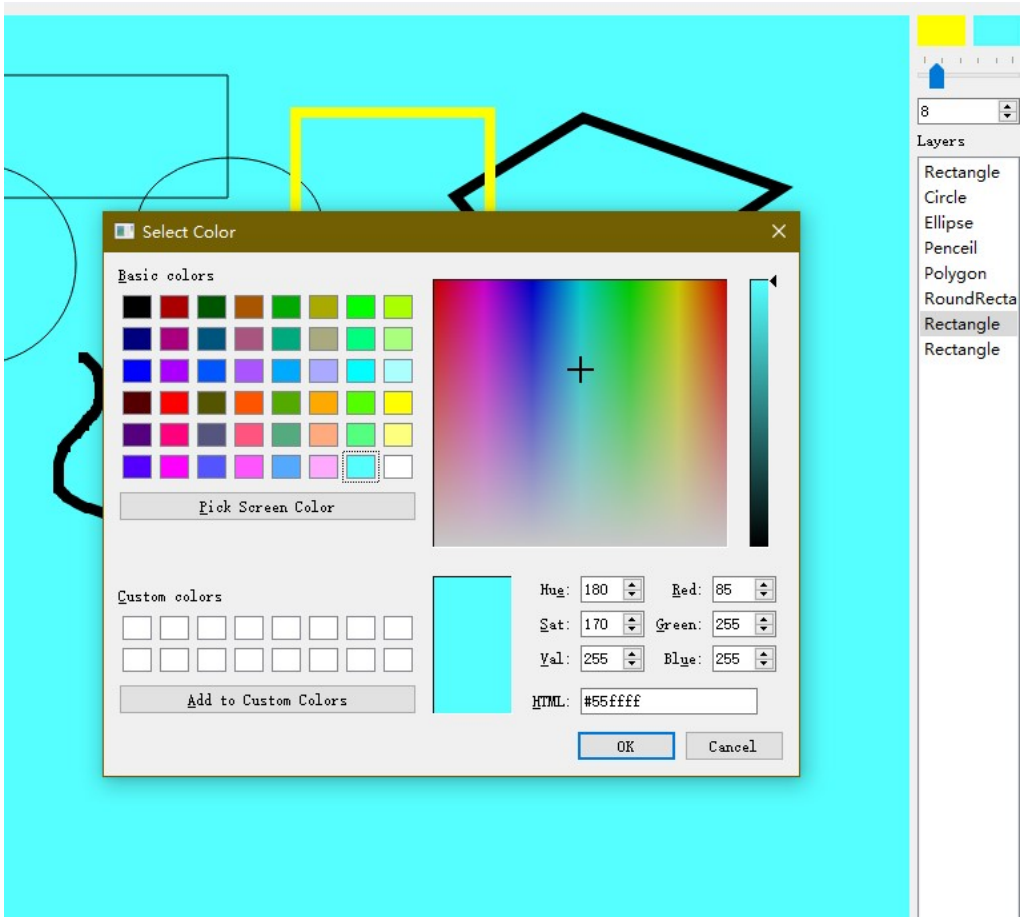


2. 工具栏，按钮依次为 指针（不绘制）、铅笔（涂鸦）、线段、矩形、圆角矩形、圆形、椭圆、多边形、清空画板、导出。除了指针、清空画板和导出，其他按钮为 checkable。

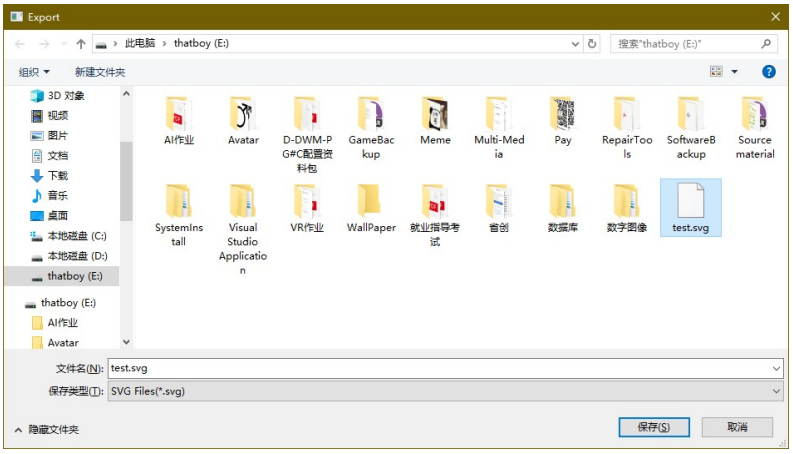
3. 侧边栏，最上方为前景和背景色，滑块和微调框用来设置画笔宽度，范围 1-50，Layers 列表展示对象列表（图层）。



