

```

1 package StV1p1ver04;
2 /*
3
4  * SectionToVolume is made calcualte volumes based on serial sections
5  *
6  * Version 1.1
7  *
8  * Created by Anders Hånell in 2012
9
10 This software is not subject to copyright protection and is in the public domain.
11 SectionToVolume is an experimental system and the author assumes no responsibility
12 whatsoever for its use by other parties, and makes no guarantees, expressed or
13 implied, about its quality, reliability, or any other characteristic.
14
15 Send questions and suggestions to SectionToVolume@gmail.com
16
17 * The first part of the program declares variables and sets up the window.
18 *
19 * Button panel attach code to the buttons in the upper panel.
20 *
21 * PressedW/A/S/D is used for the keyboard control.
22 *
23 * HelpDialog, AboutDialog and AboutBox displays information.
24 *
25 * The CreateProjectDialogs collects information and makes some controls
26 * when creating a new project.
27 *
28 * The class Project holds code used when creating and opening projects.
29 *
30 * TreePanel and the TreeObject classes is used to execute the correct action
31 * when the user clicks the tree panel in the left panel.
32 *
33 * The MiddlePanel and RightSidePanel classes are displayed when selected in
34 * the tree panel
35 *
36 * The ProjectData and SectionData holds the data which is saved to disk.
37 * A copy of the original image file is also saved to disk
38 *
39 * MyMethods holds various methods used in the program.
40
41 */
42
43 import com.sun.media.jai.codec.FileSeekableStream;
44 import java.awt.*;
45 import java.awt.datatransfer.Clipboard;
46 import java.awt.datatransfer.StringSelection;
47 import java.awt.event.*;
48 import java.awt.geom.Ellipse2D;
49 import java.awt.image.*;
50 import java.io.*;
51 import java.util.ArrayList;
52 import java.util.zip.GZIPInputStream;
53 import java.util.zip.GZIPOutputStream;
54 import javax.imageio.ImageIO;
55 import javax.media.jai.JAI;
56 import javax.media.jai.PlanarImage;
57 import javax.swing.*;
58 import javax.swing.event.TreeSelectionEvent;
59 import javax.swing.event.TreeSelectionListener;
60 import javax.swing.table.TableColumn;

```

```

61 import javax.swing.tree.DefaultMutableTreeNode;
62 import javax.swing.tree.TreePath;
63
64 public class StV1p1ver04 extends JFrame implements WindowListener
65 {
66     public static void main(String[] args)
67     {
68         javax.swing.SwingUtilities.invokeLater(new Runnable()
69         {
70             @Override
71             public void run()
72             {
73                 createAndShowGUI();
74             }
75         });
76     }
77
78     static StV1p1ver04 frame;
79
80     //These scroll panes are added to split panes when the program is initiated
81     //The upper scrollPane always holds the new, open and save buttons
82     //The left scrollPane always holds the tree structure
83     //The content of the middle and right scrollPane depends on user action
84     static JScrollPane upperScrollPane= new JScrollPane();
85     static JScrollPane leftScrollPane= new JScrollPane();
86     static JScrollPane middleScrollPane = new JScrollPane();
87     static JScrollPane rightScrollPane= new JScrollPane();
88
89     static JSplitPane firstSplitPane;
90     static JSplitPane secondSplitPane;
91     static JSplitPane highLowSplitPane;
92
93     //These two objects are saved to disk
94     //One copy of SectionData is saved for each section in the project
95     //The original image files are also saved to disk
96     ProjectData myProjectData = new ProjectData(1,1,1,1);
97     SectionData currentSectionData = new SectionData(1,1);
98
99     //Holds some data to improve efficiency when switching between panels in the same section
100     OpenSection openSection = new OpenSection();
101
102     LR_DivideMiddlePanel lr_divideMiddlePanel = new LR_DivideMiddlePanel();
103     LR_DivideRightSidePanel lr_DivideRightSidePanel = new LR_DivideRightSidePanel();
104
105     Subject_MiddlePanel subject_middlePanel = new Subject_MiddlePanel();
106     Subject_RightSidePanel subject_rightSidePanel = new Subject_RightSidePanel();
107
108     Section_MiddlePanel section_middlePanel = new Section_MiddlePanel();
109     Section_RightSidePanel section_rightSidePanel = new Section_RightSidePanel();
110
111     AdjustImage_MiddlePanel adjustImage_middlePanel = new AdjustImage_MiddlePanel();
112
113     //The top panel which holds for example the open and save buttons
114     ButtonPanel buttonPanel = new ButtonPanel();
115
116     SectionResults_MiddlePanel sectionResults_middlePanel = new SectionResults_MiddlePanel();
117     SectionResults_RightSidePanel sectionResults_rightSidePanel = new SectionResults_RightSidePanel();
118
119     SubjectResults_MiddlePanel subjectResults_middlePanel = new SubjectResults_MiddlePanel();
120     SubjectResults_RightSidePanel subjectResults_rightSidePanel = new SubjectResults_RightSidePanel();

```

```

121
122 UDA_MiddlePanel uda_middlePanel = new UDA_MiddlePanel();
123 UDA_RightSidePanel uda_rightSidePanel = new UDA_RightSidePanel();
124
125 TissueDetection_MiddlePanel tissueDetection_middlePanel = new TissueDetection_MiddlePanel();
126 TissueDetection_RightSidePanel tissueDetection_rightSidePanel = new
TissueDetection_RightSidePanel();
127
128 Conversion_MiddlePanel conversion_middlePanel = new Conversion_MiddlePanel();
129 Conversion_RightSidePanel conversion_rightSidePanel = new Conversion_RightSidePanel();
130
131 AdjustImage_RightSidePanel adjustImage_rightSidePanel = new AdjustImage_RightSidePanel();
132
133 TreePanel treePanel = new TreePanel(1,1,1);
134
135 //This class holds methods for creating new projects and opening existing projects
136 Project myProject = new Project();
137
138 //Various methods are stored here
139 MyMethods myMethods = new MyMethods();
140
141 CreateProjectDialog1 createProjectDialog1 = new CreateProjectDialog1();
142 CreateProjectDialog2 createProjectDialog2 = new CreateProjectDialog2();
143 CreateProjectDialog3 createProjectDialog3 = new CreateProjectDialog3();
144 CreateProjectDialog4 createProjectDialog4 = new CreateProjectDialog4(1);
145 CreateProjectDialog5 createProjectDialog5 = new CreateProjectDialog5();
146
147 HelpDialog helpDialog = new HelpDialog();
148 ErrorDialog errorDialog = new ErrorDialog();
149 AboutBox aboutBox = new AboutBox();
150
151 //Used to keep track of the progress in long tasks
152 ProgressMonitor progressMonitor;
153 ProgressMonitor analyzeFolderMonitor;
154
155 BufferedImage errorImage = myMethods.createErrorImage();
156
157 private static void createAndShowGUI()
158 {
159     //Creates the main window and adds split panes and scroll panes
160     frame = new StV1p1ver04();
161     frame.setTitle("SectionToVolume 1.1");
162     frame.setSize(600,400);
163     frame.setExtendedState(MAXIMIZED_BOTH);
164     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
165
166     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
167     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
168
169     firstSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
170         middleScrollPane, rightScrollPane);
171     firstSplitPane.setResizeWeight(0.75);
172     firstSplitPane.setDividerLocation( (int) (screenWidth*14)/20);
173
174     secondSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
175         leftScrollPane, firstSplitPane);
176     secondSplitPane.setDividerLocation( (int) (screenWidth*3)/20);
177
178     highLowSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
179         upperScrollPane, secondSplitPane);

```

```

180
181     highLowSplitPane.setDividerLocation( (int) screenHeight/10);
182
183     frame.add(highLowSplitPane);
184     firstSplitPane.resetToPreferredSizes();
185     frame.setVisible(true);
186 }
187
188 public StV1p1ver04()
189 {
190     //Sets the scroll speed when using the mouse wheel
191     middleScrollPane.getVerticalScrollBar().setUnitIncrement(32);
192
193     upperScrollPane.getViewport().add(buttonPanel);
194
195     //The letters W and S can be used to zoom in and out
196     //The letters A and D can be used to move up and down in the tree structure
197     PressedW pressedW = new PressedW();
198     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
199         KeyStroke.getKeyStroke("W"), "w");
200     upperScrollPane.getActionMap().put("w", pressedW);
201
202     PressedA pressedA = new PressedA();
203     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
204         KeyStroke.getKeyStroke("A"), "a");
205     upperScrollPane.getActionMap().put("a", pressedA);
206
207     PressedS pressedS = new PressedS();
208     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
209         KeyStroke.getKeyStroke("S"), "s");
210     upperScrollPane.getActionMap().put("s", pressedS);
211
212     PressedD pressedD = new PressedD();
213     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
214         KeyStroke.getKeyStroke("D"), "d");
215     upperScrollPane.getActionMap().put("d", pressedD);
216
217     upperScrollPane.requestFocus();
218 }
219
220 @Override
221 public void windowClosing(WindowEvent e)
222 {
223     //The project is always saved before closing
224     myMethods.setWaitCursor();
225     myMethods.saveProjectData();
226     System.exit(0);
227 }
228
229 @Override
230 public void windowClosed(WindowEvent e) {}
231
232 @Override
233 public void windowOpened(WindowEvent e) {}
234
235 @Override
236 public void windowIconified(WindowEvent e) {}
237
238 @Override
239 public void windowDeiconified(WindowEvent e) {}

```

```

240
241 @Override
242 public void windowActivated(WindowEvent e) {}
243
244 @Override
245 public void windowDeactivated(WindowEvent e) {}
246
247 public void windowGainedFocus(WindowEvent e) {}
248 public void windowLostFocus(WindowEvent e) {}
249 public void windowStateChanged(WindowEvent e) {}
250
251 @SuppressWarnings("LeakingThisInConstructor")
252 private class ButtonPanel extends JPanel implements ActionListener
253 {
254     //This panel is located on the upper scroll pane
255     JButton newProjectButton;
256     JButton saveProjectButton;
257     JButton openProjectButton;
258     JButton zoomInButton;
259     JButton zoomOutButton;
260     JButton decLineWidthButton;
261     JButton incLineWidthButton;
262     JButton blackLinesButton;
263     JButton whiteLinesButton;
264     JButton helpButton;
265     JButton aboutButton;
266
267     public ButtonPanel()
268     {
269         newProjectButton=new JButton("New Project");
270         saveProjectButton=new JButton("Save Project");
271         openProjectButton=new JButton("Open Project");
272         zoomInButton=new JButton("Zoom In");
273         zoomOutButton=new JButton("Zoom Out");
274         decLineWidthButton=new JButton("- Line Width");
275         incLineWidthButton=new JButton("+ Line Width");
276         blackLinesButton=new JButton("Black lines");
277         whiteLinesButton=new JButton("White lines");
278         helpButton=new JButton("Help");
279         aboutButton=new JButton("About");
280
281         this.setLayout(new GridBagLayout());
282         GridBagConstraints c = new GridBagConstraints();
283
284         c.gridx = 0;
285         c.gridy = 0;
286         c.gridwidth = 1;
287         c.insets = new Insets(5,5,5,5);
288         this.add(newProjectButton, c);
289
290         c.gridx++;
291         this.add(saveProjectButton, c);
292
293         c.gridx++;
294         this.add(openProjectButton, c);
295
296         c.insets = new Insets(5,40,5,5);
297         c.gridx++;
298         this.add(zoomInButton, c);
299

```

```

300     c.insets = new Insets(5,5,5,5);
301     c.gridx++;
302     this.add(zoomOutButton, c);
303
304     c.insets = new Insets(5,40,5,5);
305     c.gridx++;
306     this.add(decLineWidthButton, c);
307
308     c.insets = new Insets(5,5,5,5);
309     c.gridx++;
310     this.add(incLineWidthButton, c);
311
312     c.insets = new Insets(5,40,5,5);
313     c.gridx++;
314     this.add(blackLinesButton, c);
315
316     c.insets = new Insets(5,5,5,5);
317     c.gridx++;
318     this.add(whiteLinesButton, c);
319
320     c.insets = new Insets(5,40,5,5);
321     c.gridx++;
322     this.add(helpButton, c);
323
324     c.insets = new Insets(5,5,5,5);
325     c.gridx++;
326     this.add(aboutButton, c);
327
328     newProjectButton.addActionListener(this);
329     saveProjectButton.addActionListener(this);
330     openProjectButton.addActionListener(this);
331     zoomInButton.addActionListener(this);
332     zoomOutButton.addActionListener(this);
333     decLineWidthButton.addActionListener(this);
334     incLineWidthButton.addActionListener(this);
335     blackLinesButton.addActionListener(this);
336     whiteLinesButton.addActionListener(this);
337     helpButton.addActionListener(this);
338     aboutButton.addActionListener(this);
339 }
340
341 @Override
342 public void actionPerformed(ActionEvent e)
343 {
344     if (e.getSource() == newProjectButton)
345     {
346         addWindowListener(frame);
347         createProjectDialog1.setVisible(true);
348     }
349
350     if (e.getSource() == openProjectButton)
351     {
352         addWindowListener(frame);
353
354         //The file dialog uses a file filter to hide any file except main project files
355         FileFilter fileFilter = new FileFilter();
356         File file = new File("");
357         JFileChooser fc = new JFileChooser(file);
358         fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
359         fc.setFileFilter(fileFilter);

