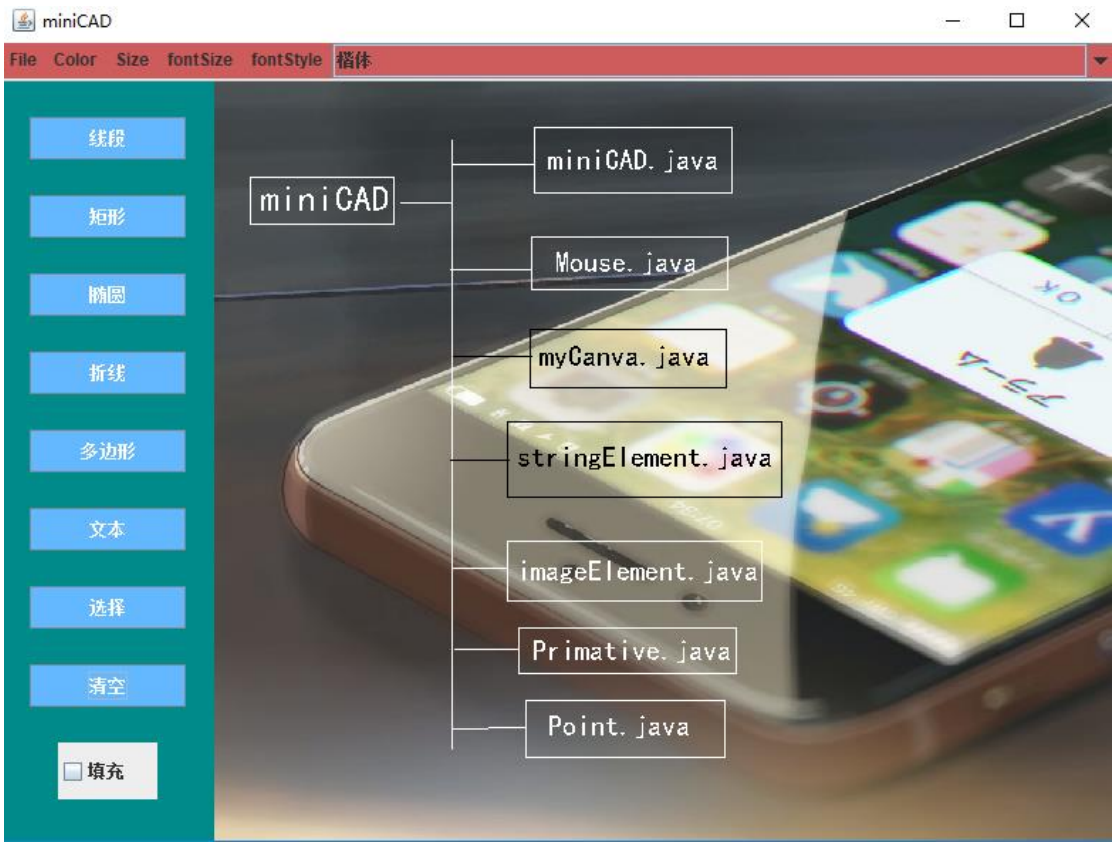
miniCAD

developed by
漆翔宇　3170104557
计算机科学与技术

Section1：项目基本组织结构概述

项目由七个 java 文件组成:



1. miniCAD.java

我们在 miniCAD.java 中实现了 miniCAD 的主框架：

①创建窗口

创建了窗口的顶层结构，并对窗口的基本属性进行了配置。

②为窗口添加了菜单栏

创建了一个菜单横栏，为每一个菜单选项设置了相应的事件监听和配套的事件处理。

File 栏下方，可以选择 Open,Save,Load Image,Quit。对应于加载文件，保存文件，加载图片(Swing 支持的各种格式都可以加载)，退出程序。

Color 栏下方可以选择颜色，在选中一个元素后，可以用它来改变其颜色。Color 栏选择一个颜色后，同时也会更新当前画笔的颜色，后续新画的元素也会应用这个颜色。

