Appendix B: Code

Classes sorted by package

1.0. main

```
package main;

import java.awt.EventQueue;

public class Main {

    public static void main(String args[]) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    new AppFrame().setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

}
```

```
package main;

import java.awt.*;

import javax.swing.*;

import io.IOController;
import ui.*;

/**
 * The AppFrame constructs and manages the main frame
 */
public class AppFrame extends JFrame {

    private static final long serialVersionUID = 0;

    private static Menubar menubar;
    private static PickerPanel pickerPanel;
    private static CanvasPanel canvasPanel;
    private static DebugPanel debugPanel;
    private static IOController ioCon;

    public static final String appName = "Mind Mapper";

    public AppFrame() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setAppTitle(null);

        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        this.setBounds(50, 50, screenSize.width - 100, screenSize.height - 100);
        this.setLayout(new BorderLayout());

        initComponents();
    }

    private void initComponents() {
```

```
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            UIManager.put("OptionPane.background", Color.white);
            UIManager.put("Panel.background", Color.white);
        } catch (Exception e) {
            e.printStackTrace();
        }

        canvasPanel = new CanvasPanel(this);
        this.add(canvasPanel, BorderLayout.CENTER);

        pickerPanel = new PickerPanel(canvasPanel);
        this.add(pickerPanel, BorderLayout.WEST);

        debugPanel = new DebugPanel();
        this.add(debugPanel, BorderLayout.EAST);

        ioCon = new IOController(this, canvasPanel);

        menubar = new Menubar(this);
        this.setJMenuBar(menubar);
    }

    public void setAppTitle(String fileName) {
        if (fileName == null) this.setTitle(appName);
        else this.setTitle(fileName + " - " + appName);
    }

    // Getters
    public IOController getIOCon() {
        return ioCon;
    }
    public CanvasPanel getCanvasPanel() {
        return canvasPanel;
    }
    public PickerPanel getPickerPanel() {
        return pickerPanel;
    }
    public DebugPanel getDebugPanel() {
        return debugPanel;
    }

}
```

1.1. ui

```java
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.File;

import javax.swing.*;
import javax.swing.event.MenuEvent;
import javax.swing.event.MenuListener;
import javax.swing.filechooser.FileNameExtensionFilter;

import main.AppFrame;

/**
 * The Menubar manages the menu bar and handles actions, including the open, save and export
dialogs
 */
public class Menubar extends JMenuBar {

    private static final long serialVersionUID = 0;

    private JMenu fileMenu;
    private JMenuItem newFile;
    private JMenuItem open;
    private JMenuItem save;
    private JMenuItem saveAs;
    private JMenuItem export;
    private static final JFileChooser fileChooser = new
JFileChooser(System.getProperty("user.home"));
    static {
        fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        fileChooser.setAcceptAllFileFilterUsed(false);
    }
    private static final FileNameExtensionFilter mindMapFilter = new
FileNameExtensionFilter("Mind Maps (*.json)", "json");
    private static final FileNameExtensionFilter jpgFilter = new FileNameExtensionFilter("JPEG
Image (*.jpg)", "jpg");
    private static final FileNameExtensionFilter pngFilter = new FileNameExtensionFilter("PNG
Image (*.png)", "png");
    private JMenuItem exit;

    private JMenu editMenu;

    private JMenu viewMenu;
    private JMenuItem toggleGrid;
    private JMenuItem zoomIn;
    private JMenuItem zoomOut;
    private JMenuItem zoomReset;
    private JMenuItem centerCanvas;

    private JMenu windowMenu;
    private JMenuItem togglePickerPanel;
    private JMenuItem toggleDebugPanel;

    private JMenu helpMenu;
    private JMenuItem viewQuickStartGuide;
```

```
    private AppFrame appFrame;

    public Menubar(AppFrame appFrame) {
        this.appFrame = appFrame;
        initFileMenu();
        initEditMenu();
        initViewMenu();
        initWindowMenu();
        initHelpMenu();
    }

    private void initFileMenu() {
        fileMenu = new JMenu("File");
        this.add(fileMenu);

        newFile = new JMenuItem("New");
        newFile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
KeyEvent.CTRL_DOWN_MASK));
        newFile.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                appFrame.getCanvasPanel().reset();
            }
        });
        fileMenu.add(newFile);

        open = new JMenuItem("Open ...");
        open.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, KeyEvent.CTRL_DOWN_MASK));
        open.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                fileChooser.setDialogTitle("Open Mind Map");
                fileChooser.resetChoosableFileFilters();
                fileChooser.addChoosableFileFilter(mindMapFilter);

                // Change FileFilter selection label
                UIManager.put("FileChooser.filesOfTypeLabelText", "File Format:");
                SwingUtilities.updateComponentTreeUI(fileChooser);

                if (fileChooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
                    appFrame.getIOCon().handleOpen(fileChooser.getSelectedFile());
                }
            }
        });
        fileMenu.add(open);

        save = new JMenuItem("Save");
        save.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, KeyEvent.CTRL_DOWN_MASK));
        save.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (appFrame.getIOCon().getCurrentFile() != null) {
                    appFrame.getIOCon().handleSave(appFrame.getIOCon().getCurrentFile());
                } else
                    saveAs.doClick();
            }
        });
        fileMenu.add(save);

        saveAs = new JMenuItem("Save As ...");
        saveAs.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S, KeyEvent.CTRL_DOWN_MASK |
KeyEvent.SHIFT_DOWN_MASK));
        saveAs.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
```

```
            fileChooser.setDialogTitle("Save Mind Map");
            fileChooser.resetChoosableFileFilters();
            fileChooser.addChoosableFileFilter(mindMapFilter);
            fileChooser.setSelectedFile(new File("Untitled.json"));

            // Change FileFilter selection label
            UIManager.put("FileChooser.filesOfTypeLabelText", "File Format:");
            SwingUtilities.updateComponentTreeUI(fileChooser);

            if (fileChooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
                // Append ".json" extension if missing
                File file = fileChooser.getSelectedFile();
                if (! file.getName().toLowerCase().endsWith(".json"))
                    file = new File(file.getParentFile(), file.getName() + ".json");
                appFrame.getIOCon().handleSave(file);
            }
        }
    });
    fileMenu.add(saveAs);

    fileMenu.addSeparator();

    export = new JMenuItem("Export As ...");
    export.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            fileChooser.setDialogTitle("Export Mind Map");
            fileChooser.resetChoosableFileFilters();
            fileChooser.addChoosableFileFilter(jpgFilter);
            fileChooser.addChoosableFileFilter(pngFilter);
            fileChooser.setSelectedFile(new File("*.*"));

            // Change FileFilter selection label
            UIManager.put("FileChooser.filesOfTypeLabelText", "Select Export Format:");
            SwingUtilities.updateComponentTreeUI(fileChooser);

            if (fileChooser.showDialog(null, "Export") == JFileChooser.APPROVE_OPTION) {
                if (fileChooser.getFileFilter() == jpgFilter) {
                    // Append ".jpg" extension if missing
                    File file = fileChooser.getSelectedFile();
                    if (! file.getName().toLowerCase().endsWith(".jpg"))
                        file = new File(file.getParentFile(), file.getName() + ".jpg");
                    int scale = showExportScalePopup();
                    if (scale != 0) appFrame.getIOCon().handleExport(file, "jpg", scale);
                } else if (fileChooser.getFileFilter() == pngFilter) {
                    // Append ".png" extension if missing
                    File file = fileChooser.getSelectedFile();
                    if (! file.getName().toLowerCase().endsWith(".png"))
                        file = new File(file.getParentFile(), file.getName() + ".png");
                    int scale = showExportScalePopup();
                    if (scale != 0) appFrame.getIOCon().handleExport(file, "png", scale);
                }
            }
        }
    });
    fileMenu.add(export);

    fileMenu.addSeparator();

    exit = new JMenuItem("Exit");
    exit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q, KeyEvent.CTRL_DOWN_MASK));
    exit.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent evt) {
                System.exit(0);
            }
        });
        fileMenu.add(exit);
    }

    private void initEditMenu() {
        editMenu = new JMenu("Edit");
        this.add(editMenu);

        editMenu.add(new ContextMenu(appFrame.getCanvasPanel()).getAddMenu());
    }

    private void initViewMenu() {
        viewMenu = new JMenu("View");