```

```

360     int dialogResult = fc.showOpenDialog(this);
361
362     if (dialogResult==JFileChooser.APPROVE_OPTION)
363     {
364         File projectFile = fc.getSelectedFile();
365
366         ObjectInputStream in;
367         try
368         {
369             FileInputStream fis = new FileInputStream(projectFile);
370             in = new ObjectInputStream(fis);
371             myProjectData = (ProjectData)in.readObject();
372             in.close();
373             myProjectData.projectLocation=projectFile.getParent();
374         }
375         catch(IOException ex)
376         {
377             ex.printStackTrace();
378         }
379         catch(ClassNotFoundException ex){ }
380
381         myProject.refreshWhenOpened(); //Sets up the tree structure for example
382     }
383 }
384
385 if (e.getSource() == saveProjectButton)
386 {
387     myMethods.saveProjectData();
388 }
389
390 // If the zoom factor is for example 47 images will be shown at 47% of their original size
391 if (e.getSource() == zoomInButton)
392 {
393     myProjectData.zoomFactor= myProjectData.zoomFactor+10;
394     middleScrollPane.repaint();
395     middleScrollPane.revalidate();
396 }
397 if (e.getSource() == zoomOutButton)
398 {
399     if (myProjectData.zoomFactor>10) myProjectData.zoomFactor= myProjectData.zoomFactor-10;
400     middleScrollPane.repaint();
401     middleScrollPane.revalidate();
402 }
403
404 //When zoomed out thin lines might not be displayed
405 //but by increasing the line width this is avoided
406 //The setting for the line width does not affect
407 //the area calculations
408 if (e.getSource() == decLineWidthButton)
409 {
410     myMethods.setWaitCursor();
411
412     if (myProjectData.lineWidth>0) myProjectData.lineWidth--;
413
414     try
415     {
416         lr_divideMiddlePanel.generateLR_LinesImage();
417     }catch (Exception error){ }
418
419     try

```

```
420     {
421         uda_middlePanel.generateUdaPanelImage();
422     }catch (Exception error){}
423
424     try
425     {
426         conversion_middlePanel.generateConversionImage();
427     }catch (Exception error){}
428
429
430     middleScrollPane.repaint();
431     myMethods.setCustomCursor();
432 }
433 if (e.getSource() == incLineWidthButton)
434 {
435     myMethods.setWaitCursor();
436     myProjectData.lineWidth++;
437
438     try
439     {
440         lr_divideMiddlePanel.generateLR_LinesImage();
441     }catch (Exception error){}
442
443     try
444     {
445         uda_middlePanel.generateUdaPanelImage();
446     }catch (Exception error){}
447
448     try
449     {
450         conversion_middlePanel.generateConversionImage();
451     }catch (Exception error){}
452
453
454     middleScrollPane.repaint();
455     myMethods.setCustomCursor();
456 }
457
458 if (e.getSource() == blackLinesButton)
459 {
460     myMethods.setWaitCursor();
461     int[] black = {0,0,0};
462     myProjectData.lineColor= black;
463
464     try
465     {
466         lr_divideMiddlePanel.generateLR_LinesImage();
467     }catch (Exception error){}
468
469     try
470     {
471         uda_middlePanel.generateUdaPanelImage();
472     }catch (Exception error){}
473
474     try
475     {
476         conversion_middlePanel.generateConversionImage();
477     }catch (Exception error){}
478
479     middleScrollPane.repaint();
```



```

480     myMethods.setCustomCursor();
481 }
482
483 if (e.getSource() == whiteLinesButton)
484 {
485     myMethods.setWaitCursor();
486     int[] white = {255,255,255};
487     myProjectData.lineColor= white;
488
489     try
490     {
491         lr_divideMiddlePanel.generateLR_LinesImage();
492     }catch (Exception error){}
493
494     try
495     {
496         uda_middlePanel.generateUdaPanelImage();
497     }catch (Exception error){}
498
499     try
500     {
501         conversion_middlePanel.generateConversionImage();
502     }catch (Exception error){}
503
504     middleScrollPane.repaint();
505     myMethods.setCustomCursor();
506 }
507
508 if (e.getSource() == helpButton)
509 {
510     //Pressing the main help button displays the entire help text
511     String helpText=myMethods.getHelpTextChoseProjectFolder();
512     helpText=helpText+"\n"+"n";
513     helpText=helpText+myMethods.getHelpTextChoseSaveFolder();
514     helpText=helpText+"\n"+"n";
515     helpText=helpText+myMethods.getHelpTextChoseConversionImage();
516     helpText=helpText+"\n"+"n";
517     helpText=helpText+myMethods.getHelpTextBregmaLevels();
518     helpText=helpText+"\n"+"n";
519     helpText=helpText+myMethods.getHelpTextUDA();
520     helpText=helpText+"\n"+"n";
521     helpText=helpText+myMethods.getHelpTextTissueDetection();
522     helpText=helpText+"\n"+"n";
523     helpText=helpText+myMethods.getHelpTextAdjustImage();
524
525     helpDialog.newText(helpText);
526 }
527 if (e.getSource() == aboutButton)
528 {
529     aboutBox.newText(myMethods.getAboutText());
530 }
531 }
532
533 class FileFilter extends javax.swing.filechooser.FileFilter
534 {
535     //Used when navigating to project files when opening an existing project
536     //Only displays folders and the main project file
537     @Override
538     public boolean accept(File f)
539     {

```

```

540         if (f.isDirectory()) return true;
541         return (f.getName().equals("ProjectData.ser"));
542     }
543     @Override
544     public String getDescription()
545     {
546         return "Project files";
547     }
548 }
549 }
550
551 class PressedW extends AbstractAction
552 {
553     //Keyboard "W" is pressed, zooms in on the image
554     @Override
555     public void actionPerformed(ActionEvent e)
556     {
557         if (!myMethods.textFieldIsFocusOwner())
558         {
559             myProjectData.zoomFactor= myProjectData.zoomFactor+10;
560             middleScrollPane.repaint();
561             middleScrollPane.revalidate();
562         }
563     }
564 }
565 class PressedA extends AbstractAction
566 {
567     //Keyboard "A" is pressed, moves one step up in the tree panel
568     @Override
569     public void actionPerformed(ActionEvent e)
570     {
571         if (!myMethods.textFieldIsFocusOwner())
572         {
573             treePanel.tree.removeTreeSelectionListener(treePanel);
574             TreePath selectionPath = treePanel.tree.getSelectionPath();
575
576             int rowNumber = treePanel.tree.getRowForPath(selectionPath);
577             treePanel.tree.expandRow(rowNumber);
578
579             if (rowNumber<1) rowNumber=1;
580             else rowNumber--;
581
582             TreePath newSelection = treePanel.tree.getPathForRow(rowNumber);
583
584             treePanel.tree.addTreeSelectionListener(treePanel);
585             treePanel.tree.setSelectionPath(newSelection);
586         }
587     }
588 }
589 class PressedS extends AbstractAction
590 {
591     //Keyboard "S" is pressed, zooms out in the image
592     @Override
593     public void actionPerformed(ActionEvent e)
594     {
595         if (!myMethods.textFieldIsFocusOwner())
596         {
597             if (myProjectData.zoomFactor>10) myProjectData.zoomFactor= myProjectData.zoomFactor-10;
598             middleScrollPane.repaint();
599             middleScrollPane.revalidate();

```

```

600     }
601 }
602 }
603 class PressedD extends AbstractAction
604 {
605     //Keyboard "D" is pressed, moves one step down in the tree panel
606     @Override
607     public void actionPerformed(ActionEvent e)
608     {
609         if (!myMethods.textFieldIsFocusOwner())
610         {
611             treePanel.tree.removeTreeSelectionListener(treePanel);
612             TreePath selectionPath = treePanel.tree.getSelectionPath();
613
614             int rowNumber = treePanel.tree.getRowForPath(selectionPath);
615             treePanel.tree.expandRow(rowNumber);
616             int rowsDisplayed = treePanel.tree.getRowCount();
617
618             if (rowNumber==rowsDisplayed) rowNumber=1;
619             else rowNumber++;
620
621             TreePath newSelection = treePanel.tree.getPathForRow(rowNumber);
622
623             treePanel.tree.addTreeSelectionListener(treePanel);
624             treePanel.tree.setSelectionPath(newSelection);
625         }
626     }
627 }
628
629 public class HelpDialog extends JFrame
630 {
631     //Several buttons can activate the help dialog.
632     //The Help button on the upper panel will show the entire help text.
633     //The other Help buttons will only show the section relevant for
634     //that part of the program.
635
636     JTextArea textArea = new JTextArea(30, 30);
637     JScrollPane scrollPane;
638
639     public HelpDialog()
640     {
641         this.setVisible(false);
642         this.setTitle("Help");
643
644         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
645         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
646         this.setLocation(screenWidth*50/100,screenHeight*10/100);
647         this.setSize(screenWidth*45/100,screenHeight*80/100);
648
649         this.setAlwaysOnTop(true);
650         this.setModalExclusionType(Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
651
652         textArea = new JTextArea(5, 20);
653         scrollPane = new JScrollPane(textArea);
654         textArea.setEditable(false);
655         this.add(scrollPane);
656     }
657
658     public void newText(String inputText)
659     {

```

```

660 //Updates the help text when the user press one of the help buttons
661 helpDialog.remove(scrollPane);
662
663 textArea = new JTextArea(5, 20);
664 textArea.setText(inputText);
665 scrollPane = new JScrollPane(textArea);
666 textArea.setEditable(false);
667
668 this.add(scrollPane);
669 this.setVisible(true);
670 }
671 }
672
673 public class AboutBox extends JFrame
674 {
675     JTextArea textArea = new JTextArea(30, 30);
676     JScrollPane scrollPane;
677
678     public AboutBox()
679     {
680         this.setVisible(false);
681         this.setTitle("About SectionToVolume 1.1");
682
683         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
684         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
685         this.setLocation(screenWidth*5/100,screenHeight*10/100);
686         this.setSize(screenWidth*40/100,screenHeight*80/100);
687
688         this.setAlwaysOnTop(true);
689         this.setModalExclusionType(Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
690
691         textArea = new JTextArea(5, 20);
692         scrollPane = new JScrollPane(textArea);
693         textArea.setEditable(false);
694         this.add(scrollPane);
695     }
696     public void newText(String inputText)
697     {
698         helpDialog.remove(scrollPane);
699
700         textArea = new JTextArea(5, 20);
701         textArea.setText(inputText);
702         scrollPane = new JScrollPane(textArea);
703         textArea.setEditable(false);
704
705         this.add(scrollPane);
706         this.setVisible(true);
707     }
708 }
709
710 public class ErrorDialog extends JFrame
711 {
712     JTextArea textArea = new JTextArea(30, 30);
713     JScrollPane scrollPane;
714
715     boolean firstError=true;
716     String errorText= "";
717
718     public ErrorDialog()
719     {

```

```

720     this.setVisible(false);
721
722     this.setTitle("An error occured");
723     this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
724
725     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
726     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
727     this.setLocation(screenWidth*5/100,screenHeight*10/100);
728     this.setSize(screenWidth*40/100,screenHeight*80/100);
729
730     this.setAlwaysOnTop(true);
731
732     this.setModalExclusionType(Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
733
734     textArea = new JTextArea(5, 20);
735     scrollPane = new JScrollPane(textArea);
736     textArea.setEditable(false);
737
738     this.add(scrollPane);
739 }
740
741 public void newText(String inputText)
742 {
743     errorDialog.remove(scrollPane);
744
745     if (firstError)
746     {
747         errorText=inputText;
748         firstError=false;
749     }
750     else
751     {
752         errorText=errorText+"\n"+"n"+"n"+inputText;
753     }
754
755     textArea = new JTextArea(5, 20);
756     textArea.setText(errorText);
757     scrollPane = new JScrollPane(textArea);
758     textArea.setEditable(false);
759
760     this.add(scrollPane);
761     this.setVisible(true);
762 }
763 }
764
765 private class CreateProjectDialog1 extends JDialog implements ActionListener
766 {
767     //This dialog is used to locate the folder which holds the source data
768     //It checks that it contains images which can be used to create a project
769     File sourceFileLocation;
770     File[] subjectFiles = new File[1000];
771     File[][] sectionFiles = new File[1000][1000];
772
773     int numberOfSubjects=0;
774     String[] subjectNames = new String[1000];
775     int numberOfSections=0;
776     String[][] sectionNames = new String[1000][1000];
777     int highestNumberOfSectionsInSubject;
778
779     boolean[][] sectionExists = new boolean[1000][1000];