Size 栏下方选择的是画笔的粗细，和 Color 一样，既会更新当前选中元素的粗细，也会应用在当前画笔状态下。

fontSize 是选择字体的大小，fontStyle 选择字体的格式（正常/加粗/斜体）。最后的一个下拉栏可以选择字体的类型，程序在运行开始的时候会自动扫描运行环境下支持的字体类型，并全部显示在下拉栏中。

③在主窗口下创建了两个子面板

左边的墨绿色 button panel 用来作为 button 和 check box 的容器。右边的 canva 作为画板，作为绘制区。

④button panel 面板下我们添加了若干的按钮，以及一个 checkbox

我们为它们都在 miniCAD.java 种实现了相应的事件监听和处理流程。

点击相应的按钮，选择将要绘制的元素类型，或者切换到选择模式，或者清空面板。当填充选项被打上勾时，绘制的椭圆，矩形，多边形将会是被填充的。

⑤canva 面板

canva 面板是整个界面中最“敏感”的部位。

我们的 canva 面板是 myCanva 类的实例，myCanva 类是 JPanel 的子类，这个类实现在 myCanva.java 中。

我们在 miniCAD.java 中为这个面板添加了一个鼠标事件监听器，用来处理各种绘制，选择，移动，删除等操作。这个鼠标事件监听器 listener 是 Mouse 类的一个实例，Mouse 类实现了 MouseListener,mouseMotionListener,MouseWheelListener 三个接口，可以针对鼠标的行为充分的反映。

我们还对这个面板附加了一个 JTextFileld 类型的文本输入框，这个输入框是被隐藏的，但我们要在面板上添加文字的时候，会唤醒它，然后这个文本框接受我们的输入数据，在确定后，将输入内容整理成字符串返回，我们再收纳进我们的面板上。

2. myCanva.java

miniCAD 中，画板的主逻辑，它能容纳三种类型的元素，并在自己的界面上绘制。

三种元素分别是 ：Primative,stringElement,imageElement，对应图元，字符串，图片。

①三个 ArrayList 容器

分别容纳三种类型的元素。实际上，整个画板上的所有内容全都在这三个容器中。

②添加和删除元素的接口

这个接口可以提供给其他模块，直接向画板添加元素，或者从上面删除元素。

③导入和导出文件的接口
我们自定义了一套简单的文件标准。导出为文件的时候，把三个容器中的元素全部都以自己的格式导出为字符串。导入的时候，解析读入的字符串，创建对应的元素对象，放回容器中。
④管理字符串输入的接口。
激活我们之前隐藏的文本输入框到对应位置，接受文本输入，然后创建对应的 stringElement 对象来管理这个串。
⑤选择判定
用来检查当前点击等操作对应位置是否对应于一个图形元素。用于检测元素的选择。选中后会返回对应选中的元素。
⑥元素的更改接口
外部的模块可以通过 canva 来对上面的元素进行属性修改。

3.Primative.java / stringElement.java / imageElement.java
管理出现在画板上的三种元素对象。Primative 指的是图元，基本的几何形状都是由这个类管理。stringElement 指的是字符串，管理字符串对象。imageElement 指图片，用来管理图片对象。元素的各种属性以及属性的修改方法，绘制方法，导出为字符串的方法（toString 方法把对象的属性以一定格式转换成文本，我们在导出文件的时候会用到）等都封装在了对应的类中。画板类在绘制的时候，只需要把容器中的各个对象拿出来，分别调用各自的绘制方法即可。导出文件的时候，也只需要直接输出各个图形对象，由各自的 toString 方法自动化转换。

4.Mouse.java
用来响应鼠标事件，类似于一个状态机。
不同的状态下，对鼠标事件作出不同的响应，主要是和 canva 交互，在画板上添加，删除或者修改元素。

5.Point.java
封装的一个管理二维点对的类，做一些基本的二维向量操作。我们在管理元素属性的时候会用到它。

Section2：miniCAD 基本设计思路

```
public class miniCAD
{
    static private JFrame frame;
    static  JPanel buttonPanel;
    static myCanva canva;
    static Mouse listener;
    static JTextField textBlock = new JTextField(8);

    static boolean filled=false;
    static String state="choose";
    static int fontStyle=0;
    static int fontSize=25;
    static String fontType="楷体";

    static Color currentColor=new Color(0, 0, 0);
    static Color  menuColor = new Color(205,92,92);
    static int currentStroke = 1;
    static Stroke chosenBrushSize[] = new Stroke[11];
    static Stroke brushSize[] = new Stroke[11];

    public static void main(String[] args)
    {…
    }
    static void enumFonts(JMenuBar target)
    {…
    }

    static void CrateMenuBar()
    {…
    }

    static void InsertButton(JButton x,JPanel y)
    {…
    }

    static void LayoutPanel()
    {…
    }
}
```

