MiniCAD 实验报告

明确需求

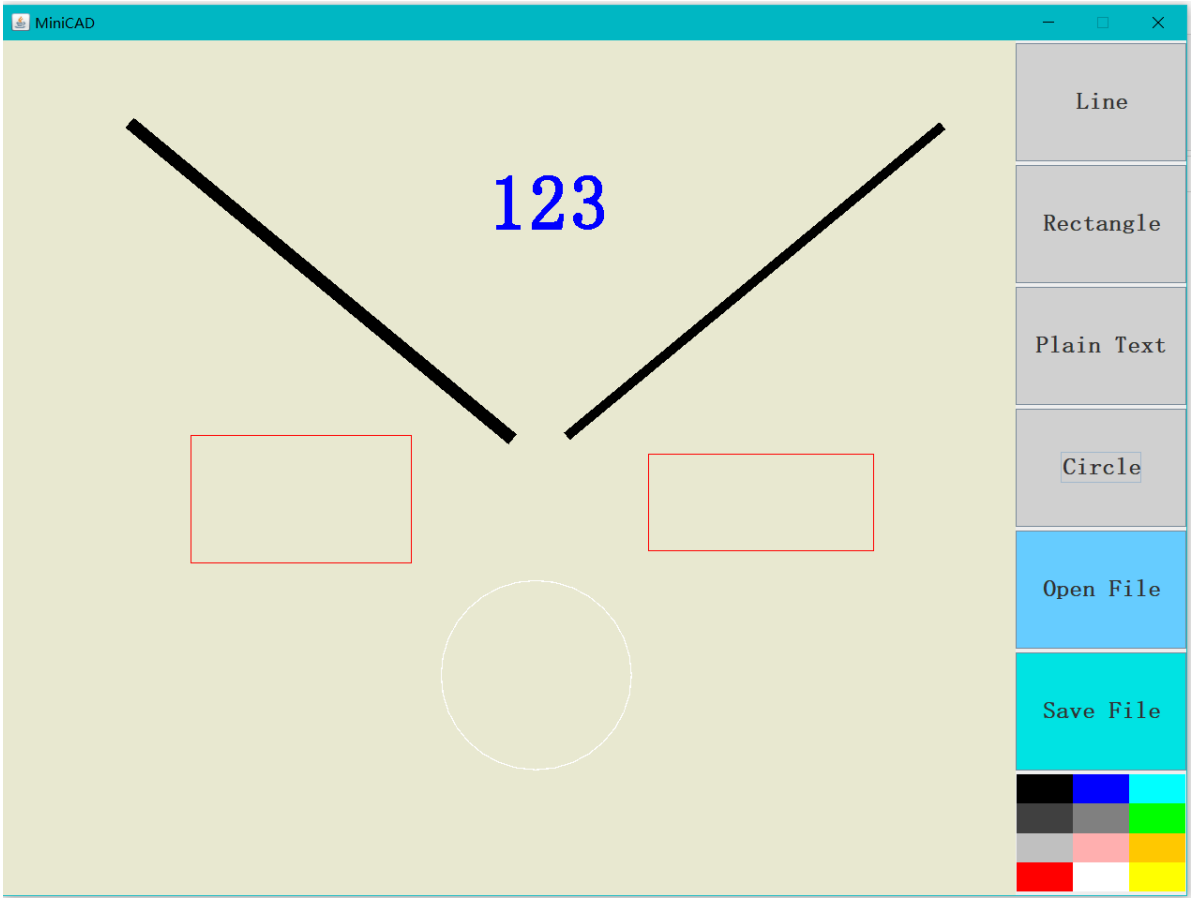
做一个简单的绘图工具，以CAD的方式操作，能放置直线、矩形、圆和文字，能选中图形，修改参数，如颜色等，能拖动图形和调整大小，可以保存和恢复。

在本项目中，具体操作实现如下：

1. 打开程序后，用户可通过选择界面右侧的Line、Rectangle、Plain Text和Circle按钮，然后通过鼠标拖拽在界面正中的画布上自由绘制图形。
2. 在已绘制的图形的区域内单击，可以选中该图形，选中的图形的外观会被略微加粗。随后，用户可以：