```

```

780
781 JButton helpButton;
782 JButton cancelButton;
783 JButton nextButton;
784 JButton ImagesLocationButton;
785
786 JLabel infoLabel = new JLabel("The project folder should contain one folder for each subject (subject
folder)");
787 JLabel infoLabel2 = new JLabel("Each subject folder should contain one image file for each section");
788 JLabel imageLocationLabel = new JLabel("Selected folder:");
789 JLabel detectedSubjectsLabel = new JLabel("Detected Subjects: ");
790 JLabel detectedSectionsLabel = new JLabel("Detected Sections: ");
791 JLabel statusLabel = new JLabel("Status: No folder selected");
792 JLabel warningLabel = new JLabel("");
793
794 boolean allImagesIsOfSameSize;
795 int detectedImageWidth;
796 int detectedImageHeight;
797
798 JPanel panel = new JPanel();
799
800 @SuppressWarnings("LeakingThisInConstructor")
801 public CreateProjectDialog1()
802 {
803     this.setTitle("Choose source images");
804     this.setModalityType(ModalityType.APPLICATION_MODAL);
805
806     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
807     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
808     this.setLocation(screenWidth*10/100,screenHeight*10/100);
809     this.setSize(screenWidth*35/100,screenHeight*80/100);
810
811     for (int i=0; i<1000; i++)
812         for (int j=0; j<1000; j++) sectionExists[i][j]=false;
813
814     helpButton = new JButton("Help");
815     ImagesLocationButton=new JButton("Select project folder");
816     nextButton=new JButton("Next");
817     cancelButton=new JButton("Cancel");
818     nextButton.setEnabled(false);
819     statusLabel.setForeground(Color.red);
820     warningLabel.setForeground(Color.ORANGE);
821
822     panel.setLayout(new GridBagLayout());
823     GridBagConstraints c = new GridBagConstraints();
824
825     c.gridx = 0;
826     c.gridy = 0;
827     c.anchor = GridBagConstraints.PAGE_START;
828     c.gridwidth=3;
829     c.insets = new Insets(20,5,5,5);
830     panel.add(infoLabel, c);
831
832     c.insets = new Insets(5,5,5,5);
833     c.gridy++;
834     panel.add(infoLabel2, c);
835
836     c.gridy++;
837     c.insets = new Insets(20,5,5,5);
838     panel.add(ImagesLocationButton, c);

```

```

839
840     c.insets = new Insets(5,5,5,5);
841     c.gridy++;
842     panel.add(imageLocationLabel, c);
843
844     c.insets = new Insets(20,5,5,5);
845     c.gridy++;
846     panel.add(detectedSubjectsLabel, c);
847
848     c.insets = new Insets(5,5,5,5);
849     c.gridy++;
850     panel.add(detectedSectionsLabel, c);
851
852     c.insets = new Insets(20,5,5,5);
853     c.gridy++;
854     panel.add(statusLabel, c);
855
856     c.insets = new Insets(5,5,5,5);
857     c.gridy++;
858     panel.add(warningLabel, c);
859
860     c.weightx=0.3333;
861     c.gridy++;
862     c.gridwidth=1;
863     c.insets = new Insets(20,5,5,5);
864     c.anchor=GridBagConstraints.FIRST_LINE_END;
865     c.gridx=0;
866     panel.add(cancelButton, c);
867
868     c.anchor = GridBagConstraints.PAGE_START;
869     c.gridx=1;
870     panel.add(helpButton, c);
871
872     c.anchor = GridBagConstraints.FIRST_LINE_START;
873     c.weighty=1;
874     c.gridx=2;
875     panel.add(nextButton, c);
876
877     this.getContentPane().removeAll();
878     this.add(new JScrollPane(panel));
879
880     helpButton.addActionListener(this);
881     ImagesLocationButton.addActionListener(this);
882     nextButton.addActionListener(this);
883     cancelButton.addActionListener(this);
884 }
885
886 @Override
887 public void actionPerformed(ActionEvent e)
888 {
889     if (e.getSource()==helpButton)
890     {
891         String helpText = myMethods.getHelpTextChoseProjectFolder();
892         helpDialog.newText(helpText);
893     }
894     if (e.getSource()==cancelButton)
895     {
896         this.setVisible(false);
897     }
898     if (e.getSource()==ImagesLocationButton)

```

```

899     {
900         File file = new File("");
901         JFileChooser fc = new JFileChooser(file);
902         fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
903         int dialogResult = fc.showOpenDialog(this);
904
905         if (dialogResult==JFileChooser.APPROVE_OPTION)
906         {
907             createProjectDialog1.setVisible(false);
908
909             sourceFileLocation = fc.getSelectedFile();
910
911             //When the user selects a folder this makes sure that it can be used to create a project
912             analyzeFolderMonitor = new ProgressMonitor(this,
913                 "Checking folder",
914                 "Folders checked: 0", 0, myMethods.countNumberOfFiles(sourceFileLocation));
915             analyzeFolderMonitor.setMillisToDecideToPopup(0);
916             analyzeFolderMonitor.setMillisToPopup(0);
917
918             CheckForSections checkForSections = new CheckForSections();
919             checkForSections.execute();
920         }
921     }
922     if (e.getSource()==nextButton)
923     {
924         this.setVisible(false);
925         createProjectDialog2.setVisible(true);
926     }
927 }
928
929 public class CheckForSections extends SwingWorker<Void, Void>
930 {
931     @Override
932     public Void doInBackground()
933     {
934         //Checks that the folder contains images, that the images can be read
935         //and that they are of the same size
936
937         frame.setVisible(false);
938         File[] firstLevelFiles;
939         File[][] secondLevelFiles = new File[1000][1000];
940         int filesChecked=0;
941         highestNumberOfSectionsInSubject=0;
942
943         for (int i=0; i<1000; i++)
944             for (int j=0; j<1000; j++) sectionExists[i][j]=false;
945
946         analyzeFolderMonitor.setProgress(filesChecked);
947         analyzeFolderMonitor.setNote("Folders checked: 0");
948
949         String openLocation = sourceFileLocation.getName();
950         imageLocationLabel.setText("Selected folder: "+openLocation);
951
952         firstLevelFiles = sourceFileLocation.listFiles();
953         numberOfSubjects=0;
954         for (int i=0; i<firstLevelFiles.length; i++)
955         {
956             //Makes sure that only directories are copied to the list of subjects
957             if (firstLevelFiles[i].isDirectory())
958                 {

```



```

959         subjectFiles[numberOfSubjects]=firstLevelFiles[i];
960         subjectNames[numberOfSubjects]=firstLevelFiles[i].getName();
961         numberOfSubjects++;
962     }
963     analyzeFolderMonitor.setNote("Folders checked: "+i);
964     filesChecked++;
965     analyzeFolderMonitor.setProgress(filesChecked);
966     if (analyzeFolderMonitor.isCanceled())
967     {
968         break;
969     }
970 }
971 detectedSubjectsLabel.setText("Detected Subjects: "+numberOfSubjects);
972
973 analyzeFolderMonitor.setNote("Files checked: 0");
974 int sectionsInSubjectCounter;
975
976 numberOfSections=0;
977 allImagesIsOfSameSize=true;
978 detectedImageWidth=-1;
979 detectedImageHeight=-1;
980
981 for (int i=0; i<numberOfSubjects; i++)
982 {
983     sectionsInSubjectCounter=0;
984     secondLevelFiles[i]=subjectFiles[i].listFiles();
985     for (int j=0; j<secondLevelFiles[i].length; j++)
986     {
987         if (secondLevelFiles[i][j].isFile())
988         {
989             if (myMethods.fileIsValidImage(secondLevelFiles[i][j]))
990             {
991                 sectionFiles[i][sectionsInSubjectCounter]=secondLevelFiles[i][j];
992                 sectionNames[i][sectionsInSubjectCounter]=secondLevelFiles[i][j].getName();
993                 sectionExists[i][sectionsInSubjectCounter]=true;
994                 numberOfSections++;
995                 sectionsInSubjectCounter++;
996             }
997         }
998     }
999     filesChecked++;
1000     analyzeFolderMonitor.setProgress(filesChecked);
1001     analyzeFolderMonitor.setNote("Files checked: "+filesChecked);
1002     if (sectionsInSubjectCounter>highestNumberOfSectionsInSubject)
1003         highestNumberOfSectionsInSubject=sectionsInSubjectCounter;
1004     if (analyzeFolderMonitor.isCanceled())
1005     {
1006         break;
1007     }
1008 }
1009 if (analyzeFolderMonitor.isCanceled())
1010 {
1011     break;
1012 }
1013 detectedSectionsLabel.setText("Detected Sections: "+numberOfSections);
1014
1015 if (analyzeFolderMonitor.isCanceled())
1016 {
1017     nextButton.setEnabled(false);
1018     statusLabel.setForeground(Color.red);

```

```

1019         statusLabel.setText("Status: Cancelled by user");
1020         warningLabel.setText("");
1021     }
1022     else if (numberOfSections==0)
1023     {
1024         nextButton.setEnabled(false);
1025         statusLabel.setForeground(Color.red);
1026         statusLabel.setText("Status: No sections detected");
1027         warningLabel.setText("");
1028     }
1029     else if (!allImagesIsOfSameSize)
1030     {
1031         nextButton.setEnabled(true);
1032         statusLabel.setForeground(Color.green);
1033         statusLabel.setText("Status: Ok to proceed");
1034         warningLabel.setText("Warning: All images are not of the same size");
1035     }
1036     else
1037     {
1038         nextButton.setEnabled(true);
1039         statusLabel.setForeground(Color.green);
1040         statusLabel.setText("Status: Ok to proceed");
1041         warningLabel.setText("");
1042     }
1043
1044     createProjectDialog4 = new CreateProjectDialog4(highestNumberOfSectionsInSubject);
1045
1046     frame.setVisible(true);
1047     frame.repaint();
1048     createProjectDialog1.setVisible(true);
1049
1050     return null;
1051 }
1052
1053 @Override
1054 public void done() {}
1055 }
1056 }
1057
1058 @SuppressWarnings("LeakingThisInConstructor")
1059 private class CreateProjectDialog2 extends JDialog implements ActionListener
1060 {
1061     //Used to locate a folder where project data should be saved.
1062     //Makes sure that the folder is empty since a lot of files
1063     //are stored in this folder
1064     String result;
1065     File saveFileLocation;
1066
1067     JLabel infoLabel = new JLabel("The folder where the project should be saved has to be empty");
1068     JLabel saveLocationLabel = new JLabel("Selected folder:");
1069     JLabel statusLabel = new JLabel("Status: No Folder Selected");
1070
1071     JButton saveLocationButton;
1072     JButton previousButton;
1073     JButton nextButton;
1074     JButton helpButton;
1075
1076     JPanel panel = new JPanel();
1077
1078     public CreateProjectDialog2()

```

```

1079 {
1080     this.setTitle("Choose save location");
1081     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1082     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1083     this.setLocation(screenWidth*10/100,screenHeight*10/100);
1084     this.setSize(screenWidth*35/100,screenHeight*80/100);
1085
1086     this.setModalityType(ModalityType.APPLICATION_MODAL);
1087
1088     saveLocationButton=new JButton("Select save folder");
1089     previousButton=new JButton("Previous");
1090     nextButton=new JButton("Next");
1091     nextButton.setEnabled(false);
1092     helpButton=new JButton("Help");
1093
1094     statusLabel.setBackground(Color.black);
1095     statusLabel.setForeground(Color.red);
1096
1097     panel.setLayout(new GridBagLayout());
1098     GridBagConstraints c = new GridBagConstraints();
1099
1100     c.gridx = 0;
1101     c.gridy = 0;
1102     c.gridwidth = 3;
1103     c.anchor = GridBagConstraints.PAGE_START;
1104     c.insets = new Insets(20,5,5,5);
1105     panel.add(infoLabel, c);
1106
1107     c.gridy++;
1108     panel.add(saveLocationButton, c);
1109
1110     c.insets = new Insets(5,5,5,5);
1111     c.gridy++;
1112     panel.add(saveLocationLabel, c);
1113
1114     c.insets = new Insets(20,5,5,5);
1115     c.gridy++;
1116     panel.add(statusLabel, c);
1117
1118     c.weighty=1;
1119     c.weightx=0.3333;
1120     c.gridy++;
1121     c.gridwidth = 1;
1122     c.gridx=0;
1123     c.anchor=GridBagConstraints.FIRST_LINE_END;
1124     panel.add(previousButton, c);
1125
1126     c.anchor = GridBagConstraints.PAGE_START;
1127     c.gridx = 1;
1128     panel.add(helpButton, c);
1129
1130     c.anchor=GridBagConstraints.FIRST_LINE_START;
1131     c.gridx = 2;
1132     panel.add(nextButton, c);
1133
1134     this.getContentPane().removeAll();
1135     this.add(new JScrollPane(panel));
1136
1137     saveLocationButton.addActionListener(this);
1138     previousButton.addActionListener(this);

```

```

1139     nextButton.addActionListener(this);
1140     helpButton.addActionListener(this);
1141 }
1142
1143 @Override
1144 public void actionPerformed(ActionEvent e)
1145 {
1146     if (e.getSource()==saveLocationButton)
1147     {
1148         File[] filesInFolder;
1149
1150         File file = new File("");
1151         JFileChooser fc = new JFileChooser(file);
1152
1153         fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
1154         int dialogResult = fc.showOpenDialog(this);
1155
1156         if (dialogResult==JFileChooser.APPROVE_OPTION)
1157         {
1158             saveFileLocation = fc.getSelectedFile();
1159             String openLocation = saveFileLocation.getName();
1160             saveLocationLabel.setText("Selected folder: "+openLocation);
1161
1162             int numberOfFilesInFolder=0;
1163             if (saveFileLocation.isDirectory())
1164             {
1165                 filesInFolder = saveFileLocation.listFiles();
1166                 numberOfFilesInFolder = filesInFolder.length;
1167             }
1168             if (!saveFileLocation.isDirectory())
1169             {
1170                 statusLabel.setForeground(Color.red);
1171                 statusLabel.setText("Status: Not a folder");
1172                 nextButton.setEnabled(false);
1173             }
1174             if (saveFileLocation.isDirectory() && numberOfFilesInFolder>0)
1175             {
1176                 nextButton.setEnabled(false);
1177                 statusLabel.setForeground(Color.red);
1178                 statusLabel.setText("Status: Folder is not empty");
1179             }
1180             if (saveFileLocation.isDirectory() && numberOfFilesInFolder==0)
1181             {
1182                 nextButton.setEnabled(true);
1183                 statusLabel.setText("Status: Ok to proceed");
1184                 statusLabel.setForeground(Color.green);
1185             }
1186         }
1187     }
1188     if (e.getSource()==helpButton)
1189     {
1190         String helpText = myMethods.getHelpTextChoseSaveFolder();
1191         helpDialog.newText(helpText);
1192     }
1193
1194     if (e.getSource()==previousButton)
1195     {
1196         this.setVisible(false);
1197         createProjectDialog1.setVisible(true);
1198     }