1. 上下文

miniCAD 类中，全是静态方法和静态成员。在我们的设计思路中，它就是一个顶层的全局环境，而不是一个可以实例化多次的对象。我们借鉴了 OpenGL 中的设计思路，把整个 miniCAD 设计成一个类似于状态机的结构。

miniCAD 中的静态数据成员就是整个绘制环境的上下文：四个界面上的子区域，监听器，当前是否选择了填充模式，当前字体类型，风格，大小，画笔的绘制模式，颜色。同时静态成员中还有我们预建的字体和颜色，作为标准的字体和颜色类型，可以直接被应用。

miniCAD 的静态方法，全部会在程序被启动时，由 main 调用，完成整个界面结构的搭建和上下文的搭建。

miniCAD.java 就是由 miniCAD 类和一大堆事件处理类构成的。整个环境都在 miniCAD 类中，下面的一大堆类，都是为 miniCAD 整个上下文服务的，处理 button，checkbox，menuItem 等组件的触发事件。这套事件处理机制可以对用户的各种选项设定做出响应，并最后作用在 miniCAD 上下文上。

miniCAD 的最终绘制图形的区域非常敏感，有大量的事件需要响应，并且要组织，管理和绘制各个图形对象。我们剩下的六个 java 文件都是为这个绘制区域服务的。图形绘制区域的工作会基于我们前面创建的上下文进行。

2.图形元素

我们最后在绘制区域呈现的内容，被分为三种类型：几何图元，文本，图片。

我们分别封装在了三个类中，在第一部分我们已经介绍过了。一个图形元素的所有属性都封装在了自己体内并且该对象提供了相应的去修改它们属性的接口以及绘制它们的接口。和图形本身相关的东西全都在图形元素对象内部，我们在外部创建好它们之后，通过这些接口去和它们进行交互，完成修改和绘制。

3.画板

画板本质上就是一个容纳上述图形元素的容器。实例化的图形元素被存在画板上。画板做的就是枚举每一个图形元素对象，调用它们的绘制方法，把它呈现在自己的图形化界面上。

这样的画板是静态的，要实现交互，我们需要提供一些接口，来选定上面的元素，修改上面的元素，并且在上面添加或者删除元素。

我们在点击它的图形界面时要怎么去选定它上面的元素呢？画板提供了一个扫描机制，通过拿被点击的位置和每个图形元素去匹配，匹配上后就返回这个被选中的图形元素，然后我们就可以对这个返回元素进行操作，相应的在画板调用这个对象的绘制方法时，它在图形界面上的表现也会改变。

而点击位置和图形元素的匹配，同样，也封装在了图形元素对象中。我们只需要不断枚举每个图形元素，调用它们的匹配方法即可。

添加和删除机制也非常简单，通过一个接口，外部模块向画板的容器 push 和 pop 图形元素对象即可。

4.事件监听器

图形类提供了画板和它进行交互的接口。画板又提供了其他外部模块和它进行交互的接口。对于用户来说，画板的图形界面是他直接与 miniCAD 进行交互的接口。但是人体是没办法调用画板的接口的。

人体只能操作鼠标和键盘，鼠标和键盘不断的在图形界面上触发事件。

所以我们需要建立一个额外的抽象层来建立鼠标键盘事件和画板的交流。

事件监听器从 miniCAD 类获取当前的上下文，根据上下文来决定对于当前的事件应该向画板发送怎样的消息。

Section3 : miniCAD 一些重要的实现细节

1.选择判定

选择判定是直接封装在一个图形元素中的。给定一个点，判定这个点是否在元素内部。

对于矩形来说，这非常的直观。

对于椭圆来说，我们近似用了它的矩形框来判定。

对于线段，我们用这个点指向线段的两个端点的夹角来判定，当角度大于 160 度时，我们认为它是在线段上的。

对于折线，只需要满足在和其中任一一个线段匹配即可。

多边形，我们按照顺序遍历每一个顶点，当前点到顶点如旋转的有向角度为 360 即为一个内部点，如果为 0 度，即为一个外部点。考虑到计算的精度误差，我们允许误差范围在 20 度内。