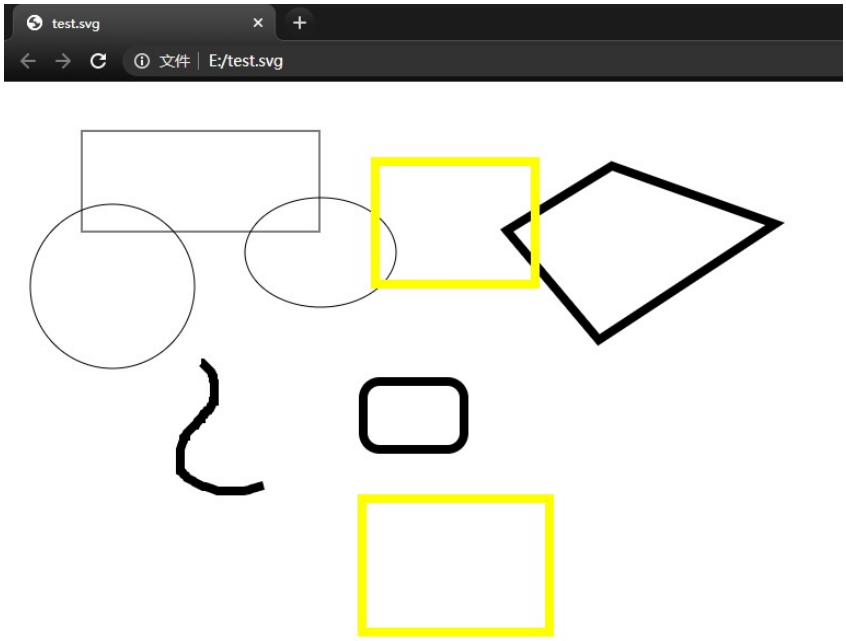
4. 修改颜色示意图。



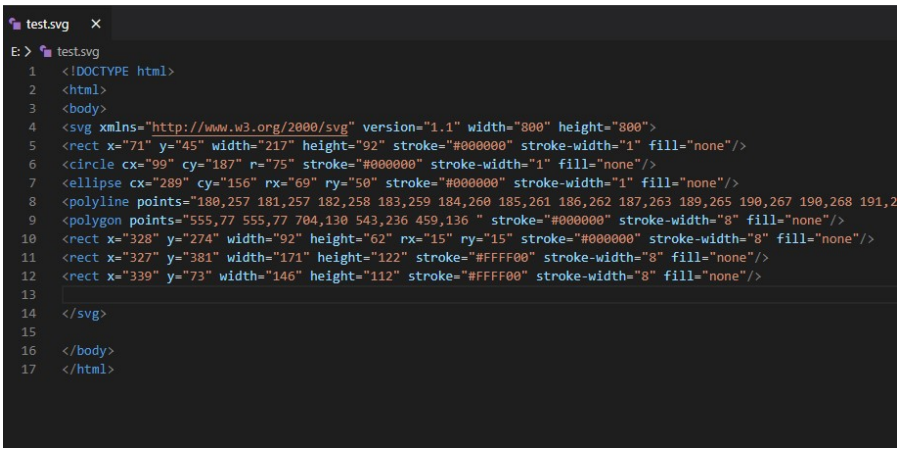
5. 图形导出操作示意图。



6. 导出图像查看（使用 chrome 浏览器）。



7. svg 图像源码（使用 VS Code）

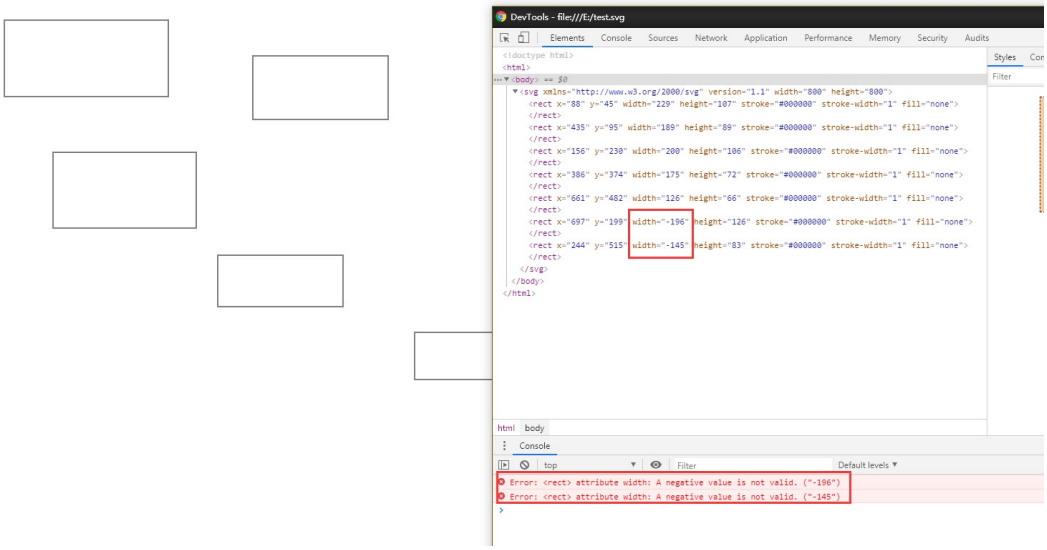


七、实训思考（10 分）

曾遇到问题：

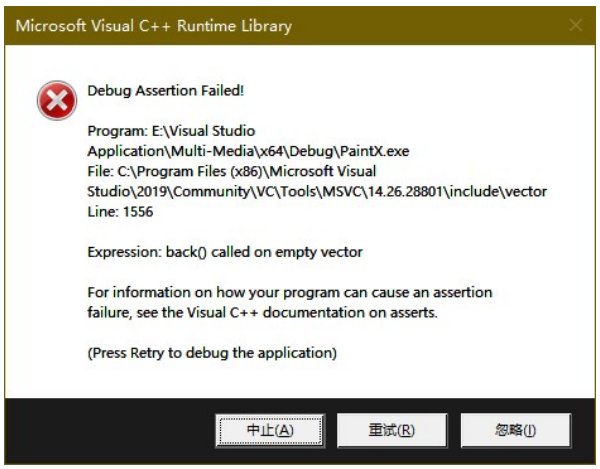
1. 如果拉伸出的图形（矩形、圆角矩形、椭圆）是反向的，即从右向左或从下到上绘制，则出现绘制正常，但导出显示错误；如图所示，可以看到错误信息为，宽度不能为负；

   解决办法：在导出时将继承自 QRect 的类归一化。



2. 在绘制多边形时，如果点击清空操作，然后接着点击画板开始绘制，则会出现向量下标越界异常；出现此问题的原因时，绘制多边形时，会设置当前为连续绘制模式，点击画板并不会插入新的图形，而是继续编辑上一次的图形，但是在清楚操作时遗漏了对于连续绘制的关闭，导致编辑上一次图形造成错误。

   解决办法：在清除操作中关闭连续绘图的状态。



拓展：

1. 增加撤回操作；
2. 增加对于右侧图层的编辑操作，点击可以显示此对象轮廓，右击可以删除等；
3. 增加导出 png、bmp 等图像；
4. 增加输入文字的功能；
5. 增加对 svg 图像的读取、解析、显示；
6. 增加插入图片，图片也可以以 Base64 编码的方式嵌入 svg 中。