```

```

1199
1200     if (e.getSource()==nextButton)
1201     {
1202         this.setVisible(false);
1203         createProjectDialog3.setVisible(true);
1204     }
1205 }
1206 }
1207
1208 @SuppressWarnings("LeakingThisInConstructor")
1209 private class CreateProjectDialog3 extends JDialog implements ActionListener
1210 {
1211     //Locates an image used to convert pixels to mm. Makes sure that it is a
1212     //readable image of the same size as the images used to create the project
1213     File conversionFileLocation;
1214     JLabel infoLabel = new JLabel("The conversion image is used to convert pixels to mm.");
1215     JLabel infoLabel2 = new JLabel("It preferably contains an object of known size.");
1216     JLabel locationLabel = new JLabel("Selected file:");
1217     JLabel statusLabel = new JLabel("Status: No file selected");
1218     JLabel warningLabel = new JLabel("");
1219
1220     JButton helpButton;
1221     JButton conversionButton;
1222     JButton previousButton;
1223     JButton nextButton;
1224
1225     JPanel panel = new JPanel();
1226
1227     public CreateProjectDialog3()
1228     {
1229         this.setTitle("Choose conversion image");
1230         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1231         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1232         this.setLocation(screenWidth*10/100,screenHeight*10/100);
1233         this.setSize(screenWidth*35/100,screenHeight*80/100);
1234
1235         this.setModalityType(ModalityType.APPLICATION_MODAL);
1236
1237         helpButton=new JButton("Help");
1238         conversionButton=new JButton("Select Conversion Image");
1239         previousButton=new JButton("Previous");
1240         nextButton=new JButton("Next");
1241         nextButton.setEnabled(false);
1242         statusLabel.setForeground(Color.red);
1243         warningLabel.setForeground(Color.orange);
1244
1245         panel.setLayout(new GridBagLayout());
1246         GridBagConstraints c = new GridBagConstraints();
1247
1248         c.gridx = 0;
1249         c.gridy = 0;
1250         c.gridwidth = 2;
1251         c.anchor = GridBagConstraints.PAGE_START;
1252         c.insets = new Insets(15,5,5,5);
1253         c.gridwidth = 3;
1254         panel.add(infoLabel, c);
1255
1256         c.insets = new Insets(5,5,5,5);
1257         c.gridy++;
1258         panel.add(infoLabel2, c);

```

```

1259
1260     c.insets = new Insets(20,5,5,5);
1261     c.gridy++;
1262     panel.add(conversionButton, c);
1263
1264     c.insets = new Insets(5,5,5,5);
1265     c.gridy++;
1266     panel.add(locationLabel, c);
1267
1268     c.insets = new Insets(20,5,5,5);
1269     c.gridy++;
1270     panel.add(statusLabel, c);
1271
1272     c.insets = new Insets(20,5,5,5);
1273     c.gridy++;
1274     panel.add(warningLabel, c);
1275
1276     c.gridy++;
1277     c.gridx=0;
1278     c.weightx=0.3333;
1279     c.gridwidth = 1;
1280     c.anchor=GridBagConstraints.FIRST_LINE_END;
1281     panel.add(previousButton, c);
1282
1283     c.gridx=1;
1284     c.anchor = GridBagConstraints.PAGE_START;
1285     panel.add(helpButton, c);
1286
1287     c.anchor=GridBagConstraints.FIRST_LINE_START;
1288     c.weighty=1;
1289     c.gridx = 2;
1290     c.gridwidth = 1;
1291     panel.add(nextButton, c);
1292
1293     this.getContentPane().removeAll();
1294     this.add(new JScrollPane(panel));
1295
1296     helpButton.addActionListener(this);
1297     conversionButton.addActionListener(this);
1298     nextButton.addActionListener(this);
1299     previousButton.addActionListener(this);
1300 }
1301
1302 @Override
1303 public void actionPerformed(ActionEvent e)
1304 {
1305     if (e.getSource()==helpButton)
1306     {
1307         String helpText = myMethods.getHelpTextChoseConversionImage();
1308         helpDialog.newText(helpText);
1309     }
1310
1311     if (e.getSource()==conversionButton)
1312     {
1313         File file = new File("");
1314         JFileChooser fc = new JFileChooser(file);
1315         fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
1316         int dialogResult = fc.showOpenDialog(this);
1317
1318         if (dialogResult==JFileChooser.APPROVE_OPTION)

```

```

1319     {
1320         conversionFileLocation = fc.getSelectedFile();
1321         String openLocation = conversionFileLocation.getName();
1322         locationLabel.setText("Selected file: "+openLocation);
1323
1324         if (myMethods.fileIsValidImage(conversionFileLocation))
1325         {
1326             if (createProjectDialog1.allImagesIsOfSameSize)
1327             {
1328                 statusLabel.setText("Ok to proceed");
1329                 nextButton.setEnabled(true);
1330                 statusLabel.setForeground(Color.green);
1331                 warningLabel.setText("");
1332             }
1333             else
1334             {
1335                 statusLabel.setText("Ok to proceed");
1336                 nextButton.setEnabled(true);
1337                 statusLabel.setForeground(Color.green);
1338                 warningLabel.setText("Warning: All images are not of the same size");
1339             }
1340         }
1341         else
1342         {
1343             statusLabel.setText("Status: Could not read image format");
1344             statusLabel.setForeground(Color.red);
1345             nextButton.setEnabled(false);
1346             warningLabel.setText("");
1347         }
1348     }
1349 }
1350
1351 if (e.getSource()==previousButton)
1352 {
1353     this.setVisible(false);
1354     createProjectDialog2.setVisible(true);
1355 }
1356
1357 if (e.getSource()==nextButton)
1358 {
1359     this.setVisible(false);
1360     createProjectDialog4.setVisible(true);
1361 }
1362 }
1363 }
1364
1365 private class CreateProjectDialog4 extends JDialog implements ActionListener
1366 {
1367     //Determines the levels to be associated with each image
1368     //Makes sure that user entry is numbers and that the levels
1369     //are either continously increasing or decreasing
1370     String result;
1371
1372     double[] levels;
1373     double firstLevel=0;
1374     double distance=1;
1375
1376     JLabel firstInfo = new JLabel("Use this for equally spaced sections:");
1377     JLabel info = new JLabel("Otherwise enter levels manually here:");
1378

```

```

1379 JLabel firstLevelLabel = new JLabel("First bregma level:");
1380 JTextField firstLevelTextField = new JTextField("0", 5);
1381 JLabel firstLevelResultLabel = new JLabel("0");
1382
1383 JLabel distanceLabel = new JLabel("Distance between sections:");
1384 JTextField distanceTextField = new JTextField("1", 5);
1385 JLabel distanceResultLabel = new JLabel("0");
1386
1387 JButton calculateButton;
1388 JButton helpButton;
1389 JButton previousButton;
1390 JButton nextButton;
1391
1392 int numberOfBregmaLevels;
1393
1394 JLabel[] bregmaLevelLabel;
1395 JTextField[] bregmaLevelTextFields ;
1396 JLabel[] bregmaLevelResultLabel;
1397
1398 JLabel statusLabel = new JLabel("Ok to proceed");
1399
1400 JPanel panel = new JPanel();
1401
1402 boolean allOK =true;
1403
1404 public CreateProjectDialog4(int initNumberOfBregmaLevels)
1405 {
1406     this.setTitle("Enter bregma levels");
1407     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1408     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1409     this.setLocation(screenWidth*10/100,screenHeight*10/100);
1410     this.setSize(screenWidth*35/100,screenHeight*80/100);
1411
1412     numberOfBregmaLevels=initNumberOfBregmaLevels;
1413     levels = new double[numberOfBregmaLevels];
1414
1415     bregmaLevelLabel = new JLabel[numberOfBregmaLevels];
1416     bregmaLevelTextFields = new JTextField[numberOfBregmaLevels];
1417     bregmaLevelResultLabel = new JLabel[numberOfBregmaLevels];
1418
1419     statusLabel.setForeground(Color.green);
1420
1421     KeyListener keyListener = new KeyListener()
1422     {
1423         @Override
1424         public void keyPressed(KeyEvent keyEvent)
1425         {
1426             printIt("Pressed", keyEvent);
1427         }
1428         @Override
1429         public void keyReleased(KeyEvent keyEvent)
1430         {
1431             printIt("Released", keyEvent);
1432             createProjectDialog4.readTextFields();
1433         }
1434         @Override
1435         public void keyTyped(KeyEvent keyEvent)
1436         {
1437             printIt("Typed", keyEvent);
1438         }
1439     }

```



```

1439     private void printIt(String title, KeyEvent keyEvent){ }
1440 };
1441
1442 firstLevelTextField.addKeyListener(keyListener);
1443 distanceTextField.addKeyListener(keyListener);
1444
1445 for (int i=0; i<numberOfBregmaLevels; i++)
1446 {
1447     bregmaLevelLabel[i] = new JLabel("Bregma level "+(i+1)+":");
1448     bregmaLevelTextFields[i]=new JTextField(""+i,5);
1449     bregmaLevelTextFields[i].addKeyListener(keyListener);
1450     bregmaLevelResultLabel[i] = new JLabel(""+i);
1451 }
1452
1453 this.setUpWindow();
1454 }
1455
1456 public void setUpWindow()
1457 {
1458     this.getContentPane().removeAll();
1459     panel.removeAll();
1460     this.add(new JScrollPane(panel));
1461     this.setModalityType(ModalityType.APPLICATION_MODAL);
1462
1463     calculateButton=new JButton("Calculate levels");
1464     calculateButton.addActionListener(this);
1465     helpButton=new JButton("Help");
1466     helpButton.addActionListener(this);
1467     previousButton=new JButton("Previous");
1468     previousButton.addActionListener(this);
1469     nextButton=new JButton("Next");
1470     nextButton.addActionListener(this);
1471
1472     panel.setLayout(new GridBagLayout());
1473     GridBagConstraints c = new GridBagConstraints();
1474
1475     c.gridx = 1;
1476     c.gridy = 0;
1477     c.gridwidth = 1;
1478     c.insets = new Insets(30,5,20,5);
1479     c.anchor = GridBagConstraints.PAGE_START;
1480
1481     panel.add(firstInfo, c);
1482
1483     c.gridwidth = 1;
1484     c.insets = new Insets(5,5,5,5);
1485     c.gridx = 0;
1486     c.gridy=1;
1487     c.anchor = GridBagConstraints.FIRST_LINE_END;
1488     panel.add(firstLevelLabel, c);
1489
1490     c.gridx = 1;
1491     c.anchor = GridBagConstraints.PAGE_START;
1492     panel.add(firstLevelTextField, c);
1493
1494     c.gridx = 2;
1495     c.anchor = GridBagConstraints.FIRST_LINE_START;
1496     panel.add(firstLevelResultLabel, c);
1497
1498     c.insets = new Insets(5,5,5,5);

```

```

1499     c.gridx = 0;
1500     c.gridy=2;
1501     c.gridwidth = 1;
1502     c.anchor = GridBagConstraints.FIRST_LINE_END;
1503     panel.add(distanceLabel, c);
1504
1505     c.gridx = 1;
1506     c.anchor = GridBagConstraints.PAGE_START;
1507     panel.add(distanceTextField, c);
1508
1509     c.gridx = 2;
1510     c.anchor = GridBagConstraints.FIRST_LINE_START;
1511     panel.add(distanceResultLabel, c);
1512
1513     c.anchor = GridBagConstraints.PAGE_START;
1514     c.gridx = 1;
1515     c.gridy=3;
1516     c.gridwidth = 1;
1517     panel.add(calculateButton, c);
1518
1519     c.gridx = 1;
1520     c.gridy = 4;
1521     c.gridwidth = 1;
1522     c.insets = new Insets(30,5,5,5);
1523     panel.add(info, c);
1524
1525     c.gridwidth = 1;
1526     c.insets = new Insets(5,5,5,5);
1527     for (int i=0; i<numberOfBregmaLevels; i++)
1528     {
1529         c.gridy = i+5;
1530
1531         c.anchor = GridBagConstraints.FIRST_LINE_END;
1532         c.gridx= 0;
1533         panel.add(bregmaLevelLabel[i], c);
1534
1535         c.anchor = GridBagConstraints.PAGE_START;
1536         c.gridx= 1;
1537         panel.add(bregmaLevelTextFields[i], c);
1538
1539         c.anchor = GridBagConstraints.FIRST_LINE_START;
1540         c.gridx= 2;
1541         panel.add(bregmaLevelResultLabel[i], c);
1542     }
1543
1544     c.gridwidth=1;
1545     c.gridy = numberOfBregmaLevels+6;
1546     c.gridx= 1;
1547     c.anchor = GridBagConstraints.PAGE_START;
1548     c.insets = new Insets(30,5,5,5);
1549     panel.add(statusLabel, c);
1550
1551     c.weighty=1;
1552     c.anchor = GridBagConstraints.FIRST_LINE_END;
1553     c.gridx = 0;
1554     c.weightx=0.3333;
1555     c.gridy = numberOfBregmaLevels+7;
1556     c.gridwidth = 1;
1557     c.insets = new Insets(30,5,5,5);
1558     panel.add(previousButton, c);