对于图形，在其矩形框内即可。

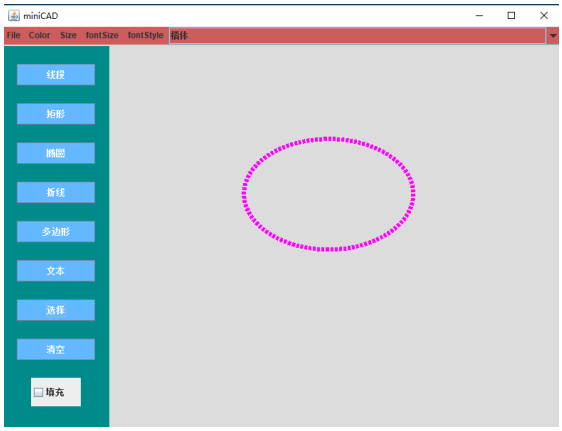
对于文本，我们会在创建文本的时候为它订制一个矩形框，包裹着这串文字。问题转换为和矩形做选择判断。

2.选中对象的特殊表现

选中一个对象的时候，我们希望它能变化，如变色，线条变粗之类的特征。这样可以让我们更直观的看到哪个对象被我们选中了。这需要我们在每一个图形对象中封装一个

boolean 参数，来标注它是否被选中。在 draw 方法中，标记为选中的对象，应该调用另一套绘制流程，来区别于没被选中时的状态。

对于几何对象，选中时，无论是实心还是空心的，我们都让它变成空心的，并且让它的边框变成虚线。

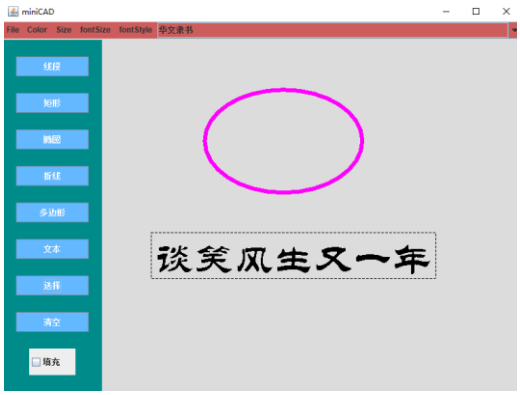


对于虚线型 stroke，我们预建一个 arr 数组，在创建的时候以它为参数即可。

```
float[] arr = {4.0f,2.0f};
BasicStroke(x,BasicStroke.CAP_BUTT,BasicStroke.JOIN_BEVEL,1.0f,arr,0);
```

这部分我们在创建上下文时已经预先建好了，在后面绘制的时候直接使用即可。

对于文字对象，我们前面提到了，我们为它们都建了一个矩形框来包裹，用于做选择判定，这里我们让它被选中时，把矩形框也虚线化的显示出来。



3.放大和缩小该怎么办

在处理放大和缩小的时候，我遇到了一个很棘手的问题。我希望我在滑动鼠标滚轮的时候，对象是按比例放大和缩小的，这势必需要乘上一个缩放因子，而这个缩放因子势必会使得计算产生实数。而我在这之前都是用整数坐标来表示的元素。如果简单的使用四舍五入来做计算，我们还会发现当我们在缩小再放大回来的时候，产生惊人的累计误差。。这不仅会影响大小，还会影响形状。。。

所以最后对于需要连续放大和缩小的对象，我们都是用的实数来保存它的坐标，尽管它们最后都对应于一个整数域的像素阵列。。

4.字符串应该怎么处理?