        viewMenu.addMenuListener(new MenuListener() {
            public void menuSelected(MenuEvent evt) {
                if (appFrame.getCanvasPanel().getViewport().isGridVisible())
toggleGrid.setText("Hide Grid");
                else toggleGrid.setText("Show Grid");
            }
            public void menuDeselected(MenuEvent evt) {
            }
            public void menuCanceled(MenuEvent evt) {
            }
        });
        this.add(viewMenu);

        toggleGrid = new JMenuItem("", KeyEvent.VK_F4);
        toggleGrid.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F4, 0));
        toggleGrid.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                if (appFrame.getCanvasPanel().getViewport().isGridVisible())
appFrame.getCanvasPanel().getViewport().setGridVisible(false);
                else appFrame.getCanvasPanel().getViewport().setGridVisible(true);
            }
        });
        viewMenu.add(toggleGrid);

        viewMenu.addSeparator();

        zoomIn = new JMenuItem("Zoom In");
        zoomIn.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_EQUALS,
KeyEvent.CTRL_DOWN_MASK | KeyEvent.SHIFT_DOWN_MASK));
        zoomIn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                appFrame.getCanvasPanel().getViewport().zoomIn();
            }
        });
        viewMenu.add(zoomIn);

        zoomOut = new JMenuItem("Zoom Out");
        zoomOut.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_MINUS,
KeyEvent.CTRL_DOWN_MASK));
        zoomOut.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                appFrame.getCanvasPanel().getViewport().zoomOut();
            }
        });
        viewMenu.add(zoomOut);
```

```
        zoomReset = new JMenuItem("Zoom 100%");
        zoomReset.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_0, KeyEvent.CTRL_MASK));
        zoomReset.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                appFrame.getCanvasPanel().getViewport().reset();
            }
        });
        viewMenu.add(zoomReset);

        centerCanvas = new JMenuItem("Center Canvas");
        centerCanvas.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_PERIOD,
KeyEvent.CTRL_MASK));
        centerCanvas.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                appFrame.getCanvasPanel().getViewport().centerView();
            }
        });
        viewMenu.add(centerCanvas);
    }

    private void initWindowMenu() {
        windowMenu = new JMenu("Window");
        windowMenu.addMenuListener(new MenuListener() {
            public void menuSelected(MenuEvent evt) {
                if (appFrame.getPickerPanel().isVisible()) togglePickerPanel.setText("Hide
Picker Panel");
                else togglePickerPanel.setText("Show Picker Panel");

                if (appFrame.getDebugPanel().isVisible()) toggleDebugPanel.setText("Hide Debug
Panel");
                else toggleDebugPanel.setText("Show Debug Panel");
            }
            public void menuDeselected(MenuEvent evt) {
            }
            public void menuCanceled(MenuEvent evt) {
            }
        });
        this.add(windowMenu);

        togglePickerPanel = new JMenuItem("", KeyEvent.VK_F12);
        togglePickerPanel.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F12, 0));
        togglePickerPanel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                // Dynamically change offset of viewport
                if (appFrame.getPickerPanel().isVisible()) {
                    appFrame.getPickerPanel().setVisible(false);
                    appFrame.getCanvasPanel().getViewport().xOffset +=
appFrame.getPickerPanel().getWidth();
                } else {
                    appFrame.getPickerPanel().setVisible(true);
                    appFrame.getCanvasPanel().getViewport().xOffset -=
appFrame.getPickerPanel().getWidth();
                }
                appFrame.getCanvasPanel().repaint();
            }
        });
        windowMenu.add(togglePickerPanel);

        toggleDebugPanel = new JMenuItem("", KeyEvent.VK_F3);
        toggleDebugPanel.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F3, 0));
```

```
        toggleDebugPanel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                if (appFrame.getDebugPanel().isVisible()) {
                    appFrame.getDebugPanel().setVisible(false);
                    appFrame.getCanvasPanel().getViewport().debugLabelsUpdater.stop();
                } else {
                    appFrame.getDebugPanel().setVisible(true);
                    appFrame.getCanvasPanel().getViewport().debugLabelsUpdater.start();
                }
            }
        });
        windowMenu.add(toggleDebugPanel);
    }

    private void initHelpMenu() {
        helpMenu = new JMenu("Help");
        this.add(helpMenu);

        viewQuickStartGuide = new JMenuItem("Quick Start Guide");
        viewQuickStartGuide.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(appFrame, new QuickStartGuide(), "Mind Mapper",
JOptionPane.INFORMATION_MESSAGE, null);
            }
        });
        helpMenu.add(viewQuickStartGuide);
    }

    /**
     * Allow the user to select the export quality via a dialog
     * @return int chosen scale factor or 0 if cancelled
     */
    private int showExportScalePopup() {
        // Add choosable export qualities and calculate their final dimensions
        String exportQualities[] = new String[5];
        for (int i = 1; i <= exportQualities.length; i++)
            exportQualities[i-1] = i +"x (" + appFrame.getCanvasPanel().getWidth()*i + "x" +
appFrame.getCanvasPanel().getHeight()*i + ")";
        // Pop up a dialog
        String output = (String) JOptionPane.showInputDialog(fileChooser, "Image size:",
"Export Options",
                                    JOptionPane.PLAIN_MESSAGE, null, exportQualities, null);
        if (output != null) return Integer.parseInt(Character.toString(output.charAt(0)));
        else                return 0;
    }

}
```

```
package ui;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import main.AppFrame;
import shapes.*;

/**
 * The CanvasPanel controls all mind map elements and listeners
```

```
 */
public class CanvasPanel extends JPanel {

    private static final long serialVersionUID = 0;

    private ContextMenu contextMenu;
    public boolean isContextTrigger = false;
    public MouseEvent contextTriggerEvent;

    private Viewport viewport;
    private MapController mapCon;

    private AppFrame appFrame;

    public CanvasPanel(AppFrame appFrame) {
        this.appFrame = appFrame;
        contextMenu = new ContextMenu(this);
        reset();
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        viewport.drawAll(g);        // Draw all shapes, text, and lines
    }

    // Handle displaying context menu
    public void popup(MouseEvent e) {
        if (mapCon.getShapesUnderCursor(e.getPoint()).size() > 0) {
            contextMenu.updateMenuValues(mapCon.getSelectedShape());
            contextMenu.getEditMenu().setEnabled(true);
            contextMenu.getOrderMenu().setEnabled(true);
            contextMenu.getConnectionsMenu().setEnabled(true);
        } else {
            contextMenu.getEditMenu().setEnabled(false);
            contextMenu.getOrderMenu().setEnabled(false);
            contextMenu.getConnectionsMenu().setEnabled(false);
        }
        contextMenu.show(e.getComponent(), e.getX(), e.getY());
    }

    public void reset() {
        viewport = new Viewport(this);
        mapCon = new MapController(this, viewport);
        MapListener mapListener = new MapListener(this, viewport);
        this.addMouseListener(mapListener);
        this.addMouseMotionListener(mapListener);
        this.addMouseWheelListener(mapListener);
        this.setBackground(Color.white);
        appFrame.setAppTitle(null);
        repaint();
    }

    // Getters
    public Viewport getViewport() {
        return viewport;
    }
    public MapController getMapController() {
        return mapCon;
    }
    public ContextMenu getContextMenu() {
```

```
        return contextMenu;
    }

}
```

```
package ui;

import java.awt.Component;
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;

import javax.swing.JTextField;

import shapes.MapController;

/**
 * The MapListener handles clicking, dragging and scrolling in the CanvasPanel
 */
public class MapListener implements MouseListener, MouseMotionListener, MouseWheelListener {

    private CanvasPanel canvasPanel;
    private Viewport viewport;
    private MapController mapCon;

    public MapListener(CanvasPanel canvasPanel, Viewport viewport) {
        this.canvasPanel = canvasPanel;
        this.viewport = viewport;
        this.mapCon = canvasPanel.getMapController();
    }

    // Mouse activity listeners
    public void mousePressed(MouseEvent evt) {
        viewport.panStartPoint = evt.getLocationOnScreen();
        viewport.setMouseReleased(false);

        selectShapeUnderCursor(evt);
        triggerContext(evt);
    }
    public void mouseReleased(MouseEvent evt) {
        viewport.setMouseReleased(true);
        canvasPanel.repaint();        // Bypass FPS limiter and force repaint to lock in
position

        selectShapeUnderCursor(evt);
        triggerContext(evt);
    }
    public void mouseDragged(MouseEvent evt) {
        if (mapCon.getSelectedShape() == null && !canvasPanel.isContextTrigger)