```

```

1559
1560     c.anchor = GridBagConstraints.PAGE_START;
1561     c.gridx = 1;
1562     panel.add(helpButton, c);
1563
1564     c.anchor = GridBagConstraints.FIRST_LINE_START;
1565     c.gridx = 2;
1566     panel.add(nextButton, c);
1567
1568     this.readTextFields();
1569
1570     this.repaint();
1571 }
1572
1573 public void readTextFields()
1574 {
1575     calculateButton.setEnabled(true);
1576     result = firstLevelTextField.getText();
1577     try
1578     {
1579         firstLevel= Double.parseDouble(result);
1580         firstLevelResultLabel.setForeground(Color.green);
1581         firstLevelResultLabel.setText(result);
1582     }
1583     catch (NumberFormatException error)
1584     {
1585         firstLevelResultLabel.setForeground(Color.red);
1586         firstLevelResultLabel.setText("Not a number");
1587         calculateButton.setEnabled(false);
1588     }
1589
1590     result = distanceTextField.getText();
1591     try
1592     {
1593         distance= Double.parseDouble(result);
1594         distanceResultLabel.setForeground(Color.green);
1595         distanceResultLabel.setText(result);
1596         if (distance==0)
1597         {
1598             distanceResultLabel.setForeground(Color.red);
1599             distanceResultLabel.setText("Can't be 0");
1600             calculateButton.setEnabled(false);
1601         }
1602     }
1603     catch (NumberFormatException error)
1604     {
1605         distanceResultLabel.setForeground(Color.red);
1606         distanceResultLabel.setText("Not a number");
1607         calculateButton.setEnabled(false);
1608     }
1609
1610     allOK = true;
1611     for (int i=0; i<numberOfBregmaLevels; i++)
1612     {
1613         result = bregmaLevelTextFields[i].getText();
1614         try
1615         {
1616             levels[i] = Double.parseDouble(result);
1617             bregmaLevelResultLabel[i].setForeground(Color.green);
1618             bregmaLevelResultLabel[i].setText(result);

```

```

1619     }
1620 }
1621 catch (NumberFormatException error)
1622 {
1623     bregmaLevelResultLabel[i].setForeground(Color.red);
1624     bregmaLevelResultLabel[i].setText("Not a number");
1625     allOK = false;
1626 }
1627 }
1628
1629 boolean allIncreasing=true;
1630 boolean allDecreasing=true;
1631 if (allOK)
1632 {
1633     for(int i=0;i<numberOfBregmaLevels-1;i++)
1634     {
1635         if (levels[i]>=levels[i+1]) allIncreasing=false;
1636         if (levels[i]<=levels[i+1]) allDecreasing=false;
1637     }
1638 }
1639
1640 if (!allOK)
1641 {
1642     statusLabel.setForeground(Color.red);
1643     statusLabel.setText("A non-number entered");
1644     nextButton.setEnabled(false);
1645 }
1646 else if (!allIncreasing && !allDecreasing)
1647 {
1648     statusLabel.setForeground(Color.red);
1649     statusLabel.setText("Values must continously increase or decrease");
1650     nextButton.setEnabled(false);
1651 }
1652 else
1653 {
1654     statusLabel.setForeground(Color.green);
1655     statusLabel.setText("Ok to proceed");
1656     nextButton.setEnabled(true);
1657 }
1658 }
1659
1660 @Override
1661 public void actionPerformed(ActionEvent e)
1662 {
1663     if (e.getSource()==calculateButton)
1664     {
1665         firstLevel=0;
1666         distance=1;
1667         double newLevel;
1668         double multiplier;
1669         double tempDouble;
1670
1671         try
1672         {
1673             result = firstLevelTextField.getText();
1674             firstLevel = Double.parseDouble(result);
1675             result = distanceTextField.getText();
1676             distance = Double.parseDouble(result);
1677         }
1678         catch (NumberFormatException error) {}

```

```

1679
1680     for (int i=0; i<numberOfBregmaLevels; i++)
1681     {
1682         multiplier = i;
1683         newLevel = firstLevel + multiplier*distance;
1684         tempDouble= (double) (Math.round(newLevel*1000))/1000;
1685         bregmaLevelTextFields[i].setText(""+tempDouble);
1686     }
1687     this.readTextFields();
1688 }
1689
1690 if (e.getSource()==helpButton)
1691 {
1692     String helpText = myMethods.getHelpTextBregmaLevels();
1693     helpDialog.newText(helpText);
1694 }
1695
1696 if (e.getSource()==previousButton)
1697 {
1698     this.setVisible(false);
1699     createProjectDialog3.setVisible(true);
1700 }
1701
1702 if (e.getSource()==nextButton)
1703 {
1704     this.readTextFields();
1705
1706     if (allOK)
1707     {
1708         this.setVisible(false);
1709         createProjectDialog5.setVisible(true);
1710     }
1711 }
1712 }
1713 }
1714
1715 private class CreateProjectDialog5 extends JDialog implements ActionListener
1716 {
1717     //Lets the user select the number of User defined areas, the name of
1718     //each area and wheter it should count tissue, background or both
1719     JLabel firstInfo = new JLabel("Enter Number Of User Defined Areas");
1720
1721     int numberOfUDA=3;
1722     JTextField[] udaTextFields;
1723     ArrayList udaNames = new ArrayList();
1724     ArrayList includeTissue= new ArrayList();
1725     ArrayList includeBackground = new ArrayList();
1726
1727     String[] dialogUdaNames;
1728     boolean[] dialogIncludeBackground;
1729     boolean[] dialogIncludeTissue;
1730
1731     JTextField numberOfUDATextField = new JTextField("3", 5);
1732
1733     JButton helpButton;
1734     JButton increaseButton;
1735     JButton decreaseButton;
1736
1737     JRadioButton[] tissueRadioButton;
1738     JRadioButton[] backgroundRadioButton;

```

```

1739 JRadioButton[] bothRadioButton;
1740 ButtonGroup[] buttonGroup;
1741
1742 JButton createProjectButton;
1743 JButton previousButton;
1744
1745 JPanel panel = new JPanel();
1746
1747 public CreateProjectDialog5()
1748 {
1749     this.setTitle("Enter user defined areas");
1750     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1751     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1752     this.setLocation(screenWidth*10/100,screenHeight*10/100);
1753     this.setSize(screenWidth*35/100,screenHeight*80/100);
1754
1755     numberOfUDATextField.addActionListener(this);
1756
1757     udaTextFields = new JTextField[numberOfUDA];
1758     tissueRadioButton = new JRadioButton[numberOfUDA];
1759     backgroundRadioButton = new JRadioButton[numberOfUDA];
1760     bothRadioButton = new JRadioButton[numberOfUDA];
1761     buttonGroup = new ButtonGroup[numberOfUDA];
1762
1763
1764     this.generateUdaNames();
1765
1766     for (int i=0; i<numberOfUDA; i++)
1767     {
1768         udaTextFields[i]=new JTextField((String) udaNames.get(i),20);
1769     }
1770
1771     this.setUpWindow();
1772 }
1773
1774 public void generateUdaNames () {
1775     String tempString;
1776     while (udaNames.size()<numberOfUDA) {
1777         tempString = "Area " + (udaNames.size()+1);
1778         udaNames.add(tempString);
1779         includeTissue.add(true);
1780         includeBackground.add(true);
1781     }
1782 }
1783
1784 public void readTextFields () {
1785     String tempString;
1786     for (int i=0; i<numberOfUDA; i++) {
1787         tempString = udaTextFields[i].getText();
1788         udaNames.set(i, tempString);
1789
1790         if (tissueRadioButton[i].isSelected()) {
1791             includeTissue.set(i,true);
1792             includeBackground.set(i,false);
1793         }
1794         if (backgroundRadioButton[i].isSelected()) {
1795             includeTissue.set(i,false);
1796             includeBackground.set(i,true);
1797         }
1798         if (bothRadioButton[i].isSelected()) {

```

```

1799         includeTissue.set(i,true);
1800         includeBackground.set(i,true);
1801     }
1802 }
1803 }
1804
1805 public void setUpWindow()
1806 {
1807     this.getContentPane().removeAll();
1808     panel.removeAll();
1809     this.add(new JScrollPane(panel));
1810     this.setModalityType(ModalityType.APPLICATION_MODAL);
1811
1812     helpButton=new JButton("Help");
1813     helpButton.addActionListener(this);
1814     decreaseButton=new JButton("Decrease");
1815     decreaseButton.addActionListener(this);
1816     increaseButton=new JButton("Increase");
1817     increaseButton.addActionListener(this);
1818     createProjectButton=new JButton("Create project");
1819     createProjectButton.addActionListener(this);
1820     previousButton=new JButton("Previous");
1821     previousButton.addActionListener(this);
1822
1823     panel.setLayout(new GridBagLayout());
1824     GridBagConstraints c = new GridBagConstraints();
1825
1826     c.insets = new Insets(30,5,0,5);
1827     c.anchor = GridBagConstraints.PAGE_START;
1828     c.gridwidth = 1;
1829     c.gridx = 1;
1830     c.gridy = 0;
1831     panel.add(firstInfo, c);
1832
1833     c.insets.top=10;
1834     c.gridx = 0;
1835     c.gridy = 1;
1836     panel.add(decreaseButton, c);
1837
1838     numberOfUDATextField.setText(""+numberOfUDA);
1839     c.gridx = 1;
1840     c.gridy = 1;
1841     c.gridwidth = 1;
1842     panel.add(numberOfUDATextField, c);
1843
1844     c.gridx = 2;
1845     c.gridy = 1;
1846     c.gridwidth = 1;
1847
1848     panel.add(increaseButton, c);
1849
1850     c.insets.top=40;
1851
1852     for (int i=0; i<numberOfUDA; i++)
1853     {
1854         c.gridx = 0;
1855         c.gridy = i*2+2;
1856         c.gridwidth = 3;
1857         panel.add(udaTextFields[i], c);
1858     }

```

```

1859     tissueRadioButton[i]= new JRadioButton("Tissue");
1860     backgroundRadioButton[i]= new JRadioButton("Background");
1861     bothRadioButton[i]= new JRadioButton("Both",true);
1862
1863     buttonGroup[i] = new ButtonGroup();
1864
1865     buttonGroup[i].add(tissueRadioButton[i]);
1866     buttonGroup[i].add(backgroundRadioButton[i]);
1867     buttonGroup[i].add(bothRadioButton[i]);
1868
1869     c.gridx = 0;
1870     c.gridy = i*2+3;
1871     c.gridwidth = 1;
1872     c.insets.top=10;
1873     panel.add(tissueRadioButton[i], c);
1874     c.gridx = 1;
1875     panel.add(backgroundRadioButton[i], c);
1876     c.gridx = 2;
1877     panel.add(bothRadioButton[i], c);
1878     c.insets.top=40;
1879
1880     if ((Boolean) includeTissue.get(i) && (Boolean) includeBackground.get(i)) {
1881         bothRadioButton[i].setSelected(true);
1882     }
1883     else if ((Boolean) includeTissue.get(i)) {
1884         tissueRadioButton[i].setSelected(true);
1885     }
1886     else if ((Boolean) includeBackground.get(i)) {
1887         backgroundRadioButton[i].setSelected(true);
1888     }
1889 }
1890
1891 c.weighty=1;
1892 c.gridx = 2;
1893 c.gridy = numberOfUDA*2+2;
1894 c.gridwidth = 1;
1895 c.insets.top=40;
1896 c.insets.bottom=40;
1897
1898 c.gridx = 0;
1899 panel.add(previousButton, c);
1900
1901 c.gridx=1;
1902 panel.add(helpButton, c);
1903
1904 c.gridx=2;
1905 panel.add(createProjectButton, c);
1906 }
1907
1908 @Override
1909 public void actionPerformed(ActionEvent e)
1910 {
1911     if (e.getSource()==helpButton)
1912     {
1913         String helpText = myMethods.getHelpTextUDA();
1914         helpDialog.newText(helpText);
1915     }
1916
1917     if (e.getSource()==numberOfUDATextField)
1918     {