我们直接借用 swing 自带的文本输入框来为我们解决这个问题。我们在建立窗口的时候，就创建一个大小合适的文本输入框并把它隐藏起来（setVisible(false)）。它不会对我们产生任何影响。当我们要添加文本的时候，我们把它的位置修改在对应的地方，并让它显示。这样就可以在这个地方进行输入。当输入完成后，我们从文本输入框用 swing 给我们提供的一套方法把里面的内容提取出来，我们这个时候，才实际的建立 stringElement 对象，并且把它 push 到画板上显示。

当我们双击这个 stringElement 对象时，对其进行修改，stringElement 中一个内建的 boolean 参数被激活，其绘制方法检测到这个参数时，就会停止绘制自身。这个时候，我们相当于隐藏了这个画板上的 stringElement 对象。我们再告诉画板，激活输入框，这时，原位置，又变成了输入框，我们输入完毕后，把新的内容送进 stringElement 对象，并重新 enable 它的内建修改参数。然后其绘制方法又会正常的绘制自己了。

为了防止空间被无端的浪费，在文本输入框中的字符串被导出后，我们会检测这是否是一个空字符串，如果是空的，我们不会为它建立 stringElement 对象。

5.导入和导出

先按照一定的规则，覆写三个图形元素对象的 toString 方法。导出的时候，直接把画板中所有的对象都输出。导入文本的时候，再解析字符串，重新建立对应的元素对象，把解析来的信息交给构造函数，构造出对应的图形元素，送入画板。

6.判断导入的文件是否为图片?

非可识别图片文件，最后加载进来，长宽都是-1。这部分对象不会被创建。

Section4 : miniCAD 使用介绍

1.几何图形绘制

点击对应的按钮，即可进入相应的几何图形绘制模式。

在绘制线段，矩形，椭圆的时候，只需要指定两个点，点击第一个点的时候，这个点被扎在画板上，第二个点会实时的跟着鼠标移动走，并绘制临时的图形，直到我们第二次点击鼠标指定最终点。

绘制折线和多边形时，点的个数是不定的。所以可以一直用鼠标左键点击屏幕连续的绘制。。。这个时候需要按鼠标右键停止绘制。

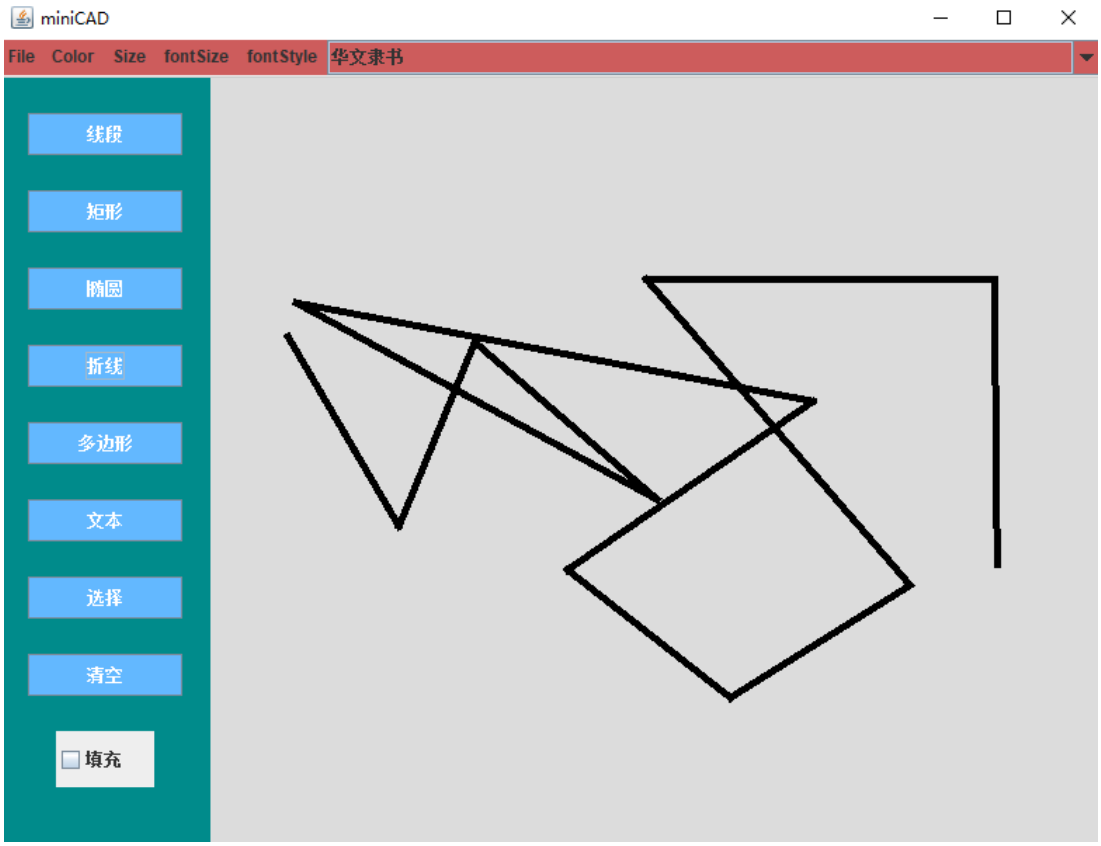
选择模式下，点击选中后:

可以通过菜单栏的 Color，size 修改颜色与尺寸。

可以用鼠标拖拽移动图形位置。

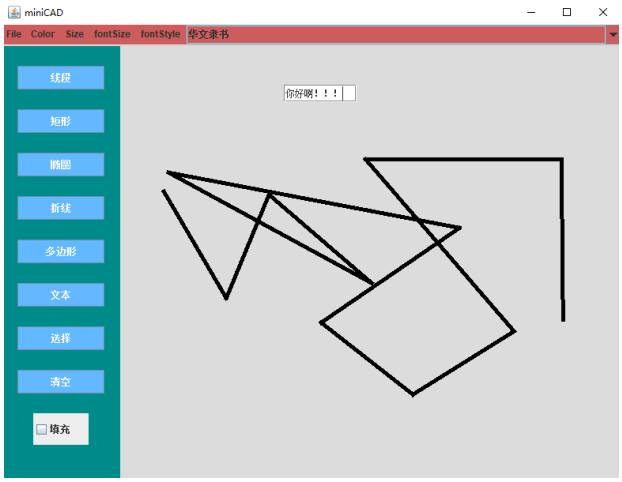
可以通过点击鼠标右键直接删除。

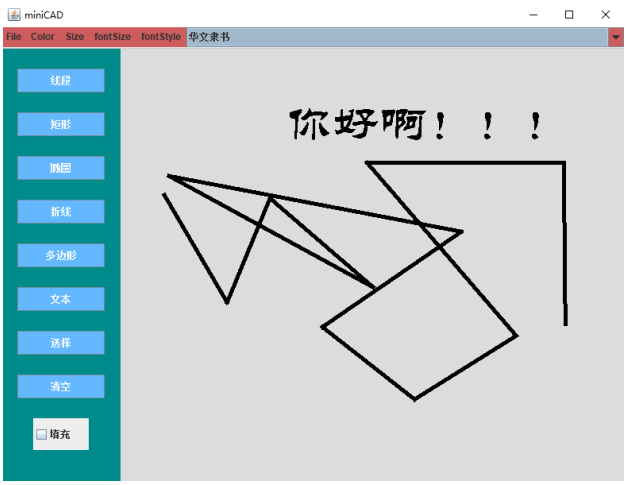
可以滑动鼠标滑轮修改大小（以鼠标当前位置为坐标原点做缩放）。



2.文本添加

进入文本模式后，点击图形区域，就会出现输入框，输入内容后，点击输入框外部区域或者按下回车即可结束输入，产生实际的输入文本。选择模式下，点击选中，可以通过菜单栏修改字体大小，类型，风格，拖拽可以移动位置。双击文本，会再次出现文本框，可以在这里修改文本内容。选中时点击鼠标右键可以删除文本。