- 点击界面右下角的调色板，更改选中的图形的颜色；
- 使用键盘上的`,`或`<`键使选中的图形的尺寸缩小，使用`.`或`>`键使其尺寸放大，使用`-`或`_`键使其加粗，使用`=`或`+`键使其变细；
- 使用`r`或`R`键，删除选中的图形；
- 鼠标拖拽选中的图形，修改图形的位置。

3. 用户可通过选择界面右侧的Save File按钮，将当前绘制的画布上的图形以`.cad`文件的方式存入文件系统；或者，选择`Open File`按钮，将画布上的图形加载为文件系统上的某个`.cad`文件。



总体架构

源代码结构示意图如下：

```
└─minicad
    |  App.java
    |  Canvas.java
    |
    ├─shapes
    |      Circle.java
    |      Line.java
    |      Rectangle.java
    |      Shape.java
    |      Text.java
    |
    ├─toolbar
    |      Operation.java
    |      Toolbar.java
    |
    └─utils  // 这些类只提供静态方法
            IOHandler.java
            ShapeManager.java
```

- `App`：程序入口，主窗体，View。
- `Canvas`：画布，继承`JPanel`，View、Controller。注册了鼠标、键盘操作的多个回调，实现了绘制四种`shape`的方法。
- `shapes.Shape`：抽象图形类，Model。实现了可串行化。
- `shapes.Circle, shapes.Line, shapes.Rectangle, shapes.Text`：圆、线段、矩形和文本，均继承自`Shape`。
- `toolbar.Operation`：用户操作的枚举类，包括`DRAW_LINE, DRAW_RECTANGLE, DRAW_PLAIN_TEXT, DRAW_CIRCLE, OPEN_FILE, SAVE_FILE`。
- `toolbar.Toolbar`：工具栏，继承`JPanel`，View、Controller。注册了选择操作、颜色以及打开、保存文件的按钮的回调。
- `utils.IOHandler`：提供打开文件、保存文件的静态方法。
- `utils.ShapeManager`：当前画布上所有图形的容器，使用单一`ArrayList<Shape>`静态变量完成，Model。也提供对该容器的各种访问方法。

实验步骤

编写App主窗体

在`App`类中，提供`run`方法，完成`appFrame`的配置：

```
private static JFrame appFrame = new JFrame(TITLE);
public static void run() {
    Canvas canvas = new Canvas();
    appFrame.setSize(WINDOW_SIZE_X, WINDOW_SIZE_Y);
    appFrame.setResizable(false);
    appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    appFrame.setLayout(new BorderLayout());
    appFrame.add(canvas, BorderLayout.CENTER);
    appFrame.add(canvas.getToolbar(), BorderLayout.EAST);
    appFrame.setLocationRelativeTo(null);
    appFrame.setVisible(true);
}

public static void main(String[] args) {
    App.run();
```

```
}
```

编写Canvas和Toolbar的视图

`Toolbar`依赖于`Canvas`存在，`Canvas`应有能力设置自己的`Toolbar`，因此将`Toolbar`设为`Canvas`的成员变量。在`Canvas`

```
private Toolbar toolbar;

public Canvas() {
    this.toolbar = new Toolbar(this);
    this.setBackground(new Color(0xE8, 0xE8, 0xD0));  // This color should be
good
    this.setFocusable(true);
    this.addMouseMotionListener( ... );  // Controller stub
    this.addMouseListener( ... );
    this.addKeyListener( ... );
}
```

在`Toolbar`中，加入操作按钮和调色板（事先实现一下简单的`Operation`枚举类）：

```
public Toolbar(Canvas canvas) {
    this.setPreferredSize(new Dimension(App.WINDOW_SIZE_X / BUTTON_NUM,
App.WINDOW_SIZE_X));
    this.setLayout(new GridLayout(BUTTON_NUM, 1, 3, 3));
    this.setBackground(canvas.getBackground());
    EnumSet<Operation> operations = EnumSet.allOf(Operation.class);
    for (Operation operation : operations) {
        JButton operationButton = new JButton(operation.toString());
        operationButton.setFont(new Font(FONT_NAME, Font.BOLD, 20));
        // register event listener
        if (operation == Operation.OPEN_FILE) {
            operationButton.setBackground(new Color(0x66, 0xCC, 0xFF));
            operationButton.addActionListener( ... );
        } else if (operation == Operation.SAVE_FILE) {
            operationButton.setBackground(new Color(0x00, 0xE3, 0xE3));
            operationButton.addActionListener( ... );
        } else {
            operationButton.setBackground(new Color(0xD0, 0xD0, 0xD0));
            operationButton.addActionListener( ... );
        }
        this.add(operationButton);
    }

    // add colorPanel
    JPanel colorPanel = new JPanel();
    colorPanel.setLayout(new GridLayout(4, 3));
    for (Color color : COLORS) {
        JButton colorButton = new JButton();
        colorButton.setBackground(color);
        colorButton.setOpaque(true);
        colorButton.setBorderPainted(false);
        colorButton.addActionListener( ... );
        colorPanel.add(colorButton);
    }
    this.add(colorPanel);
```

```
}
```