```



```

1919     try
1920     {
1921         readTextFields();
1922
1923         numberOfUDA = Integer.parseInt(numberOfUDATextField.getText());
1924         this.setVisible(false);
1925
1926         udaTextFields = new JTextField[numberOfUDA];
1927         tissueRadioButton = new JRadioButton[numberOfUDA];
1928         backgroundRadioButton = new JRadioButton[numberOfUDA];
1929         bothRadioButton = new JRadioButton[numberOfUDA];
1930         buttonGroup = new ButtonGroup[numberOfUDA];
1931
1932         generateUdaNames();
1933
1934         for (int i=0; i<numberOfUDA; i++)
1935         {
1936             udaTextFields[i]=new JTextField((String) udaNames.get(i),20);
1937         }
1938
1939         this.setUpWindow();
1940         this.setVisible(true);
1941     } catch (Exception error) { }
1942 }
1943
1944 if (e.getSource()==decreaseButton)
1945 {
1946     readTextFields();
1947
1948     if (numberOfUDA>0) numberOfUDA--;
1949     this.setVisible(false);
1950     numberOfUDATextField.setText(""+numberOfUDA);
1951
1952     udaTextFields = new JTextField[numberOfUDA];
1953     tissueRadioButton = new JRadioButton[numberOfUDA];
1954     backgroundRadioButton = new JRadioButton[numberOfUDA];
1955     bothRadioButton = new JRadioButton[numberOfUDA];
1956     buttonGroup = new ButtonGroup[numberOfUDA];
1957
1958     for (int i=0; i<numberOfUDA; i++)
1959     {
1960         udaTextFields[i]=new JTextField((String) udaNames.get(i),20);
1961     }
1962
1963     this.setUpWindow();
1964     this.setVisible(true);
1965 }
1966
1967 if (e.getSource()==increaseButton)
1968 {
1969     readTextFields();
1970
1971     numberOfUDA++;
1972
1973     this.setVisible(false);
1974     numberOfUDATextField.setText(""+numberOfUDA);
1975
1976     udaTextFields = new JTextField[numberOfUDA];
1977     tissueRadioButton = new JRadioButton[numberOfUDA];
1978     backgroundRadioButton = new JRadioButton[numberOfUDA];

```

```

1979     bothRadioButton = new JRadioButton[numberOfUDA];
1980     buttonGroup = new ButtonGroup[numberOfUDA];
1981
1982     generateUdaNames();
1983
1984     for (int i=0; i<numberOfUDA; i++)
1985     {
1986         udaTextFields[i]=new JTextField((String) udaNames.get(i),20);
1987     }
1988
1989     this.setUpWindow();
1990     this.setVisible(true);
1991 }
1992
1993 if (e.getSource()==createProjectButton)
1994 {
1995
1996     dialogUdaNames = new String[numberOfUDA];
1997     dialogIncludeBackground = new boolean[numberOfUDA];
1998     dialogIncludeTissue = new boolean[numberOfUDA];
1999
2000     for (int i=0; i<numberOfUDA; i++)
2001     {
2002         dialogUdaNames[i] = udaTextFields[i].getText();
2003
2004         if (tissueRadioButton[i].isSelected())
2005         {
2006             dialogIncludeTissue[i] = true;
2007             dialogIncludeBackground[i] = false;
2008         }
2009         if (backgroundRadioButton[i].isSelected())
2010         {
2011             dialogIncludeTissue[i] = false;
2012             dialogIncludeBackground[i] = true;
2013         }
2014         if (bothRadioButton[i].isSelected())
2015         {
2016             dialogIncludeTissue[i] = true;
2017             dialogIncludeBackground[i] = true;
2018         }
2019     }
2020
2021     try
2022     {
2023         this.setVisible(false);
2024         myProject.createNewProject() ;
2025     }catch (Exception error) {}
2026
2027     this.setVisible(false);
2028 }
2029
2030 if (e.getSource()==previousButton)
2031 {
2032     this.setVisible(false);
2033     createProjectDialog4.setVisible(true);
2034 }
2035 }
2036 }
2037
2038 private class Project

```

```

2039 {
2040     public void createNewProject()
2041     {
2042         progressMonitor = new ProgressMonitor(null,
2043             "Building Project",
2044             "Sections analyzed: 0", 0, createProjectDialog1.numberOfSections);
2045
2046         progressMonitor.setMillisToDecideToPopup(0);
2047         progressMonitor.setMillisToPopup(0);
2048
2049         BuildProject buildProject = new BuildProject();
2050         buildProject.execute();
2051     }
2052
2053     public class BuildProject extends SwingWorker<Void, Void>
2054     {
2055         //Uses the informaiton from the createProjectDailog frames to build a new project.
2056         @Override
2057         public Void doInBackground()
2058         {
2059             frame.setVisible(false);
2060             int sectionsAnalyzed=0;
2061
2062             myProjectData = new ProjectData(createProjectDialog1.numberOfSubjects,
2063                 createProjectDialog1.highestNumberOfSectionsInSubject,
2064                 createProjectDialog5.numberOfUDA,
2065                 createProjectDialog1.numberOfSections);
2066
2067             treePanel = new TreePanel(createProjectDialog1.numberOfSubjects,
2068                 createProjectDialog1.highestNumberOfSectionsInSubject,
2069                 createProjectDialog5.numberOfUDA);
2070
2071             myProjectData.projectLocation = createProjectDialog2.saveFileLocation.getPath();
2072             myProjectData.imageWidth=createProjectDialog1.detectedImageWidth;
2073             myProjectData.imageHeight=createProjectDialog1.detectedImageHeight;
2074
2075             myProjectData.allImagesOfSameSize = createProjectDialog1.allImagesIsOfSameSize;
2076
2077             for (int i=0; i<myProjectData.numberOfUDA; i++)
2078             {
2079                 myProjectData.UDANames[i]=createProjectDialog5.dialogUdaNames[i];
2080                 myProjectData.includeBackground[i]=createProjectDialog5.dialogIncludeBackground[i];
2081                 myProjectData.includeTissue[i]=createProjectDialog5.dialogIncludeTissue[i];
2082             }
2083
2084             //Saves a copy of the conversion image in the save folder
2085             try
2086             {
2087                 String conversionFileName = myProjectData.projectLocation+"/ConversionImage";
2088                 File conversionFile = new File(conversionFileName);
2089
2090                 BufferedImage tempImage=myMethods.readOriginalImageFile(
2091                     createProjectDialog3.conversionFileLocation);
2092                 ImageIO.write(tempImage, "JPG", conversionFile);
2093
2094                 //ImageIO.write(ImageIO.read(createProjectDialog3.conversionFileLocation),"JPG",conversionFile);
2095
2096
2097

```

```

2098     }catch (IOException event) { }
2099
2100     //Adds subjects to the tree structure
2101     for (int i = 0; i < createProjectDialog1.numberOfSubjects; i++)
2102     {
2103         treePanel.addSubjectToTree(i, createProjectDialog1.subjectNames[i]);
2104         myProjectData.subjectNameList[i]=createProjectDialog1.subjectNames[i];
2105     }
2106
2107     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
2108
2109     for (int subjectCounter=0; subjectCounter<createProjectDialog1.numberOfSubjects;
subjectCounter++)
2110     {
2111         for (int sectionCounter=0;
sectionCounter<createProjectDialog1.highestNumberOfSectionsInSubject; sectionCounter++)
2112         {
2113             if (createProjectDialog1.sectionExists[subjectCounter][sectionCounter])
2114             {
2115                 //Saves the image file in the save folder
2116                 originalImage = myMethods.createErrorImage();
2117
2118                 String originalImageFileName = myMethods.getOriginalImageFileName(subjectCounter,
sectionCounter);
2119                 File saveImageFile= new File(originalImageFileName);
2120
2121                 try
2122                 {
2123                     originalImage=myMethods.readOriginalImageFile(
createProjectDialog1.sectionFiles[subjectCounter][sectionCounter]);
2124                 } catch (Exception error) {}
2125                 try
2126                 {
2127                     ImageIO.write(originalImage, "JPG", saveImageFile);
2128                 } catch (Exception error) {}
2129
2130
2131                 //Analyses the section and adds it to the tree structure
2132                 currentSectionData= new SectionData(myProjectData.imageWidth,
myProjectData.imageHeight);
2133
2134                 myProjectData.positionInProject[subjectCounter][sectionCounter] = sectionsAnalyzed;
2135
2136                 currentSectionData.imageWidth = originalImage.getWidth();
2137                 currentSectionData.imageHeight = originalImage.getHeight();
2138
2139                 currentSectionData.UDApointList = new ArrayList[myProjectData.numberOfUDA];
2140
2141                 for (int UDACounter=0; UDACounter<myProjectData.numberOfUDA;UDACounter++)
2142                 {
2143
2144                     currentSectionData.UDApointList[UDACounter] = new ArrayList();
2145                     myProjectData.numberOfUdaPixels[subjectCounter][sectionCounter][UDACounter] = 0;
2146                 }
2147
2148                 myMethods.saveSectionData(subjectCounter, sectionCounter, currentSectionData);
2149
2150                 treePanel.addSectionToTree(subjectCounter, sectionCounter);
2151                 myProjectData.sectionExists[subjectCounter][sectionCounter] = true;
2152                 myProjectData.sectionName[subjectCounter][sectionCounter] =
createProjectDialog1.sectionNames[subjectCounter][sectionCounter];

```

```

2153         myProjectData.bregmaLevels[subjectCounter][sectionCounter] =
createProjectDialog4.levels[sectionCounter];
2154
2155         boolean[][] tissuePixels = myMethods.getTissuePixels(originalImage);
2156         myProjectData.numberOfTissuePixels[subjectCounter][sectionCounter] =
myMethods.countPixels(tissuePixels);
2157         myProjectData.numberOfTissuePixelsLeft[subjectCounter][sectionCounter] = 0;
2158         myProjectData.numberOfTissuePixelsRight[subjectCounter][sectionCounter] = 0;
2159
2160         boolean[][] ventriclePixels = myMethods.newGetVentriclePixels2(tissuePixels);
2161         myProjectData.numberOfVentriclePixels[subjectCounter][sectionCounter] =
myMethods.countPixels(ventriclePixels);
2162         myProjectData.numberOfVentriclePixelsLeft[subjectCounter][sectionCounter] = 0;
2163         myProjectData.numberOfVentriclePixelsRight[subjectCounter][sectionCounter] = 0;
2164
2165         myMethods.updateSectionResultsObject(subjectCounter, sectionCounter);
2166
2167         sectionsAnalyzed++;
2168         progressMonitor.setNote("Sections analyzed: "+sectionsAnalyzed);
2169         progressMonitor.setProgress(sectionsAnalyzed);
2170     }
2171     if (progressMonitor.isCanceled())
2172     {
2173         break;
2174     }
2175 }
2176 myMethods.analyzeSubject(subjectCounter);
2177
2178 if (progressMonitor.isCanceled())
2179 {
2180     errorDialog.newText(myMethods.getErrorTextCreateProjectInterrupted());
2181     errorDialog.setVisible(true);
2182     break;
2183 }
2184 }
2185
2186 //Sets the zoomFactor to make one image fill the panel
2187 Dimension panelSize = middleScrollPane.getSize();
2188 double widthFactor = 100*panelSize.width/originalImage.getWidth();
2189 double heightFactor = 100*panelSize.height/originalImage.getHeight();
2190 if (widthFactor<heightFactor) myProjectData.zoomFactor = (int) widthFactor;
2191 else myProjectData.zoomFactor = (int) heightFactor;
2192 if (myProjectData.zoomFactor<1) myProjectData.zoomFactor=1;
2193
2194 myMethods.saveProjectData();
2195
2196 leftScrollPane.getViewport().add(treePanel);
2197
2198 treePanel.expandTree();
2199
2200 treePanel.tree.setSelectionRow(4); //Displays the conversion panel
2201
2202 upperScrollPane.requestFocus();
2203
2204 middleScrollPane.repaint();
2205 leftScrollPane.repaint();
2206
2207 frame.setVisible(true);
2208
2209 return null;

```

```

2210     }
2211     @Override
2212     public void done() { }
2213 }
2214
2215 //Triggered when the user opens an existing project
2216 public void refreshWhenOpened()
2217 {
2218     treePanel = new TreePanel(myProjectData.numberOfSubjects,
2219                               myProjectData.numberOfLevels,
2220                               myProjectData.numberOfUDA);
2221
2222     for (int i=0;i<myProjectData.numberOfSubjects;i++)
2223     {
2224         treePanel.addSubjectToTree(i, myProjectData.subjectNameList[i]);
2225     }
2226
2227     for (int i=0;i<myProjectData.numberOfSubjects;i++)
2228     for (int j=0;j<myProjectData.numberOfLevels;j++)
2229     {
2230         if (myProjectData.sectionExists[i][j])
2231         {
2232             treePanel.addSectionToTree(i,j);
2233         }
2234     }
2235     leftScrollPane.getViewport().add(treePanel);
2236
2237     treePanel.expandTree();
2238
2239     treePanel.tree.setSelectionRow(4);
2240
2241     upperScrollPane.requestFocus();
2242 }
2243 }
2244
2245 @SuppressWarnings("LeakingThisInConstructor")
2246 private class TreePanel extends JPanel implements TreeSelectionListener
2247 {
2248     //The tree structure is displayed on the left scroll pane.
2249     //Whenever the user makes a new selection in the tree structure
2250     //a whenLeftInTree is executed for the previous selection and then
2251     //a whenClickedInTree is executed for the new selection.
2252     JTree tree;
2253
2254     DefaultMutableTreeNode currentSelection;
2255     DefaultMutableTreeNode projectLeaf = new DefaultMutableTreeNode("Project");
2256
2257     DefaultMutableTreeNode[] subjectLeafs;
2258     DefaultMutableTreeNode[][] sectionLeafs;
2259     DefaultMutableTreeNode[][] adjustImageLeafs;
2260     DefaultMutableTreeNode[][] setLeftRightBorderLeafs;
2261     DefaultMutableTreeNode[][] udaLeafs;
2262
2263     DefaultMutableTreeNode resultLeaf = new DefaultMutableTreeNode("Results");
2264     DefaultMutableTreeNode subjectResultLeaf = new DefaultMutableTreeNode("Subject Results");
2265     DefaultMutableTreeNode conversionLeaf = new DefaultMutableTreeNode("Conversion");
2266     DefaultMutableTreeNode tissueDetectionLeaf = new DefaultMutableTreeNode("TissueDetection");
2267
2268     public TreePanel(int newNumberOfSubjects, int newNumberOfLevels, int newNumberOfUda)
2269     {