{            // Pan the canvas
            viewport.pan(evt.getLocationOnScreen());
        } else if (mapCon.getSelectedShape() != null && !canvasPanel.isContextTrigger) {    //
Drag the selected shape
            Point curPoint = evt.getLocationOnScreen();
            if (curPoint.x  != mapCon.dragStartPoint.x || curPoint.y !=
mapCon.dragStartPoint.y) {
```

```
                mapCon.getSelectedShape().setNewCoordinates(                // Update
coordinates
                        mapCon.getSelectedShape().getX() + (int)((curPoint.x -
mapCon.dragStartPoint.x)/viewport.zoomFactor),
                        mapCon.getSelectedShape().getY() + (int)((curPoint.y -
mapCon.dragStartPoint.y)/viewport.zoomFactor));
                mapCon.dragStartPoint = evt.getLocationOnScreen();            // Update drag
diff reference
            }
            viewport.handleRepaint();
        }
    }
    public void mouseWheelMoved(MouseWheelEvent evt) {
        if (!canvasPanel.isContextTrigger) {
            if (evt.getWheelRotation() < 0) {                // Mouse wheel rolls forward
                viewport.zoomIn();
            } else if (evt.getWheelRotation() > 0) {        // Mouse wheel rolls backwards
                viewport.zoomOut();
            }
        }
    }
    public void mouseClicked(MouseEvent evt) {
        // Handle double-click
        if (evt.getClickCount() == 2 && mapCon.getSelectedShape() != null) {
            canvasPanel.add(mapCon.getSelectedShape().getTextField());
            mapCon.setEditingShape(mapCon.getSelectedShape());
            mapCon.getSelectedShape().getTextField().requestFocusInWindow();
            mapCon.getSelectedShape().getTextField().selectAll();
        }
    }
    public void mouseMoved(MouseEvent evt) {
    }
    public void mouseEntered(MouseEvent evt) {
    }
    public void mouseExited(MouseEvent evt) {
    }

    private void selectShapeUnderCursor(MouseEvent evt) {
        // Select the shape that is clicked on
        if (mapCon.getShapesUnderCursor(evt.getPoint()).size() > 0) {
            mapCon.dragStartPoint = evt.getLocationOnScreen();            // Update drag diff
reference
            if (mapCon.shapeSelectionIndex >
mapCon.getShapesUnderCursor(evt.getPoint()).size() - 1)
                mapCon.shapeSelectionIndex = 0;        // Prevent index overflow by restarting
cycle

mapCon.setSelectedShape(mapCon.getShapesUnderCursor(evt.getPoint()).get(mapCon.shapeSelectionI
ndex));
            mapCon.shapeSelectionIndex++;                // Increment index to select
overlapped shapes
        } else {    // Remove the selection
            for (Component com : canvasPanel.getComponents())
                if (com instanceof JTextField) canvasPanel.remove(com);        // Remove all
text fields
            mapCon.setEditingShape(null);
            mapCon.setSelectedShape(null);
        }
    }

    private void triggerContext(MouseEvent evt) {
```

```
        if (evt.isPopupTrigger()) {
            canvasPanel.isContextTrigger = true;
            canvasPanel.contextTriggerEvent = evt;
            canvasPanel.popup(evt);
        } else {
            canvasPanel.isContextTrigger = false;
        }
    }
}
```

```
package ui;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.MouseInfo;
import java.awt.Point;
import java.awt.RenderingHints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.AffineTransform;

import javax.swing.Timer;

import com.google.gson.JsonObject;

import shapes.Grid;
import shapes.MapLine;
import shapes.MapShape;

/**
 * The Viewport manages and handles drawing mind map elements
 */
public class Viewport {

    private static final int MAX_FPS = 60;
    private long lastFrameTime = 0;
    // Zoom and pan variables
    private boolean zooming;
    public double zoomFactor = 1, prevZoomFactor = 1;
    private boolean released;
    public int xOffset = 0, yOffset = 0;
    protected Point panStartPoint;
    protected int panXDiff, panYDiff;

    private boolean showGrid = true;
    private boolean initGrid = false;
    private int gridOffsetX, gridOffsetY;
    private Grid grid;

    protected Timer debugLabelsUpdater;

    private CanvasPanel canvasPanel;

    public Viewport(CanvasPanel canvasPanel) {
        this.canvasPanel = canvasPanel;
        grid = new Grid(10000);
        initTimers();
    }
```

```
    /**
     * Manages the scale and offset of mind map elements and their drawing process
     * @param g
     */
    public void drawAll(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        AffineTransform at = new AffineTransform();
        if (zooming) {                // Handle zooming relative to cursor
            double xRel = MouseInfo.getPointerInfo().getLocation().getX() -
canvasPanel.getX();
            double yRel = MouseInfo.getPointerInfo().getLocation().getY() -
canvasPanel.getY();
            double zoomDiv = zoomFactor / prevZoomFactor;
            xOffset = (int)((zoomDiv) * (xOffset) + (1 - zoomDiv) * xRel);
            yOffset = (int)((zoomDiv) * (yOffset) + (1 - zoomDiv) * yRel);

            prevZoomFactor = zoomFactor;
            zooming = false;
        }                             // If released, reset pan diff
        if (released) {
            xOffset += panXDiff;
            yOffset += panYDiff;
            panXDiff = 0;
            panYDiff = 0;
        }
        at.translate(xOffset + panXDiff, yOffset + panYDiff);
        at.scale(zoomFactor, zoomFactor);
        g2d.transform(at);

        // Iterate and print all lines before shapes
        for (MapLine connection : canvasPanel.getMapController().getConnections()) {
            connection.updateConnection();
            g2d.setColor(Color.black);
            g2d.setStroke(connection.getStroke());
            g2d.draw(connection.getLine());
        }
        for (MapShape mapShape : canvasPanel.getMapController().getShapes()) {
            // Fill shape background with white to hide lines within the shape
            g2d.setColor(Color.white);
            g2d.fill(mapShape.getShape());
            // Draw border around shape
            if (mapShape.isHighlighted) g2d.setColor(Color.cyan);
            else g2d.setColor(mapShape.getBorderColour());
            g2d.setStroke(mapShape.getBorderStroke());
            g2d.draw(mapShape.getShape());
            // Draw text
            drawShapeText(g, mapShape);
        }
        if (!initGrid) {                              // Calculate new offset for grid to be
centered
            gridOffsetX = canvasPanel.getWidth()/2;     // Only run once when canvasPanel is
drawn on-screen
            gridOffsetY = canvasPanel.getHeight()/2;
            initGrid = true;
        }
        if (showGrid) {
            Graphics2D gridG2d = (Graphics2D) g2d.create();
```

```
            gridG2d.translate(gridOffsetX, gridOffsetY);    // Grid is now offset by a static
amount
            grid.drawGrid(gridG2d);
        }
    }

    private void drawShapeText(Graphics g, MapShape mapShape) {
        /**
         * Draw text from textfield only, if not being edited, or position the textfield
correctly if being edited
         */
        if (! mapShape.equals(canvasPanel.getMapController().getEditingShape())) {
            // Offset the location of text fields to center of shape
            mapShape.getTextField().setBounds(mapShape.getX() +
mapShape.getShape().getBounds().width/2 - 100 + xOffset,
                                              mapShape.getY() +
mapShape.getShape().getBounds().height/2 - 50 + yOffset,
                                              200, 100);
            Graphics2D textGraphics = (Graphics2D)
g.create(mapShape.getTextField().getBounds().x - xOffset,
                                              mapShape.getTextField().getBounds().y -
yOffset,
                                              mapShape.getTextField().getBounds().width,
                                              mapShape.getTextField().getBounds().height);
            mapShape.getTextField().paint(textGraphics);
        } else {
            mapShape.getTextField().setBounds(mapShape.getX() +
mapShape.getShape().getBounds().width/2 - 100,
                                              mapShape.getY() +
mapShape.getShape().getBounds().height/2 - 50,
                                              200, 100);
        }
    }

    protected void handleRepaint() {        // A handler to limit framerate and CPU usage
        // Calculate frame time and only repaint at the specified framerate
        if (System.currentTimeMillis() - lastFrameTime >= (1000/MAX_FPS)) {
            canvasPanel.repaint();
            lastFrameTime = System.currentTimeMillis();
        }
    }

    // Viewport controls
    public void pan(Point curPoint) {
        panXDiff = curPoint.x - panStartPoint.x;
        panYDiff = curPoint.y - panStartPoint.y;
        handleRepaint();
    }
    public void zoomIn() {
        zooming = true;
        zoomFactor *= 1.1;
        handleRepaint();
    }
    public void zoomOut() {
        zooming = true;
        zoomFactor /= 1.1;
        handleRepaint();
    }
    public void centerView() {
        // Calculate average center
        int numShapes = canvasPanel.getMapController().getShapes().size();
```

```
            int totalX = 0, totalY = 0;
            for (MapShape shape : canvasPanel.getMapController().getShapes()) {
                totalX += shape.getX() + shape.getShape().getBounds().getWidth()/2;