编写shapes和ShapeManager

首先完成抽象类`Shape`的编写：

```
public abstract class Shape implements Serializable {
    private Color color = Color.BLACK;
    private float strokeWidth = 1.0f; // 轮廓厚度（API require float）
    private transient boolean selected = false;  // 串行化时不用存selected

    public void draw(Graphics2D graphics2d) {
        graphics2d.setColor(this.color);
        Stroke stroke = null;
        if (this.selected == true) {
            stroke = new BasicStroke(this.strokeWidth + 1.0f); // highlight
selected
        } else {
            stroke = new BasicStroke(this.strokeWidth);
        }
        graphics2d.setStroke(stroke);
    }

    public void incWidth() {
        this.strokeWidth++;
    }

    public void decWidth() {
        this.strokeWidth = Math.max(this.strokeWidth - 1.0f, 1.0f);
    }

    public abstract void moveTo(int dx, int dy);

    public abstract void expand();

    public abstract void shrink();

    public abstract boolean containsPoint(Point point);
}
```

然后编写四种形状，它们都需要Override父类的`draw`方法，并实现所有抽象方法。在实现`containsPoint, expand, shrink`时，可能需要用到高中数学的解析几何知识。在确定线段`Line`是否能被鼠标点击选中时，以及进行其他由于人类鼠标点击而导致不够"精确"的操作时，需要添加容许的偏移量`TOLERANCE`，以提升用户友好度。由于`Canvas`中注册的鼠标点击回调方法也需要用到该`TOLERANCE`，故将`TOLERANCE`放到了`Canvas`类的常量中，其值暂且设定为`5`。

`ShapeManager`的实现较简单，设计一个静态容器变量即可：

```
private static ArrayList<Shape> shapes = new ArrayList<>();
```

后续将在该工具类中，根据业务逻辑加入更多方法。

编写Controllers

编写绘制形状的方法

在`Canvas`中实现`drawShape`方法，根据`toolbar`给出的`curSelectedOperation`确定绘制的形状。

```
public void drawShape() {
    Operation operation = this.toolbar.getCurSelectedOperation();
    Shape shape = null;
    switch (operation) {
    case DRAW_LINE: {
        shape = new Line(new Point(mouseStart), new Point(mouseEnd));
        break;
    }
    case DRAW_RECTANGLE: {
        double tmpX, tmpY, tmpWidth, tmpHeight;
        // ... calc values
        shape = new Rectangle(new Point((int) tmpX, (int) tmpY), tmpWidth,
tmpHeight);
        break;
    }
    case DRAW_CIRCLE: {
        double tmpX, tmpY, tmpDoubleRadius;
        // ... calc values
        shape = new Circle(new Point((int) tmpX, (int) tmpY), tmpDoubleRadius);
        break;
    }
    case DRAW_PLAIN_TEXT: {
        shape = new Text(new Point(mouseStart.x, mouseStart.y),
                JOptionPane.showInputDialog("Input the text you want to show:
"));
        break;
    }
    case OPEN_FILE:
        return;
    case SAVE_FILE:
        return;
    }
    ShapeManager.addShape(shape);
    this.repaint();
}
```

然后，重写`paintComponent`方法，实现界面加载重绘：

```
@Override
public void paintComponent(Graphics graphics) {
    super.paintComponent(graphics);
    ShapeManager.drawAll((Graphics2D) graphics);
}
```

编写事件监听回调

补充之前在`Canvas`和`Toolbar`中省略的`EventListener`。若匿名类中只需要重写一个接口方法，则可用轻便的Lambda表达式代替`new`关键字和方法体。此处代码不再一一列出，详见源代码。