```

```

2270     tree = new JTree(projectLeaf);
2271
2272     subjectLeafs = new DefaultMutableTreeNode[newNumberOfSubjects];
2273     sectionLeafs = new DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2274     adjustImageLeafs = new DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2275     setLeftRightBorderLeafs = new
DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2276
2277     udaLeafs = new DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2278
2279     projectLeaf.add(resultLeaf);
2280     resultLeaf.setUserObject(sectionResults_middlePanel);
2281     projectLeaf.add(subjectResultLeaf);
2282     subjectResultLeaf.setUserObject(subjectResults_middlePanel);
2283     projectLeaf.add(tissueDetectionLeaf);
2284     tissueDetectionLeaf.setUserObject(tissueDetection_middlePanel);
2285     projectLeaf.add(conversionLeaf);
2286     conversionLeaf.setUserObject(conversion_middlePanel);
2287
2288     this.add(tree);
2289     tree.addTreeSelectionListener(this);
2290 }
2291
2292 // This is triggered when the user clicks the TreePanel or press the keys A or D
2293 @Override
2294 public void valueChanged(TreeSelectionEvent event)
2295 {
2296     int previousSubjectNumber=-1;
2297     int newSubjectNumber=-2;
2298     int previousSectionNumber=-1;
2299     int newSectionNumber=-2;
2300
2301     Object previousSelection = conversion_middlePanel;
2302
2303     try {
2304         previousSelection = currentSelection.getUserObject();
2305     } catch (Exception e) { }
2306
2307     Object newSelection = conversion_middlePanel;
2308
2309     currentSelection = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
2310     try {
2311         newSelection = currentSelection.getUserObject();
2312     } catch (Exception e) { }
2313
2314     try
2315     {
2316         AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject) previousSelection;
2317         previousSubjectNumber = currentAdjustImage.subjectNumber;
2318         previousSectionNumber = currentAdjustImage.sectionNumber;
2319     } catch (Exception e) { }
2320
2321     try
2322     {
2323         AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject) newSelection;
2324         newSubjectNumber = currentAdjustImage.subjectNumber;
2325         newSectionNumber = currentAdjustImage.sectionNumber;
2326     } catch (Exception e) { }
2327
2328     try

```



```

2329     {
2330         SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
previousSelection;
2331         previousSubjectNumber = currentSetLeftRightBorder.subjectNumber;
2332         previousSectionNumber = currentSetLeftRightBorder.sectionNumber;
2333     }catch (Exception e) {}
2334
2335     try
2336     {
2337         SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
newSelection;
2338         newSubjectNumber = currentSetLeftRightBorder.subjectNumber;
2339         newSectionNumber = currentSetLeftRightBorder.sectionNumber;
2340     }catch (Exception e) {}
2341
2342     try
2343     {
2344         UdaTreeObject currentUdaTreeObject = (UdaTreeObject) previousSelection;
2345         previousSubjectNumber = currentUdaTreeObject.subjectNumber;
2346         previousSectionNumber = currentUdaTreeObject.sectionNumber;
2347     }catch (Exception e) {}
2348     try
2349     {
2350         UdaTreeObject currentUdaTreeObject = (UdaTreeObject) newSelection;
2351         newSubjectNumber = currentUdaTreeObject.subjectNumber;
2352         newSectionNumber = currentUdaTreeObject.sectionNumber;
2353     }catch (Exception e) {}
2354
2355     if(previousSubjectNumber==newSubjectNumber &&
previousSectionNumber==newSectionNumber) {
2356
2357         try {
2358             AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject) previousSelection;
2359             currentAdjustImage.whenLeftInTreeForSameSection();
2360         }catch (Exception e) {}
2361
2362         try {
2363             SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
previousSelection;
2364             currentSetLeftRightBorder.whenLeftInTreeForSameSection();
2365         }catch (Exception e) {}
2366
2367         try {
2368             UdaTreeObject previousUdaTreeObject = (UdaTreeObject) previousSelection;
2369             previousUdaTreeObject.whenLeftInTreeForSameSection();
2370         }catch (Exception e) {}
2371
2372         //Sets the default cursor in case the cursor was changed in the adjustImagePanel
2373         middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
2374
2375         try {
2376             AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject) newSelection;
2377             currentAdjustImage.whenClickedInTreeFromSameSection();
2378         }catch (Exception e) {}
2379
2380         try {
2381             SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
newSelection;
2382             currentSetLeftRightBorder.whenClickedInTreeFromSameSection();
2383         }catch (Exception e) {}

```



```

2384
2385     try {
2386         UdaTreeObject previousUdaTreeObject = (UdaTreeObject) newSelection;
2387         previousUdaTreeObject.whenClickedFromSameSection();
2388     } catch (Exception e) {}
2389
2390     }
2391     else {
2392
2393         try{
2394             conversion_middlePanel = (Conversion_MiddlePanel) previousSelection;
2395             conversion_middlePanel.whenLeftInTree();
2396         } catch (Exception e) {}
2397
2398         try{
2399             SectionTreeObject currentSection = (SectionTreeObject) previousSelection;
2400             currentSection.whenLeftInTree();
2401         } catch (Exception e) {}
2402
2403         try
2404         {
2405             AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject) previousSelection;
2406             currentAdjustImage.whenLeftInTree();
2407         } catch (Exception e) {}
2408
2409         try
2410         {
2411             SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
previousSelection;
2412             currentSetLeftRightBorder.whenLeftInTree();
2413         } catch (Exception e) {}
2414
2415         try{
2416             UdaTreeObject leftUdaTreeObject = (UdaTreeObject) previousSelection;
2417             leftUdaTreeObject.whenLeftInTree();
2418         } catch (Exception e) {}
2419
2420         try{
2421             subjectResults_middlePanel = (SubjectResults_MiddlePanel) previousSelection;
2422             subjectResults_middlePanel.whenLeftInTree();
2423         } catch (Exception e) {}
2424
2425         try{
2426             sectionResults_middlePanel = (SectionResults_MiddlePanel) previousSelection;
2427             sectionResults_middlePanel.whenLeftInTree();
2428         } catch (Exception e) {}
2429
2430         //Sets the default cursor in case the cursor was changed in the adjustImagePanel
2431         middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
2432
2433
2434         try{
2435             conversion_middlePanel = (Conversion_MiddlePanel) newSelection;
2436             conversion_middlePanel.whenClickedInTree();
2437         } catch (Exception e) {}
2438
2439         try{
2440             SectionTreeObject currentSection = (SectionTreeObject) newSelection;
2441             currentSection.whenClickedInTree();
2442         } catch (Exception e) {}

```

```

2443
2444     try{
2445         SubjectTreeObject currentSubject = (SubjectTreeObject) newSelection;
2446         currentSubject.whenClickedInTree();
2447     }catch (Exception e) {}
2448
2449     try
2450     {
2451         AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject) newSelection;
2452         currentAdjustImage.whenClickedInTree();
2453     }catch (Exception e) {}
2454
2455     try
2456     {
2457         SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
newSelection;
2458         currentSetLeftRightBorder.whenClickedInTree();
2459     }catch (Exception e) {}
2460
2461     try{
2462         UdaTreeObject leftUdaTreeObject = (UdaTreeObject) newSelection;
2463         leftUdaTreeObject.whenClickedInTree();
2464     }catch (Exception e) {}
2465
2466     try{
2467         subjectResults_middlePanel = (SubjectResults_MiddlePanel) newSelection;
2468         subjectResults_middlePanel.whenClickedInTree();
2469     }catch (Exception e) {}
2470
2471     try{
2472         sectionResults_middlePanel = (SectionResults_MiddlePanel) newSelection;
2473         sectionResults_middlePanel.whenClickedInTree();
2474     }catch (Exception e) {}
2475
2476     try{
2477         tissueDetection_middlePanel = (TissueDetection_MiddlePanel) newSelection;
2478         tissueDetection_middlePanel.whenClickedInTree();
2479 }catch (Exception e) {}
2480
2481     }
2482
2483     //Collapses all rows except the selected one
2484     tree.removeTreeSelectionListener(this);
2485     TreePath selectionPath = tree.getSelectionPath();
2486     int rowCount= tree.getRowCount();
2487     for (int i=rowCount; i>-1; i--)
2488     {
2489         try
2490         {
2491             tree.collapseRow(i);
2492         }catch(Exception error){ }
2493     }
2494     tree.makeVisible(selectionPath);
2495     tree.setSelectionPath(selectionPath);
2496     tree.addTreeSelectionListener(this);
2497
2498     upperScrollPane.requestFocus();
2499 }
2500
2501 public void addSubjectToTree (int i, String subject)

```

```

2502     {
2503         subjectLeafs[i] = new DefaultMutableTreeNode();
2504         subjectLeafs[i].setUserObject(new SubjectTreeObject(i));
2505         projectLeaf.add(subjectLeafs[i]);
2506     }
2507 }
2508 public void addSectionToTree (int subjectNumber, int sectionNumber)
2509 {
2510     sectionLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2511     sectionLeafs[subjectNumber][sectionNumber].setUserObject(new
SectionTreeObject(subjectNumber, sectionNumber));
2512     subjectLeafs[subjectNumber].add(sectionLeafs[subjectNumber][sectionNumber]);
2513
2514     adjustImageLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2515     adjustImageLeafs[subjectNumber][sectionNumber].setUserObject(new
AdjustImageTreeObject(subjectNumber, sectionNumber));
2516
sectionLeafs[subjectNumber][sectionNumber].add(adjustImageLeafs[subjectNumber][sectionNumber]);
2517
2518     setLeftRightBorderLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2519     setLeftRightBorderLeafs[subjectNumber][sectionNumber].setUserObject(new
SetLeftRightBorderTreeObject(subjectNumber, sectionNumber));
2520
sectionLeafs[subjectNumber][sectionNumber].add(setLeftRightBorderLeafs[subjectNumber][sectionNumber]);
2521
2522     for (int i=0; i<myProjectData.numberOfUDA;i++)
2523     {
2524         udaLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2525         udaLeafs[subjectNumber][sectionNumber].setUserObject(new UdaTreeObject(subjectNumber,
sectionNumber, i));
2526         sectionLeafs[subjectNumber][sectionNumber].add(udaLeafs[subjectNumber][sectionNumber]);
2527     }
2528 }
2529 }
2530 public void expandTree()
2531 {
2532     tree.expandRow(0);
2533 }
2534 }
2535
2536 //The different TreeObjects are attached to the tree structure.
2537 //They rely info about which section was activated in the tree
2538 //to the relevant panel object.
2539 private class SubjectTreeObject
2540 {
2541     int subjectNumber;
2542
2543     public SubjectTreeObject (int subject)
2544     {
2545         subjectNumber=subject;
2546     }
2547
2548     public void whenClickedInTree()
2549     {
2550         subject_middlePanel.whenClickedInTree(subjectNumber);
2551     }
2552
2553     @Override
2554     public String toString()
2555     {

```

```

2556     return myProjectData.subjectNameList[subjectNumber];
2557 }
2558 }
2559
2560 private class SectionTreeObject
2561 {
2562     int subjectNumber;
2563     int sectionNumber;
2564
2565     public SectionTreeObject (int subject, int section)
2566     {
2567         subjectNumber=subject;
2568         sectionNumber=section;
2569     }
2570
2571     public void whenClickedInTree()
2572     {
2573         section_middlePanel.whenClickedInTree(subjectNumber, sectionNumber);
2574     }
2575
2576     public void whenLeftInTree()
2577     {
2578         section_middlePanel.whenLeftInTree(subjectNumber, sectionNumber);
2579     }
2580
2581     @Override
2582     public String toString()
2583     {
2584         return myProjectData.sectionName[subjectNumber][sectionNumber];
2585     }
2586 }
2587
2588 public class AdjustImageTreeObject
2589 {
2590     int subjectNumber;
2591     int sectionNumber;
2592
2593     public AdjustImageTreeObject(int initSubjectNumber, int initSectionNumber)
2594     {
2595         subjectNumber=initSubjectNumber;
2596         sectionNumber=initSectionNumber;
2597     }
2598
2599     public void whenClickedInTree()
2600     {
2601         adjustImage_middlePanel.whenClickedInTree(subjectNumber, sectionNumber);
2602     }
2603     public void whenClickedInTreeFromSameSection()
2604     {
2605         adjustImage_middlePanel.whenClickedInTreeFromSameSection(subjectNumber, sectionNumber);
2606     }
2607
2608     public void whenLeftInTree() {
2609         adjustImage_middlePanel.whenLeftInTree();
2610     }
2611
2612     public void whenLeftInTreeForSameSection() {
2613         adjustImage_middlePanel.whenLeftInTreeForSameSection();
2614     }
2615