                totalY += shape.getY() + shape.getShape().getBounds().getHeight()/2;
            }
            xOffset = (int)-((totalX/numShapes - canvasPanel.getWidth()/2)/zoomFactor);
            yOffset = (int)-((totalY/numShapes - canvasPanel.getHeight()/2)/zoomFactor);

            canvasPanel.repaint();
        }
        public void reset() {
            xOffset = 0; yOffset = 0;
            zoomFactor = 1.0; prevZoomFactor = 1.0;
            released = true;
            zooming = true;
            canvasPanel.repaint();
        }

        private void initTimers() {
            debugLabelsUpdater = new Timer(150, new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    if (System.currentTimeMillis() - lastFrameTime != 0)
                        DebugPanel.fpsLbl.setText("FPS: " + 1000/(System.currentTimeMillis()-
lastFrameTime));
                    DebugPanel.zoomLbl.setText("Zoom: " +
Double.toString(Math.round(zoomFactor*100)/100.0));
                    DebugPanel.xOffsetLbl.setText("xOffset: " + (xOffset+panXDiff));
                    DebugPanel.yOffsetLbl.setText("yOffset: " + (yOffset+panYDiff));
                }
            });
            debugLabelsUpdater.start();
        }


        // Getters and setters
        public void setMouseReleased(boolean state) {
            released = state;
        }
        public boolean isGridVisible() {
            return showGrid;
        }
        public void setGridVisible(boolean state) {
            showGrid = state;
            canvasPanel.repaint();
        }
        public JsonObject getViewportData() {
            JsonObject viewportData = new JsonObject();
            viewportData.addProperty("Zoom", zoomFactor);
            viewportData.addProperty("xOffset", xOffset);
            viewportData.addProperty("yOffset", yOffset);
            return viewportData;
        }
        public void setViewportData(JsonObject viewportData) {
            reset();
            zoomFactor = viewportData.get("Zoom").getAsDouble();
            prevZoomFactor = canvasPanel.getViewport().zoomFactor;
            xOffset = viewportData.get("xOffset").getAsInt();
            yOffset = viewportData.get("yOffset").getAsInt();
            released = true;
        }
```

```
}
```

```
package ui;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

import shapes.MapLine;
import shapes.MapShape;

import java.awt.Color;
import java.awt.Font;

/**
 * The ContextMenu manages and handles actions in the right-click menu
 */
public class ContextMenu extends JPopupMenu {

    private static final long serialVersionUID = 0;

    private JMenu addMenu;
    private JMenuItem addEllipse;
    private JMenuItem addRectangle;
    private JMenuItem addTriangle;

    private JMenu editMenu;
    private JMenuItem removeElement;

    private JMenu changeBorderWidthMenu;
    private JSlider changeBorderWidthSlider;
    private JMenu changeBorderColourMenu;

    private JMenu changeFontColourMenu;
    private JMenu changeFontMenu;
    private JMenu changeFontStyleMenu;
    private JMenu changeFontSizeMenu;
    private JTextField changeFontSizeField;

    // Lookup tables for values and their names
    private static final HashMap<String,Color> colours = new HashMap<String,Color>();
    static {
        colours.put("Black", Color.black);
        colours.put("Red", Color.red);
        colours.put("Green", Color.green);
        colours.put("Blue", Color.blue);
        colours.put("Yellow", Color.yellow);
        colours.put("Orange", Color.orange);
        colours.put("Magenta", Color.magenta);
        colours.put("Pink", Color.pink);
```

```
}
private static final HashMap<String,String> fonts = new HashMap<String,String>();
static {
    fonts.put("Times New Roman", "Serif");
    fonts.put("Helvetica", "SansSerif");
    fonts.put("Courier", "Monospaced");
}
private static final HashMap<String,Integer> fontStyles = new HashMap<String,Integer>();
static {
    fontStyles.put("Plain", Font.PLAIN);
    fontStyles.put("Bold", Font.BOLD);
    fontStyles.put("Italic", Font.ITALIC);
}

private JMenu orderMenu;
private JMenuItem bringToFront;
private JMenuItem bringForward;
private JMenuItem sendToBack;
private JMenuItem sendBackward;

private JMenu connectionsMenu;
private JMenuItem removeAllConnections;
private JMenuItem setShapeInConnection;

private CanvasPanel canvasPanel;

public ContextMenu(CanvasPanel canvasPanel) {
    this.canvasPanel = canvasPanel;
    initAddMenu();
    initEditMenu();
    initOrderMenu();
    initConnectionsMenu();
}

private void initAddMenu() {
    addMenu = new JMenu("Add ...");
    this.add(addMenu);

    addEllipse = new JMenuItem("Ellipse shape");
    addEllipse.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            canvasPanel.getMapController().addShape("shapes.EllipseShape");
        }
    });
    addMenu.add(addEllipse);

    addRectangle = new JMenuItem("Rectangle shape");
    addRectangle.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            canvasPanel.getMapController().addShape("shapes.RectangleShape");
        }
    });
    addMenu.add(addRectangle);

    addTriangle = new JMenuItem("Triangle shape");
    addTriangle.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            canvasPanel.getMapController().addShape("shapes.TriangleShape");
        }
    });
    addMenu.add(addTriangle);
```

```
    }

    private void initEditMenu() {
        editMenu = new JMenu("Edit");
        this.add(editMenu);

        removeElement = new JMenuItem("Remove this");
        editMenu.add(removeElement);
        removeElement.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                canvasPanel.getMapController().removeSelectedShape();
            }
        });

        editMenu.addSeparator();

        changeBorderWidthMenu = new JMenu("Border width");
        editMenu.add(changeBorderWidthMenu);
        // Create a slider to set border width
        changeBorderWidthSlider = new JSlider(JSlider.HORIZONTAL, 1, 10, 3);
        changeBorderWidthSlider.setMajorTickSpacing(1);
        changeBorderWidthSlider.setSnapToTicks(true);
        changeBorderWidthSlider.setPaintTicks(true);
        changeBorderWidthSlider.setPaintLabels(true);
        changeBorderWidthSlider.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {

canvasPanel.getMapController().changeBorderWidth(changeBorderWidthSlider.getValue());
            }
        });
        changeBorderWidthMenu.add(changeBorderWidthSlider);

        changeBorderColourMenu = new JMenu("Border colour");
        editMenu.add(changeBorderColourMenu);
        // Iterate through available colours and create a new menu item for each
        for (String colourName : colours.keySet()) {
            JMenuItem selectBorderColour = new JMenuItem(colourName);
            selectBorderColour.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {

canvasPanel.getMapController().changeBorderColour(colours.get(colourName));
                    canvasPanel.getMapController().setSelectedShape(null);
                }
            });
            changeBorderColourMenu.add(selectBorderColour);
        }

        editMenu.addSeparator();

        changeFontMenu = new JMenu("Text font");
        editMenu.add(changeFontMenu);
        // Iterate through available fonts and create a new menu item for each
        for (String fontName : fonts.keySet()) {
            JMenuItem selectFont = new JMenuItem(fontName);
            selectFont.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    canvasPanel.getMapController().changeFont(fontName);
                    canvasPanel.getMapController().setSelectedShape(null);
                }
            });
            changeFontMenu.add(selectFont);
```

```
        }

        changeFontStyleMenu = new JMenu("Text font style");
        editMenu.add(changeFontStyleMenu);
        // Iterate through available font styles and create a new menu item for each
        for (String fontStyle : fontStyles.keySet()) {
            JMenuItem selectFontStyle = new JMenuItem(fontStyle);
            selectFontStyle.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    canvasPanel.getMapController().changeFontStyle(fontStyles.get(fontStyle));
                    canvasPanel.getMapController().setSelectedShape(null);
                }
            });
            changeFontStyleMenu.add(selectFontStyle);
        }

        changeFontSizeMenu = new JMenu("Text font size");
        editMenu.add(changeFontSizeMenu);
        // Create a text field for custom font sizes
        changeFontSizeField = new JTextField();
        changeFontSizeField.setColumns(3);
        changeFontSizeField.getDocument().addDocumentListener(new DocumentListener() {
            public void insertUpdate(DocumentEvent e) {
                changeFontSize();
            }
            public void removeUpdate(DocumentEvent e) {
                changeFontSize();
            }
            public void changedUpdate(DocumentEvent e) {
            }
            private void changeFontSize() {
                // Parse as int if the text field only contains numbers
                if (changeFontSizeField.getText().matches("^\\d+$")) {
                    canvasPanel.getMapController()
                                    .changeFontSize(Integer.parseInt(changeFontSizeField.getTe
xt()));
                }
            }
        });