编写文件操作方法

文件操作分为视图和逻辑两部分。在`App`中，可实现视图和部分判空逻辑：

```
public static void handleOpenFile() {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter("CAD files
(*.cad)", "cad");
    chooser.setFileFilter(filter);
    chooser.setDialogTitle("Open a .cad File");

    int value = chooser.showOpenDialog(appFrame);
    if (value == JFileChooser.APPROVE_OPTION) {
        File openFile = chooser.getSelectedFile();
        if (openFile.exists() == true) {
            String path = openFile.getAbsolutePath();
            IOHandler.openFile(path);
        } else {
            JOptionPane.showMessageDialog(null, "Cannot open file. File name not
exist!", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}

public static void handleSaveFile() {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter("CAD files
(*.cad)", "cad");
    chooser.setFileFilter(filter);
    chooser.setDialogTitle("Save Current File");
    chooser.setSelectedFile(new File(DEFAULT_FILENAME));

    int value = chooser.showSaveDialog(appFrame);
    if (value == JFileChooser.APPROVE_OPTION) {
        File newFile = chooser.getSelectedFile();
        if (newFile.exists() == false) {
            try {
                newFile.createNewFile();
                String path = newFile.getAbsolutePath();
                IOHandler.saveFile(path);
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            JOptionPane.showMessageDialog(null, "Cannot save current file. File
name already exist!", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

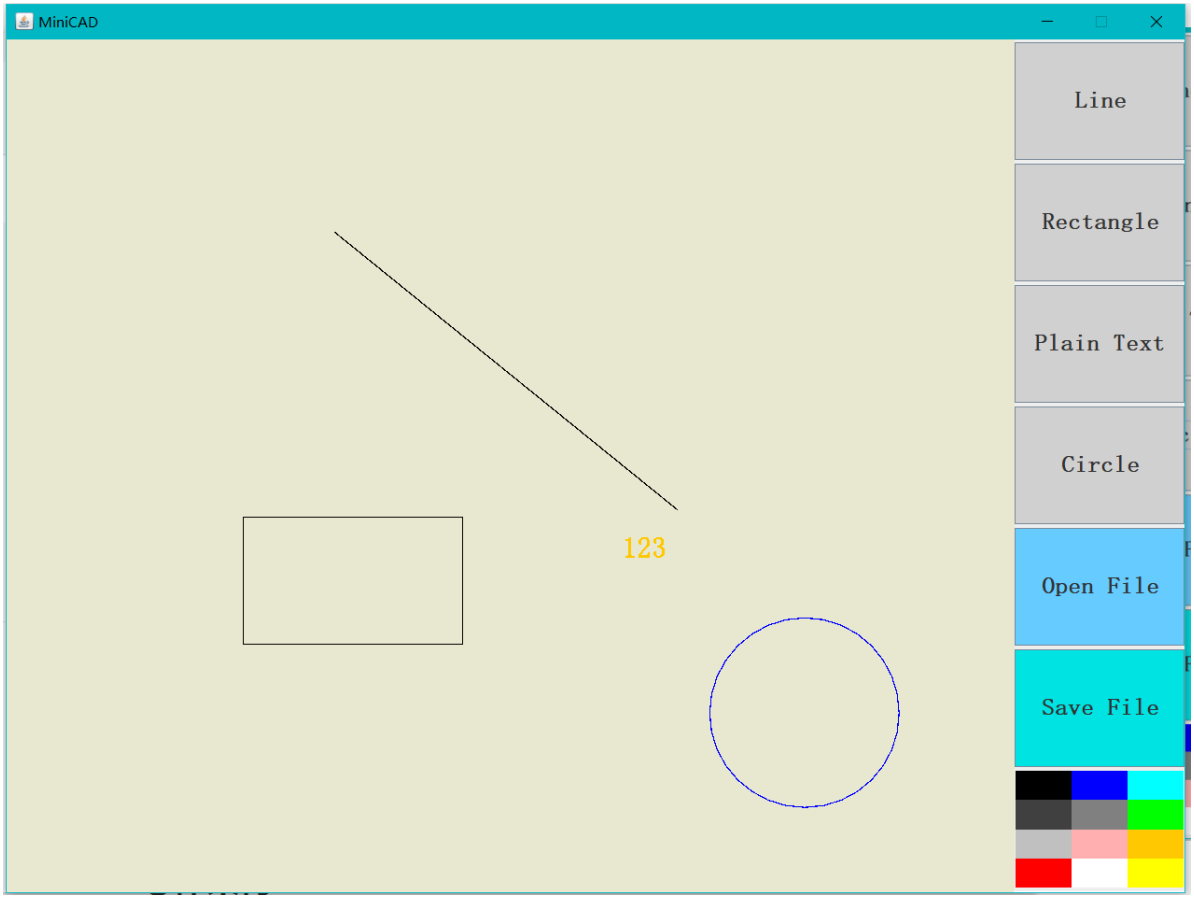
在`IOHandler`中，实现数据流的写入和读取逻辑，直接使用`ObjectInputStream`和`ObjectOutputStream`，将`Model`的集合`ShapeManager.shapes`读取/写入即可：