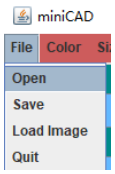


3.图片加载

点击菜单栏 File->LoadFile，选择图片文件，即可加载图片。图片默认会按照原图比例做适当的缩放加载到绘制区域的左上角。

选择模式下点击选中，拖拽可以移动，鼠标滑轮可以等比例的放缩图片大小，鼠标右键再次点击删除图片。

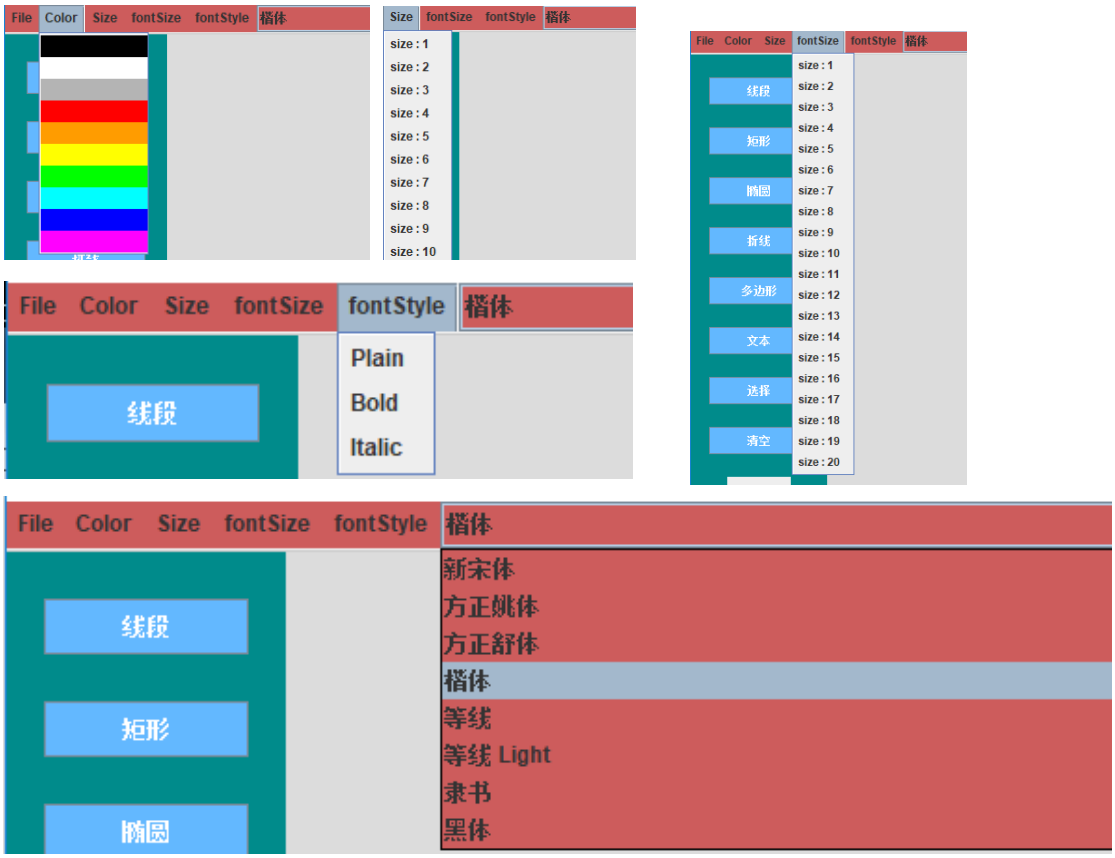
4.导入和导出文件



Open 可以导入我们指定格式的文件，Save 可以导出。

但是应该注意的是，我们导出文件时，图片类型，我们导出的是对应的路径，如果想要加载，需要确保图片依旧在原来的路径下。这里导是绝对路径，如果要想实现移植性，可能需要手动修改一下导出文件的路径。（或者下一个版本全部统一修改成使用相对路径。。。）

5.颜色/画笔粗细/字体大小/字体风格/字体类型



6.关于编码可能出现的问题。。

miniCAD.java 是用以 GBK 格式保存的，主要是因为 miniCAD.java 中出现了中文（在 vscode 下，格式比较麻烦，所以索性用了 GBK。。）所以如果要查看的话，需要使用 GBK 编码打开，否则在代码中，按钮那部分的名称你可能会看到乱码。在 windows 下，用记事本打开或者 vscode 打开（但是设定编码为 GBK）可以正常阅读代码。

由于代码是在 windows 下编写，在其他平台上可能会遇到换行符之类的麻烦。。

由于按钮上出现了中文，不保证在 Linux 或者 Mac 下按钮上的中文还能正常显示。