        changeFontSizeMenu.add(changeFontSizeField);

        changeFontColourMenu = new JMenu("Text font colour");
        editMenu.add(changeFontColourMenu);
        // Iterate through available colours and create a new menu item for each
        for (String colourName : colours.keySet()) {
            JMenuItem selectFontColour = new JMenuItem(colourName);
            selectFontColour.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    canvasPanel.getMapController().changeFontColour(colours.get(colourName));
                    canvasPanel.getMapController().setSelectedShape(null);
                }
            });
            changeFontColourMenu.add(selectFontColour);
        }
    }

    private void initOrderMenu() {
        orderMenu = new JMenu("Order");
        this.add(orderMenu);

        bringToFront = new JMenuItem("Bring to Front");
```

```
        bringToFront.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                canvasPanel.getMapController().bringSelectedShapeToFront();
            }
        });
        orderMenu.add(bringToFront);

        bringForward = new JMenuItem("Bring Forward");
        bringForward.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                canvasPanel.getMapController().bringSelectedShapeForwards();
            }
        });
        orderMenu.add(bringForward);

        sendToBack = new JMenuItem("Send to Back");
        sendToBack.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                canvasPanel.getMapController().sendSelectedShapeToBack();
            }
        });
        orderMenu.add(sendToBack);

        sendBackward = new JMenuItem("Send Backward");
        sendBackward.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                canvasPanel.getMapController().sendSelectedShapeBackward();
            }
        });
        orderMenu.add(sendBackward);
    }

    private void initConnectionsMenu() {
        connectionsMenu = new JMenu("Manage connections");
        this.add(connectionsMenu);

        removeAllConnections = new JMenuItem("Remove all connections");
        removeAllConnections.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                canvasPanel.getMapController().removeConnectionsFromSelectedShape();
            }
        });
        connectionsMenu.add(removeAllConnections);

        setShapeInConnection = new JMenuItem("Set as origin");
        setShapeInConnection.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (setShapeInConnection.getText().equals("Set as origin")) {
                    canvasPanel.getMapController().setSelectedShapeAsOrigin();
                } else {
                    canvasPanel.getMapController().setSelectedShapeAsTermination();
                }
            }
        });
        connectionsMenu.add(setShapeInConnection);
    }

    public void updateMenuValues(MapShape selectedShape) {
        if (selectedShape != null) {
            // Update border width and font size values for the selected shape
```

```
changeBorderWidthSlider.setValue((int)selectedShape.getBorderStroke().getLineWidth());

changeFontSizeField.setText(Integer.toString(selectedShape.getTextField().getFont().getSize())
);

            // Hide "Remove all connections" button if there are no connections
            boolean hasConnections = false;
            for (MapLine connection : canvasPanel.getMapController().getConnections()) {
                if (connection.getOrigin() == selectedShape || connection.getTermination() ==
selectedShape) {
                    hasConnections = true;
                    break;
                }
            }
            removeAllConnections.setEnabled(hasConnections ? true : false);

            if (canvasPanel.getMapController().getConnectionOrigin() == null) {
                // Allow selecting as origin
                setShapeInConnection.setEnabled(true);
                setShapeInConnection.setForeground(Color.black);
                setShapeInConnection.setText("Set as origin");
            } else {
                // Allow selecting as termination
                if (canvasPanel.getMapController().getSelectedShape().equals(
                            canvasPanel.getMapController().getConnectionOrigin())) {
                    setShapeInConnection.setEnabled(false);
                    setShapeInConnection.setForeground(Color.red);
                    setShapeInConnection.setText("Cannot set selected shape again");
                } else {
                    setShapeInConnection.setEnabled(true);
                    setShapeInConnection.setForeground(Color.black);
                    setShapeInConnection.setText("Set as termination");
                }
            }
        }
    }

    public JMenu getAddMenu() {
        return addMenu;
    }
    public JMenu getEditMenu() {
        return editMenu;
    }
    public JMenu getOrderMenu() {
        return orderMenu;
    }
    public JMenu getConnectionsMenu() {
        return connectionsMenu;
    }


}
```

```
package ui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
```

```java
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.List;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;

import shapes.*;

public class PickerPanel extends JPanel {

    private static final long serialVersionUID = 0;

    private static final int MAX_FPS = 60;
    private long lastFrameTime = 0;

    private static JLabel title;

    private static List<MapShape> shapes;

    public PickerPanel(CanvasPanel canvasPanel) {
        this.setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        this.setPreferredSize(new Dimension(200, 0));
        this.setBackground(Color.LIGHT_GRAY);
        this.setBorder(new EtchedBorder(EtchedBorder.RAISED));

        title = new JLabel("Add");
        title.setAlignmentX(JLabel.CENTER_ALIGNMENT);
        title.setBorder(new EmptyBorder(25, 0, 0, 0));
        title.setFont(new Font("Arial", Font.BOLD, 20));
        this.add(title);

        initShapes();

        // Mouse activity listeners
        this.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent evt) {
            }
            public void mouseReleased(MouseEvent evt) {
            }
            public void mouseClicked(MouseEvent evt) {
                // Handle double-click
                if (evt.getClickCount() == 2 && getShapeUnderCursor(evt) != null) {

canvasPanel.getMapController().addShape(getShapeUnderCursor(evt).getClass().getName());
                }
            }
        });
        this.addMouseMotionListener(new MouseAdapter() {
            public void mouseDragged(MouseEvent evt) {
                // if (shapeCon.getSelectedShape() != null)
{                                // Drag the selected shape
                //     Point curPoint = evt.getLocationOnScreen();
                //     if (curPoint.x  != shapeCon.dragStartPoint.x || curPoint.y !=
shapeCon.dragStartPoint.y) {
                //         shapeCon.getSelectedShape().setNewCoordinates(
                //                 shapeCon.getSelectedShape().getX() + (int)((curPoint.x -
shapeCon.dragStartPoint.x)/viewport.zoomFactor),
```

```
                //                 shapeCon.getSelectedShape().getY() + (int)((curPoint.y -
shapeCon.dragStartPoint.y)/viewport.zoomFactor));
                //         shapeCon.dragStartPoint = evt.getLocationOnScreen();            //
Update drag diff reference
                //     }
                //     viewport.handleRepaint();
                // }
            }
            public void mouseMoved(MouseEvent evt) {
                for (MapShape shape : shapes) {
                    if (shape.getShape().getBounds().contains(evt.getPoint()))
shape.isHighlighted = true;
                    else shape.isHighlighted = false;
                }
                handleRepaint();
            }
        });
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        for (MapShape mapShape : shapes) {
            // Fill shape background with white
            g2d.setColor(Color.white);
            g2d.fill(mapShape.getShape());
            // Draw border around shape
            if (mapShape.isHighlighted) g2d.setColor(Color.cyan);
            else g2d.setColor(mapShape.getBorderColour());
            g2d.setStroke(mapShape.getBorderStroke());
            g2d.draw(mapShape.getShape());
        }
    }

    private void handleRepaint() {        // A handler to limit framerate and CPU usage
        // Calculate frame time and only repaint at the specified framerate
        if (System.currentTimeMillis() - lastFrameTime >= (1000/MAX_FPS)) {
            repaint();
            lastFrameTime = System.currentTimeMillis();
        }
    }

    private void initShapes() {
        shapes = new ArrayList<MapShape>();

        shapes.add(new RectangleShape(40, 100, 120, 60));
        shapes.add(new EllipseShape(40, 200, 120, 60));
        shapes.add(new TriangleShape(40, 300, 120, 60));
    }

    private MapShape getShapeUnderCursor(MouseEvent evt) {
        for (MapShape shape : shapes)
            if (shape.getShape().getBounds().contains(evt.getPoint())) return shape;
        return null;
    }
}
```

```java
package ui;

import java.awt.Color;
import java.awt.Dimension;

import javax.swing.BoxLayout;
import javax.swing.border.Border;
import javax.swing.*;

public class DebugPanel extends JPanel {

    private static final long serialVersionUID = 0;