```
@SuppressWarnings("unchecked")  // 泛型erasure的后果
public static void openFile(String path) {
    ObjectInputStream in;
    try {
        in = new ObjectInputStream(new FileInputStream(path));

        ArrayList<Shape> readShapes = (ArrayList<Shape>) in.readObject();  // 泛
型erasure的后果

        ShapeManager.setShapes(readShapes);
        in.close();
        JOptionPane.showMessageDialog(null, "Successfully open file", "OK",
JOptionPane.INFORMATION_MESSAGE);
    } catch (/* mutli-catch ommited */) {
        // ...
    }
}

public static void saveFile(String path) {
    ObjectOutputStream out;
    try {
        out = new ObjectOutputStream(new FileOutputStream(path));
        out.writeObject(ShapeManager.getShapes());
        out.close();
        JOptionPane.showMessageDialog(null, "Successfully save file", "OK",
JOptionPane.INFORMATION_MESSAGE);
    } catch (/* mutli-catch ommited */) {
        // ...
    }
}
```

实验结果演示

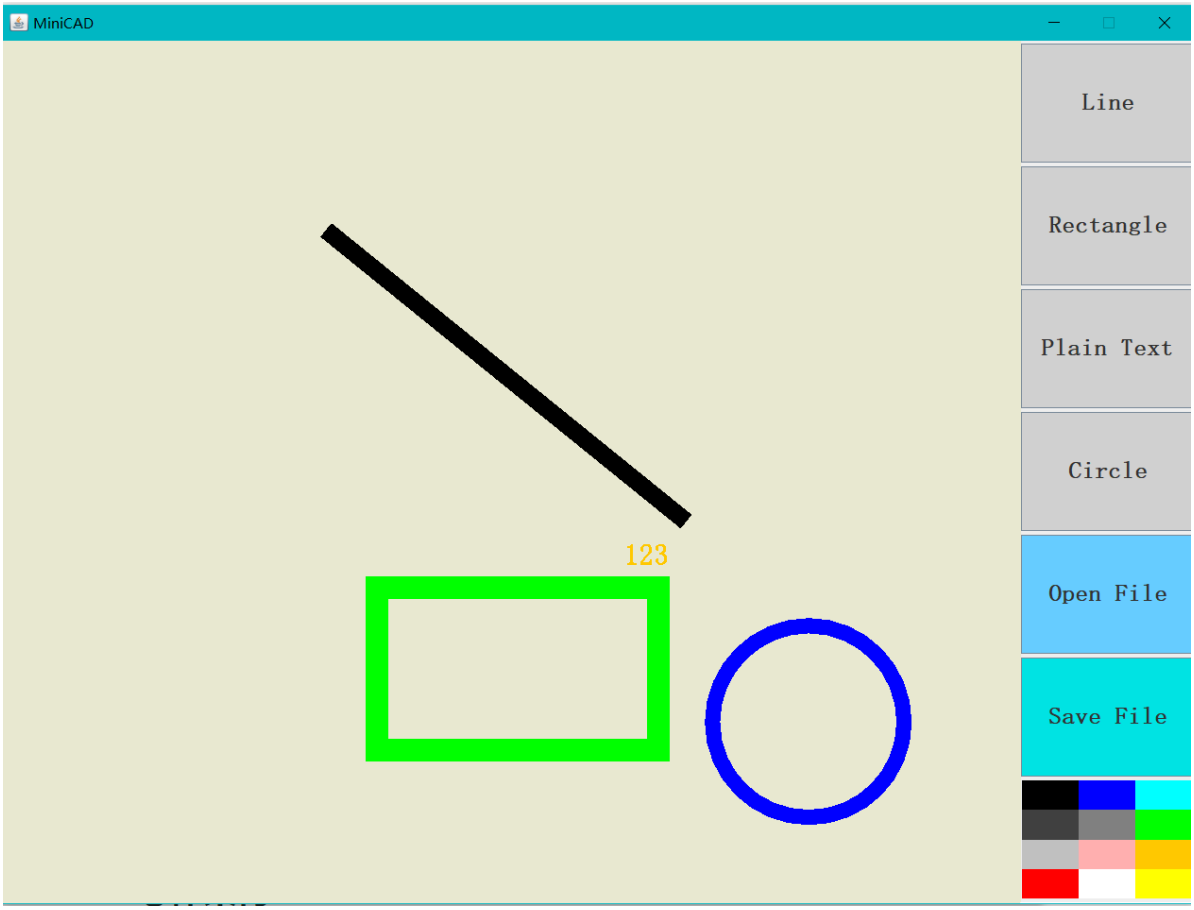
绘制图形

用户能够正常绘制图形、进行调色。注：Plain Text支持中文。



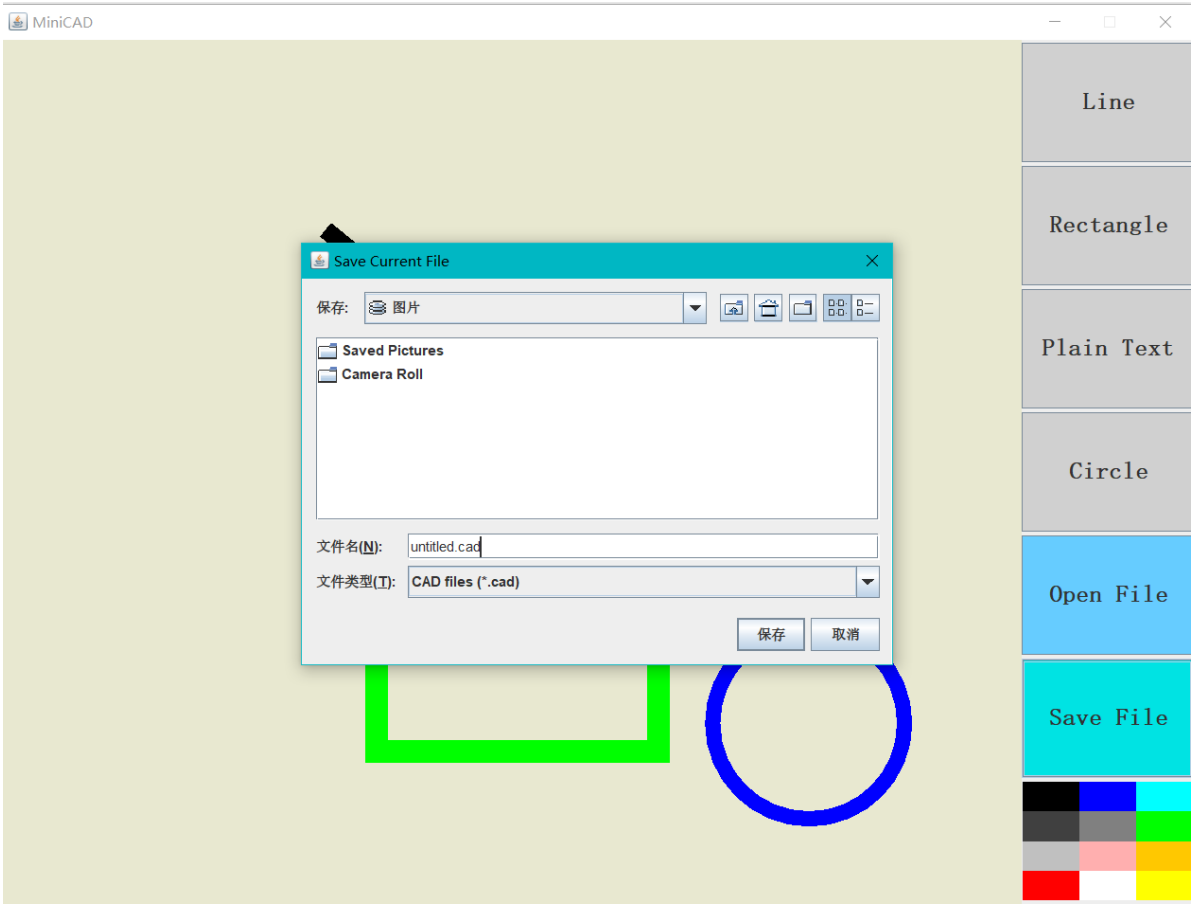
改变图形

鼠标拖拽可改变图形位置。通过`+`, `-`, `<`, `>`等各种按键，可改变图形的大小、粗细程度。按`R`可删除图形。



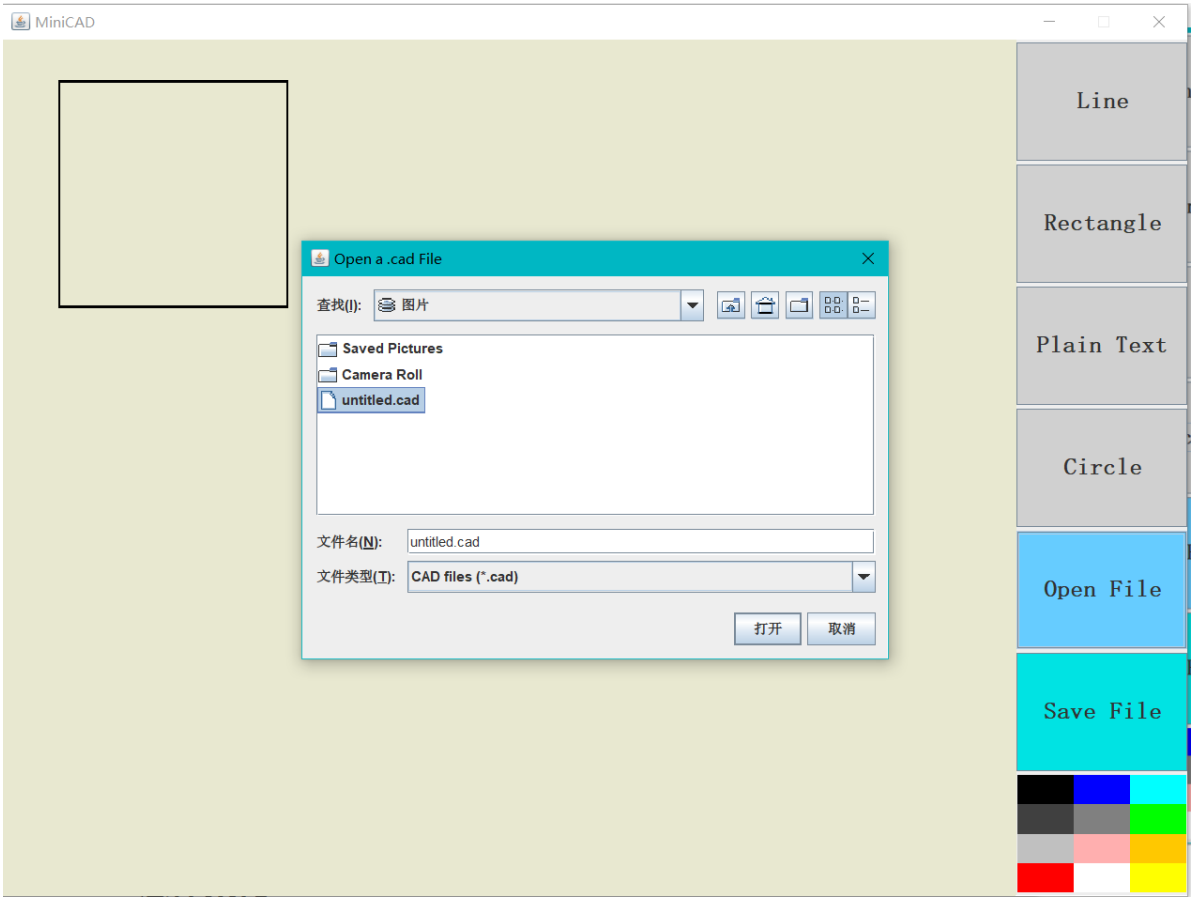
保存文件

将画布内容保存到文件。



打开文件

打开文件后，画布恢复为保存文件时的模样。退出程序后，“打开文件”仍能正常进行。


MiniCAD
Line
Rectangle
Plain Text
Circle
Open File
Save File
Open a .cad File
查找(I):
图片
Saved Pictures
Camera Roll
untitled.cad
文件名(N):
untitled.cad
文件类型(T):
CAD files (*.cad)
打开
取消