    protected static JLabel fpsLbl;
    protected static JLabel zoomLbl;
    protected static JLabel xOffsetLbl;
    protected static JLabel yOffsetLbl;

    public DebugPanel() {
        this.setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        this.setPreferredSize(new Dimension(100, 0));
        this.setBackground(Color.LIGHT_GRAY);
        Border compound = BorderFactory.createEtchedBorder();
        compound = BorderFactory.createCompoundBorder(compound,
BorderFactory.createEmptyBorder(5, 5, 5, 5));
        this.setBorder(compound);
        this.setVisible(false);

        fpsLbl = new JLabel();
        this.add(fpsLbl);

        zoomLbl = new JLabel();
        this.add(zoomLbl);

        xOffsetLbl = new JLabel();
        this.add(xOffsetLbl);
        yOffsetLbl = new JLabel();
        this.add(yOffsetLbl);
    }

}
```

```java
package ui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;

import javax.swing.JPanel;
import javax.swing.JTextPane;
import javax.swing.border.EmptyBorder;

public class QuickStartGuide extends JPanel {

    private static final long serialVersionUID = 0;

    // Render the quick start guide with HTML
    private static final String message =
        "<html>" +
```

```
            "<h1 style=\"font-family: serif;\">Quick Start Guide</h1>" +
            "<ul>" +
                "<li style=\"margin-bottom: 3px\">Add shapes via the left picker panel, top
menu bar, or <b>right-click</b> context menu" +
                "<li style=\"margin-bottom: 3px\">Edit the text of shapes by <b>double-
clicking</b> inside" +
                "<li style=\"margin-bottom: 3px\">Change the style of shapes via the <b>right-
click</b> \"Edit\" menu" +
                "<li style=\"margin-bottom: 3px\">Link shapes with lines via the <b>right-
click</b> \"Connections\" menu" +
                "<li style=\"margin-bottom: 3px\">Move shapes around by <b>dragging</b> them"
+
                "<li style=\"margin-bottom: 3px\">Pan the canvas by <b>dragging</b> on empty
space and zoom with the <b>scroll wheel</b>" +
                "<li style=\"margin-bottom: 3px\">Options for open, save, save as, export,
etc. can be found in the \"File\" menu" +
                "<li style=\"margin-bottom: 3px\">Toggle the grid by pressing <b>F4</b> and
the picker panel <b>F12</b>" +
            "</ul>" +
        "</html>"
    ;

    private static JTextPane guide;

    public QuickStartGuide() {
        this.setPreferredSize(new Dimension(500, 400));
        this.setBorder(new EmptyBorder(0, 10, 20, 40));
        this.setBackground(Color.white);
        this.setLayout(new BorderLayout());

        guide = new JTextPane();
        guide.setContentType("text/html");
        guide.setText(message);
        this.add(guide, BorderLayout.CENTER);
    }

}
```

1.2. shapes

```
package shapes;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.awt.Point;
import java.lang.reflect.Constructor;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.UUID;

import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;

import ui.CanvasPanel;
import ui.Viewport;

/**
 * The MapController manages and controls mind map elements
 */
public class MapController {

    public Point dragStartPoint;

    private List<MapShape> shapes;
    private MapShape selectedShape;
    private MapShape prevSelectedShape;
    public int shapeSelectionIndex = 0;
    private MapShape editingShape;

    public boolean isConnecting;
    private List<MapLine> connections;
    private MapShape origin;

    private CanvasPanel canvasPanel;
    private Viewport viewport;

    public MapController(CanvasPanel canvasPanel, Viewport viewport) {
        this.canvasPanel = canvasPanel;
        this.viewport = viewport;

        shapes = new ArrayList<MapShape>();
        connections = new ArrayList<MapLine>();
    }

    public void addShape(String shapeClassName) {
        int shapeWidth = 200, shapeHeight = 100;
        Point p = new Point();                // Point on screen to create the shape
        if (canvasPanel.isContextTrigger) {
            // Offset cursor to be consistent with shape location in viewport
            p.x = (int) ((canvasPanel.contextTriggerEvent.getX() - viewport.xOffset) /
viewport.zoomFactor);
```

```
            p.y = (int) ((canvasPanel.contextTriggerEvent.getY() - viewport.yOffset) /
viewport.zoomFactor);
        } else {
            // Start from center and find vacant location
            p.x = (int) ((canvasPanel.getWidth()/2 - viewport.xOffset) / viewport.zoomFactor);
            p.y = (int) ((canvasPanel.getHeight()/2 - viewport.yOffset) /
viewport.zoomFactor);
            p = findVacantPoint(p);
        }

        try {
            // Create new MapShape at center of point using Java reflection
            Class<?> newMapShapeClass = Class.forName(shapeClassName);
            Constructor<?> newMapShapeCons = newMapShapeClass.getConstructor(
                                                new Class<?>[] {int.class, int.class,
int.class, int.class});
            Object[] newMapShapeParameters = {p.x-(shapeWidth/2), p.y-(shapeHeight/2),
shapeWidth, shapeHeight};
            MapShape newMapShape =
(MapShape)newMapShapeCons.newInstance(newMapShapeParameters);

            // Continously repaint panel when editing to display changes in the text field
            newMapShape.getTextField().getDocument().addDocumentListener(new
DocumentListener() {
                public void insertUpdate(DocumentEvent e) {
                    canvasPanel.repaint();
                }
                public void removeUpdate(DocumentEvent e) {
                    canvasPanel.repaint();
                }
                public void changedUpdate(DocumentEvent e) {
                }
            });

            shapes.add(newMapShape);

            setSelectedShape(null);
            setSelectedShape(newMapShape);        // Select the newly added shape
        } catch (Exception e) {
            e.printStackTrace();
        }

        canvasPanel.repaint();
    }

    /**
     * Recursively check if point is already occupied by a shape and offset southeast if true
     * @param p Starting point
     * @return Vacant point
     */
    private Point findVacantPoint(Point p) {
        for (MapShape shape : shapes) {
            Point shapeLocation = shape.getShape().getBounds().getLocation();
            shapeLocation.translate(shape.getShape().getBounds().width/2,
shape.getShape().getBounds().height/2);
            if (p.equals(shapeLocation)) {
                p.translate(20, 20);
                return findVacantPoint(p);
            }
        }
        return p;
```

```
    }

    public void removeSelectedShape() {
        shapes.remove(selectedShape);
        canvasPanel.repaint();
    }

    public void changeBorderWidth(int borderWidth) {
        selectedShape.setBorderStroke(new BasicStroke(borderWidth));
        canvasPanel.repaint();
    }

    public void changeBorderColour(Color borderColour) {
        selectedShape.setBorderColour(borderColour);
        canvasPanel.repaint();
    }

    public void changeFont(String fontName) {
        selectedShape.getTextField().setFont(new Font(fontName,
                                    selectedShape.getTextField().getFont().getStyle(),
                                    selectedShape.getTextField().getFont().getSize()));
        canvasPanel.repaint();
    }

    public void changeFontStyle(int fontStyle) {
        selectedShape.getTextField().setFont(new
Font(selectedShape.getTextField().getFont().getFamily(),
                                    fontStyle,
                                    selectedShape.getTextField().getFont().getSize()));
        canvasPanel.repaint();
    }

    public void changeFontSize(int fontSize) {
        selectedShape.getTextField().setFont(new
Font(selectedShape.getTextField().getFont().getFamily(),
                                    selectedShape.getTextField().getFont().getStyle(),
                                    fontSize));
        canvasPanel.repaint();
    }

    public void changeFontColour(Color fontColour) {
        selectedShape.getTextField().setForeground(fontColour);
        canvasPanel.repaint();
    }

    public void bringSelectedShapeToFront() {
        Collections.swap(shapes, shapes.indexOf(selectedShape), shapes.size()-1);
        canvasPanel.repaint();
    }

    public void bringSelectedShapeForwards() {
        // Do nothing if selected shape is already at the front
        if (! (shapes.indexOf(selectedShape) == shapes.size()-1)) {
            Collections.swap(shapes, shapes.indexOf(selectedShape),
shapes.indexOf(selectedShape)+1);
            canvasPanel.repaint();
        }
    }

    public void sendSelectedShapeToBack() {
        Collections.swap(shapes, shapes.indexOf(selectedShape), 0);
```

```
        canvasPanel.repaint();
    }

    public void sendSelectedShapeBackward() {
        // Do nothing if selected shape is already at the back
        if (! (shapes.indexOf(selectedShape) == 0)) {
            Collections.swap(shapes, shapes.indexOf(selectedShape),
shapes.indexOf(selectedShape)-1);
            canvasPanel.repaint();
        }
    }

    public void setSelectedShapeAsOrigin() {
        origin = selectedShape;
    }

    public void setSelectedShapeAsTermination() {
        connections.add(new MapLine(origin, selectedShape));
        canvasPanel.repaint();
        origin = null;
    }

    public void removeConnectionsFromSelectedShape() {
        origin = null;
        List<MapLine> toBeRemoved = new ArrayList<MapLine>();
        for (MapLine connection : connections) {
            if (connection.getOrigin() == selectedShape || connection.getTermination() ==
selectedShape) {
                toBeRemoved.add(connection);
            }
        }
        connections.removeAll(toBeRemoved);
        canvasPanel.repaint();
    }


    // Getters and setters
    public List<MapShape> getShapes() {
        return shapes;
    }
    public List<MapShape> getShapesUnderCursor(Point cursor) {
        // Offset cursor to be consistent with shape location in viewport
        cursor.translate(-(int)viewport.xOffset, -(int)viewport.yOffset);
        cursor.x /= viewport.zoomFactor;
        cursor.y /= viewport.zoomFactor;

        List<MapShape> shapesUnderCursor = new ArrayList<MapShape>();
        for (MapShape shape : shapes)
            if (shape.getShape().getBounds().contains(cursor)) shapesUnderCursor.add(shape);
        return shapesUnderCursor;
    }
    public MapShape getSelectedShape() {
        return selectedShape;
    }
    public void setSelectedShape(MapShape selectedShape) {
        prevSelectedShape = this.selectedShape;
        this.selectedShape = selectedShape;
        if (prevSelectedShape != null) prevSelectedShape.isHighlighted = false;
        if (selectedShape != null) selectedShape.isHighlighted = true;
        else {        // If null is passed in, disable highlight for all shapes
            for (MapShape shape : shapes) shape.isHighlighted = false;
```

```
            shapeSelectionIndex = 0;
        }
    }
    public MapShape getEditingShape() {
        return editingShape;
    }
    public void setEditingShape(MapShape editingShape) {
        this.editingShape = editingShape;
    }
    public List<MapLine> getConnections() {
        return connections;
    }
    public MapShape getConnectionOrigin() {
        return origin;
    }
    public JsonArray getShapesAsJsonArray() {
        JsonArray shapesData = new JsonArray();
        for (MapShape shape : shapes) {
            shapesData.add(shape.getAsJsonObject());
        }
        return shapesData;
    }
    public JsonArray getConnectionsAsJsonArray() {
        JsonArray connectionsData = new JsonArray();
        for (MapLine connection : connections) {
            connectionsData.add(connection.getAsJsonObject());
        }
        return connectionsData;
    }
    public void replaceShapesFromJson(JsonArray shapesData) {
        shapes = new ArrayList<MapShape>();
        for (JsonElement shape : shapesData) {
            JsonObject thisShape = shape.getAsJsonObject();
            try {
                // Create new MapShape from Json string using Java reflection
                Class<?> newMapShapeClass =
Class.forName(thisShape.get("Type").getAsString());
                Constructor<?> newMapShapeCons = newMapShapeClass.getConstructor(
                                                    new Class<?>[] {int.class, int.class,
int.class, int.class});
                Object[] newMapShapeParameters = {thisShape.get("X").getAsInt(),
thisShape.get("Y").getAsInt(),
                                                  thisShape.get("Width").getAsInt(),
thisShape.get("Height").getAsInt()};
                MapShape newMapShape =
(MapShape)newMapShapeCons.newInstance(newMapShapeParameters);

                newMapShape.setId(UUID.fromString(thisShape.get("ID").getAsString()));
                newMapShape.setBorderStroke(new BasicStroke(thisShape.get("Border
width").getAsInt()));
                newMapShape.setBorderColour(Color.decode(thisShape.get("Border
colour").getAsString()));
                newMapShape.getTextField().setText(thisShape.get("Text").getAsString());
                newMapShape.getTextField().setFont(new Font(thisShape.get("Text font
name").getAsString(),
                                                    thisShape.get("Text font
style").getAsInt(),
                                                    thisShape.get("Text font
size").getAsInt()));
                newMapShape.getTextField().setForeground(Color.decode(thisShape.get("Font
colour").getAsString()));
```

```
                // Continously repaint panel when editing to display changes in the text field
                newMapShape.getTextField().getDocument().addDocumentListener(new
DocumentListener() {
                    public void insertUpdate(DocumentEvent e) {
                        canvasPanel.repaint();
                    }
                    public void removeUpdate(DocumentEvent e) {
                        canvasPanel.repaint();
                    }
                    public void changedUpdate(DocumentEvent e) {
                    }
                });

                shapes.add(newMapShape);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    public void replaceConnectionsFromJson(JsonArray connectionsData) {
        connections = new ArrayList<MapLine>();
        for (JsonElement connection : connectionsData) {
            JsonObject thisConnection = connection.getAsJsonObject();
            MapShape origin = null, termination = null;
            for (MapShape shape : shapes) {
                if (shape.getId().equals(UUID.fromString(thisConnection.get("Origin
ID").getAsString()))) {
                    origin = shape;
                }
                if (shape.getId().equals(UUID.fromString(thisConnection.get("Termination
ID").getAsString()))) {
                    termination = shape;
                }
                if (origin != null && termination != null) break;        // Break early if
connection found
            }
            if (origin != null && termination != null) connections.add(new MapLine(origin,
termination));
        }
    }

}
```

```
package shapes;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.awt.Shape;
import java.util.UUID;

import javax.swing.JTextField;

import com.google.gson.JsonObject;

public abstract class MapShape {

    private UUID id;
```

```
    protected Shape shape;
    protected int x, y;
    private BasicStroke borderStroke;
    private Color borderColour;
    private JTextField textField;
    public boolean isHighlighted = false;

    public MapShape(Shape shape) {
        setId(UUID.randomUUID());               // Generate new random UUID upon
instantiation
        this.shape = shape;
        this.x = shape.getBounds().x;
        this.y = shape.getBounds().y;
        // Set defaults
        borderStroke = new BasicStroke(3);
        borderColour = Color.black;
        textField = new JTextField("Example");
        textField.setFont(new Font("Serif", Font.PLAIN, 12));
        textField.setForeground(Color.black);
        textField.setBorder(null);                // Remove border
        textField.setOpaque(false);                // Transparent background for text field
        textField.setHorizontalAlignment(JTextField.CENTER);
        updateTextFieldBounds();
    }

    public void updateTextFieldBounds() {
        textField.setBounds(x + shape.getBounds().width/2 - 100, y +
shape.getBounds().height/2 - 50, 200, 100);
    }

    // Getters and setters
    public UUID getId() {
        return id;
    }
    public void setId(UUID id) {
        this.id = id;
    }
    public Shape getShape() {
        return shape;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
    public abstract void setNewCoordinates(int x, int y);    // Force subclasses (shapes) to
override this
    public BasicStroke getBorderStroke() {
        return borderStroke;
    }
    public void setBorderStroke(BasicStroke borderStroke) {
        this.borderStroke = borderStroke;
    }
    public Color getBorderColour() {
        return borderColour;
    }
    public void setBorderColour(Color borderColour) {
        this.borderColour = borderColour;
    }
    public JTextField getTextField() {
```

```
        return textField;
    }
    public JsonObject getAsJsonObject() {
        JsonObject thisShape = new JsonObject();
        thisShape.addProperty("ID", id.toString());
        thisShape.addProperty("Type", this.getClass().getName());
        thisShape.addProperty("X", x);
        thisShape.addProperty("Y", y);
        thisShape.addProperty("Width", shape.getBounds().width);
        thisShape.addProperty("Height", shape.getBounds().height);
        thisShape.addProperty("Border width", borderStroke.getLineWidth());
        thisShape.addProperty("Border colour",
"#"+Integer.toHexString(borderColour.getRGB()).substring(2));
        thisShape.addProperty("Text", textField.getText());
        thisShape.addProperty("Text font name", textField.getFont().getName());
        thisShape.addProperty("Text font style", textField.getFont().getStyle());
        thisShape.addProperty("Text font size", textField.getFont().getSize());
        thisShape.addProperty("Font colour",
"#"+Integer.toHexString(textField.getForeground().getRGB()).substring(2));
        return thisShape;
    }

}
```

```
package shapes;

import java.awt.geom.Ellipse2D;

public class EllipseShape extends MapShape {

    public EllipseShape(int x, int y, int width, int height) {
        super(new Ellipse2D.Double(x, y, width, height));
    }

    @Override
    public void setNewCoordinates(int x, int y) {
        this.x = x;
        this.y = y;
        shape = new Ellipse2D.Double(x, y, shape.getBounds().width, shape.getBounds().height);
        updateTextFieldBounds();
    }

}
```

```
package shapes;

import java.awt.geom.Rectangle2D;

public class RectangleShape extends MapShape {

    public RectangleShape(int x, int y, int width, int height) {
        super(new Rectangle2D.Double(x, y, width, height));
    }

    @Override
    public void setNewCoordinates(int x, int y) {
        this.x = x;
        this.y = y;
        shape = new Rectangle2D.Double(x, y, shape.getBounds().width,
shape.getBounds().height);
```

```
        updateTextFieldBounds();
    }

}
```

```
package shapes;

import java.awt.Shape;
import java.awt.geom.Path2D;

public class TriangleShape extends MapShape {

    public TriangleShape(int x, int y, int width, int height) {
        super(createNewTriangle(x, y, width, height));
    }

    private static Shape createNewTriangle(int x, int y, int width, int height) {
        Path2D triangle = new Path2D.Double();
        triangle.moveTo(x + width/2, y);             // Draw top point
        triangle.lineTo(x, y + height);             // Draw left point
        triangle.lineTo(x + width, y + height);      // Draw left point
        triangle.closePath();
        return triangle;
    }

    @Override
    public void setNewCoordinates(int x, int y) {
        this.x = x;
        this.y = y;
        shape = createNewTriangle(x, y, shape.getBounds().width, shape.getBounds().height);
        updateTextFieldBounds();
    }

}
```

```
package shapes;

import java.awt.BasicStroke;
import java.awt.Point;
import java.awt.geom.Line2D;

import com.google.gson.JsonObject;

public class MapLine {

    private Line2D line;
    private BasicStroke stroke;

    private MapShape origin;
    private MapShape termination;

    public MapLine(MapShape origin, MapShape termination) {
        stroke = new BasicStroke(2);
        this.origin = origin;
        this.termination = termination;
        updateConnection();
    }
```

```
    public void updateConnection() {
        // Line exists between the center of the origin and the termination
        Point originP = origin.getShape().getBounds().getLocation();
        originP.translate(origin.getShape().getBounds().width/2,
origin.getShape().getBounds().height/2);
        Point terminationP = termination.getShape().getBounds().getLocation();
        terminationP.translate(termination.getShape().getBounds().width/2,
termination.getShape().getBounds().height/2);

        line = new Line2D.Double(originP, terminationP);
    }

    // Getters
    public BasicStroke getStroke() {
        return stroke;
    }
    public MapShape getOrigin() {
        return origin;
    }
    public MapShape getTermination() {
        return termination;
    }
    public Line2D getLine() {
        return line;
    }
    public JsonObject getAsJsonObject() {
        JsonObject thisConnection = new JsonObject();
        thisConnection.addProperty("Stroke width", stroke.getLineWidth());
        thisConnection.addProperty("Origin ID", origin.getId().toString());
        thisConnection.addProperty("Termination ID", termination.getId().toString());
        return thisConnection;
    }

}
```

```
package shapes;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;

public class Grid {

    private int gridSize;
    private int gridInterval;
    private float gridWidth;
    private float gridWidthMinorAxes;
    private float gridWidthMajorAxes;
    private Color gridLines;
    private Color gridMinorAxes;
    private Color gridMajorAxes;

    public Grid(int gridSize) {
        this.gridSize = gridSize;
        gridInterval = 25;
        gridWidth = (float)0.5;
        gridWidthMinorAxes = (float)0.5;
        gridWidthMajorAxes = 2;
        gridLines = Color.lightGray;
```

```
        gridMinorAxes = Color.gray;
        gridMajorAxes = Color.gray;
    }

    public void drawGrid(Graphics2D g2d) {
        for (int x = -gridSize; x <= gridSize; x += gridInterval) {     // Draw all vertical
lines
            if (x == 0) {                                                // Draw major axes
darker
                g2d.setColor(gridMajorAxes);
                g2d.setStroke(new BasicStroke(gridWidthMajorAxes));
            } else if (x % (gridInterval*10) == 0) {                    // Draw minor axes
(every 10 intervals) slighly darker
                g2d.setColor(gridMinorAxes);
                g2d.setStroke(new BasicStroke(gridWidthMinorAxes));
            } else {
                g2d.setColor(gridLines);
                g2d.setStroke(new BasicStroke(gridWidth));
            }
            g2d.drawLine(x, -gridSize, x, gridSize);
        }
        for (int y = -gridSize; y <= gridSize; y += gridInterval) {     // Draw all horizontal
lines
            if (y == 0) {                                                // Draw major axes
darker
                g2d.setColor(gridMajorAxes);
                g2d.setStroke(new BasicStroke(gridWidthMajorAxes));
            } else if (y % (gridInterval*10) == 0) {                    // Draw minor axes
(every 10 intervals) slighly darker
                g2d.setColor(gridMinorAxes);
                g2d.setStroke(new BasicStroke(gridWidthMinorAxes));
            } else {
                g2d.setColor(gridLines);
                g2d.setStroke(new BasicStroke(gridWidth));
            }
            g2d.drawLine(-gridSize, y, gridSize, y);
        }
    }

}
```

1.3. io

```
package io;

import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

import javax.imageio.ImageIO;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.stream.JsonReader;

import main.AppFrame;
import ui.CanvasPanel;

/**
 * The IOController handles the opening, saving and exporting of mind maps
 */
public class IOController {

    private File currentFile;

    private static final Gson gson = new GsonBuilder().setPrettyPrinting().create();

    private AppFrame appFrame;
    private CanvasPanel canvasPanel;

    public IOController(AppFrame appFrame, CanvasPanel canvasPanel) {
        this.appFrame = appFrame;
        this.canvasPanel = canvasPanel;
    }

    public void handleOpen(File inFile) {
        System.out.print("Opening " + inFile.getAbsolutePath() + " ... ");
        try {
            JsonReader reader = new JsonReader(new FileReader(inFile));
            JsonObject data = gson.fromJson(reader, JsonObject.class);
            canvasPanel.getViewport().setViewportData(data.get("Viewport").getAsJsonObject());

canvasPanel.getMapController().replaceShapesFromJson(data.get("Shapes").getAsJsonArray());

canvasPanel.getMapController().replaceConnectionsFromJson(data.get("Connections").getAsJsonArr
ay());
            canvasPanel.repaint();
            setCurrentFile(inFile);
            System.out.println("Success");
        } catch (IOException e) {
            System.out.println("File not found! " + e);
        }
    }

    public void handleSave(File outFile) {
```

```
        System.out.print("Saving to " + outFile.getAbsolutePath() + " ... ");
        try {
            PrintWriter p = new PrintWriter(new FileWriter(outFile));
            JsonObject output = new JsonObject();
            output.add("Viewport",
gson.toJsonTree(canvasPanel.getViewport().getViewportData()));
            output.add("Shapes",
gson.toJsonTree(canvasPanel.getMapController().getShapesAsJsonArray()));
            output.add("Connections",
gson.toJsonTree(canvasPanel.getMapController().getConnectionsAsJsonArray()));
            p.write(gson.toJson(output));
            p.close();
            setCurrentFile(outFile);
            System.out.println("Success");
        } catch (IOException e) {
            System.out.println("File not found! " + e);
        }
    }

    public void handleExport(File file, String imgType, int scale) {
        System.out.print("Exporting to " + file.getAbsolutePath() + " ... ");
        // Upscale exported image to increase quality and enable transparency if exporting to
PNG
        BufferedImage image = new BufferedImage(canvasPanel.getWidth() * scale,
canvasPanel.getHeight() * scale,
                                                imgType == "png" ?
BufferedImage.TYPE_INT_ARGB : BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = image.createGraphics();
        // Increase quality of exported image
        g2d.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_COLOR_RENDERING,
RenderingHints.VALUE_COLOR_RENDER_QUALITY);
        g2d.setRenderingHint(RenderingHints.KEY_DITHERING,
RenderingHints.VALUE_DITHER_ENABLE);
        g2d.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
RenderingHints.VALUE_FRACTIONALMETRICS_ON);
        g2d.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
RenderingHints.VALUE_INTERPOLATION_BILINEAR);
        g2d.setRenderingHint(RenderingHints.KEY_RENDERING,
RenderingHints.VALUE_RENDER_QUALITY);
        g2d.setRenderingHint(RenderingHints.KEY_STROKE_CONTROL,
RenderingHints.VALUE_STROKE_PURE);
        g2d.scale(scale, scale);        // Upscale
        canvasPanel.printAll(g2d);
        try {
            ImageIO.write(image, imgType, file);
            System.out.println("Success");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public File getCurrentFile() {
        return currentFile;
    }
    public void setCurrentFile(File file) {
        currentFile = file;
```

```
        appFrame.setAppTitle(file.getName().substring(0, file.getName().length() - 5));    // Truncate ".json"
    }

}
```