```

```

2616     @Override
2617     public String toString()
2618     {
2619         String temp;
2620         temp="Adjust image";
2621         return temp;
2622     }
2623 }
2624
2625 public class SetLeftRightBorderTreeObject
2626 {
2627     int subjectNumber;
2628     int sectionNumber;
2629
2630     public SetLeftRightBorderTreeObject (int initSubjectNumber, int initSectionNumber)
2631     {
2632         subjectNumber=initSubjectNumber;
2633         sectionNumber=initSectionNumber;
2634     }
2635
2636     public void whenClickedInTree()
2637     {
2638         lr_divideMiddlePanel.whenClickedInTree(subjectNumber, sectionNumber);
2639     }
2640     public void whenClickedInTreeFromSameSection()
2641     {
2642         lr_divideMiddlePanel.whenClickedInTreeFromSameSection(subjectNumber, sectionNumber);
2643     }
2644
2645     public void whenLeftInTree()
2646     {
2647         lr_divideMiddlePanel.whenLeftInTree();
2648     }
2649     public void whenLeftInTreeForSameSection()
2650     {
2651         lr_divideMiddlePanel.whenLeftInTreeForSameSection();
2652     }
2653
2654     @Override
2655     public String toString()
2656     {
2657         String temp;
2658         temp="Set Left-Right Border";
2659         return temp;
2660     }
2661 }
2662
2663 public class UdaTreeObject
2664 {
2665     int subjectNumber;
2666     int sectionNumber;
2667     int UDAnumber;
2668
2669     public UdaTreeObject(int initSubject, int initSection, int initUDAnumber)
2670     {
2671         subjectNumber=initSubject;
2672         sectionNumber=initSection;
2673         UDAnumber=initUDAnumber;
2674     }
2675

```

```

2676     @Override
2677     public String toString()
2678     {
2679         String temp;
2680         temp=myProjectData.UDAnames[UDANumber];
2681         return temp;
2682     }
2683
2684     public void whenClickedInTree()
2685     {
2686         uda_middlePanel.whenClickedInTree(subjectNumber, sectionNumber, UDANumber);
2687     }
2688
2689     public void whenClickedFromSameSection() {
2690         uda_middlePanel.whenClickedFromSameSection(subjectNumber, sectionNumber, UDANumber);
2691     }
2692
2693
2694     public void whenLeftInTree()
2695     {
2696         uda_middlePanel.whenLeftInTree();
2697     }
2698
2699     public void whenLeftInTreeForSameSection() {
2700         uda_middlePanel.whenLeftInTreeForSameSection();
2701     }
2702 }
2703
2704 private class SectionResults_MiddlePanel extends JPanel
2705 {
2706     //Creates the table with the results for all sections
2707     JTable table = new JTable();
2708     String[] columnNames;
2709     String resultsForClipboard;
2710
2711     public SectionResults_MiddlePanel()
2712     {
2713         table = new JTable();
2714     }
2715
2716     public void updateTable()
2717     {
2718         sectionResults_middlePanel.remove(table);
2719
2720         int numberOfUDA= myProjectData.numberOfUDA;
2721
2722         columnNames= new String[15+2*numberOfUDA];
2723
2724         columnNames[0]= "Subject";
2725         columnNames[1]= "Section";
2726         columnNames[2]= "Bregma";
2727         columnNames[3]= "Tissue Pixels";
2728         columnNames[4]= "Tissue mm2";
2729         columnNames[5]= "Left Tissue Pixels";
2730         columnNames[6]= "Left Tissue mm2";
2731         columnNames[7]= "Right Tissue Pixels";
2732         columnNames[8]= "Right Tissue mm2";
2733         columnNames[9]= "Ventricle pixels";
2734         columnNames[10]= "Ventricle mm2";
2735         columnNames[11]= "Left Ventricle Pixels";

```

```

2736     columnNames[12]= "Left Ventricle mm2";
2737     columnNames[13]= "Right Ventricle Pixels";
2738     columnNames[14]= "Right Ventricle mm2";
2739
2740     for (int i=0; i<numberOfUDA;i++) {
2741         columnNames[15+2*i]= myProjectData.UDAnames[i] + " Pixels";
2742         columnNames[16+2*i]= myProjectData.UDAnames[i] + " mm2";
2743     }
2744
2745     this.removeAll();
2746
2747     table = new JTable(myProjectData.roundedSectionResultsObject, columnNames);
2748     table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
2749
2750     TableColumn column;
2751     for (int i=0; i<15+2*numberOfUDA ; i++)
2752     {
2753         column = table.getColumnModel().getColumn(i);
2754         column.setPreferredWidth(150);
2755     }
2756
2757     JScrollPane scrollPane = new JScrollPane(table);
2758
2759     firstSplitPane.setLeftComponent(scrollPane);
2760     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
2761     firstSplitPane.setDividerLocation(screenWidth*14/20);
2762 }
2763
2764 public void createClipboardString()
2765 {
2766     String string = "";
2767     String temp;
2768
2769     int numberOfUDA= myProjectData.numberOfUDA;
2770
2771     for (int i=0;i<15+2*numberOfUDA;i++)
2772     {
2773         try
2774         {
2775             temp=columnNames[i];
2776             string= string + temp;
2777             if (i!=15+2*numberOfUDA-1)
2778                 string = string + '\t';
2779
2780         }catch(Exception e){}
2781     }
2782
2783     for (int i=0;i<myProjectData.totalNumberOfSections;i++)
2784     {
2785         string=string+'\n';
2786         for(int j=0;j<15+2*numberOfUDA;j++)
2787         {
2788             try
2789             {
2790                 temp=myProjectData.sectionResultsObject[i][j].toString();
2791
2792                 string= string + temp;
2793                 if (j!=15+2*numberOfUDA-1)
2794                     string = string + '\t';
2795             }catch(Exception e){}

```

```

2796     }
2797     }
2798     resultsForClipboard = string;
2799 }
2800
2801 public void copyToClipboard()
2802 {
2803     this.createClipboardString();
2804
2805     StringSelection stringToClipboard = new StringSelection(resultsForClipboard);
2806     Toolkit toolkit = Toolkit.getDefaultToolkit();
2807     Clipboard cp = toolkit.getSystemClipboard();
2808     cp.setContents(stringToClipboard, null);
2809 }
2810
2811 @Override
2812 public String toString()
2813 {
2814     return "Section Results";
2815 }
2816
2817 public void whenClickedInTree()
2818 {
2819     myMethods.setWaitCursor();
2820
2821     this.updateTable();
2822     middleScrollPane.getViewport().add(sectionResults_middlePanel);
2823     rightScrollPane.getViewport().add(sectionResults_rightSidePanel);
2824
2825     if(myProjectData.conversionIsDefined)
2826     {
2827         sectionResults_rightSidePanel.conversionWarning.setForeground(Color.green);
2828         sectionResults_rightSidePanel.conversionWarning.setText("Conversion defined");
2829     }
2830     else
2831     {
2832         sectionResults_rightSidePanel.conversionWarning.setForeground(Color.red);
2833         sectionResults_rightSidePanel.conversionWarning.setText("Conversion not defined");
2834     }
2835
2836     if(myProjectData.allLevelsContinuous)
2837     {
2838         sectionResults_rightSidePanel.continuousLevelsWarning.setForeground(Color.green);
2839         sectionResults_rightSidePanel.continuousLevelsWarning.setText("All levels continuous");
2840     }
2841     else
2842     {
2843         sectionResults_rightSidePanel.continuousLevelsWarning.setForeground(Color.red);
2844         sectionResults_rightSidePanel.continuousLevelsWarning.setText("Levels have to be continuous");
2845     }
2846
2847     if(myProjectData.allImagesOfSameSize) {
2848         sectionResults_rightSidePanel.sameSizeWarning.setForeground(Color.green);
2849         sectionResults_rightSidePanel.sameSizeWarning.setText("All images of same size");
2850     }
2851     else {
2852         sectionResults_rightSidePanel.sameSizeWarning.setForeground(Color.orange);
2853         sectionResults_rightSidePanel.sameSizeWarning.setText("Warning: All images are not of the
2854 same size");
2855     }

```



```

2855
2856     myMethods.setCustomCursor();
2857 }
2858
2859 public void whenLeftInTree()
2860 {
2861     firstSplitPane.setLeftComponent(middleScrollPane);
2862     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
2863     firstSplitPane.setDividerLocation(screenWidth*14/20);
2864 }
2865 }
2866
2867 @SuppressWarnings("LeakingThisInConstructor")
2868 public class SectionResults_RightSidePanel extends JPanel implements ActionListener
2869 {
2870     JButton copyToClipboardButton;
2871
2872     JLabel conversionWarning = new JLabel("Conversion not updated");
2873     JLabel continousLevelsWarning = new JLabel("All levels continous");
2874     JLabel sameSizeWarning = new JLabel("");
2875
2876     public SectionResults_RightSidePanel()
2877     {
2878         copyToClipboardButton=new JButton("Copy to clipboard");
2879
2880         conversionWarning.setForeground(Color.red);
2881         sameSizeWarning.setForeground(Color.orange);
2882
2883         this.setLayout(new GridBagLayout());
2884         GridBagConstraints c = new GridBagConstraints();
2885
2886         c.gridx = 0;
2887         c.gridy = 0;
2888         c.gridwidth = 1;
2889         c.insets = new Insets(10,5,5,5);
2890         c.anchor = GridBagConstraints.PAGE_START;
2891         this.add(copyToClipboardButton, c);
2892
2893         c.insets = new Insets(30,5,5,5);
2894         c.gridy++;
2895         this.add(conversionWarning, c);
2896
2897         c.insets = new Insets(30,5,5,5);
2898         c.gridy++;
2899         this.add(continousLevelsWarning, c);
2900
2901         c.insets = new Insets(30,5,5,5);
2902         c.gridy++;
2903         c.weighty=1;
2904         this.add(sameSizeWarning, c);
2905
2906         copyToClipboardButton.addActionListener(this);
2907     }
2908
2909     @Override
2910     public void actionPerformed(ActionEvent e)
2911     {
2912         if (e.getSource() == copyToClipboardButton)
2913         {
2914             sectionResults_middlePanel.copyToClipboard();

```

```

2915     }
2916 }
2917 }
2918
2919 private class SubjectResults_MiddlePanel extends JPanel
2920 {
2921     //Holds the table with the results for each subject
2922     JTable table = new JTable();
2923     Object[][] data;
2924     Object[][] tableData;
2925     String[] columnNames;
2926
2927     public SubjectResults_MiddlePanel()
2928     {
2929         table = new JTable();
2930     }
2931
2932     public void updateTable()
2933     {
2934         subjectResults_middlePanel.remove(table);
2935         data = new Object[myProjectData.numberOfSubjects][7+myProjectData.numberOfUDA];
2936         tableData = new Object[myProjectData.numberOfSubjects][7+myProjectData.numberOfUDA];
2937         columnNames= new String[7+myProjectData.numberOfUDA];
2938
2939         columnNames[0] = "Subject";
2940         columnNames[1] = "Tissue Volume";
2941         columnNames[2] = "Tissue Volume Left";
2942         columnNames[3] = "Tissue Volume Right";
2943         columnNames[4] = "Ventricle Volume";
2944         columnNames[5] = "Ventricle Volume Left";
2945         columnNames[6] = "Ventricle Volume Right";
2946
2947         for (int udaCounter=0;udaCounter<myProjectData.numberOfUDA;udaCounter++)
2948         {
2949             columnNames[7+udaCounter] = myProjectData.UDAnames[udaCounter];
2950         }
2951
2952         double tempDouble;
2953         int counter=0;
2954         for (int i=0; i<myProjectData.numberOfSubjects; i++)
2955         {
2956             if (myProjectData.sectionExists[i][0])
2957             {
2958                 data[counter][0] = myProjectData.subjectNameList[i];
2959                 tableData[counter][0] = myProjectData.subjectNameList[i];
2960
2961                 tempDouble = myProjectData.tissueVolume[i];
2962                 data[counter][1] = tempDouble;
2963                 tempDouble= (double) (Math.round(tempDouble*100))/100;
2964                 tableData[counter][1] = tempDouble;
2965
2966                 tempDouble = myProjectData.tissueVolumeLeft[i];
2967                 data[counter][2] = tempDouble;
2968                 tempDouble= (double) (Math.round(tempDouble*100))/100;
2969                 tableData[counter][2] = tempDouble;
2970
2971                 tempDouble = myProjectData.tissueVolumeRight[i];
2972                 data[counter][3] = tempDouble;
2973                 tempDouble= (double) (Math.round(tempDouble*100))/100;
2974                 tableData[counter][3] = tempDouble;

```

```

2975
2976     tempDouble = myProjectData.ventricleVolume[i];
2977     data[counter][4] = tempDouble;
2978     tempDouble= (double) (Math.round(tempDouble*100))/100;
2979     tableData[counter][4] = tempDouble;
2980
2981     tempDouble = myProjectData.ventricleVolumeLeft[i];
2982     data[counter][5] = tempDouble;
2983     tempDouble= (double) (Math.round(tempDouble*100))/100;
2984     tableData[counter][5] = tempDouble;
2985
2986     tempDouble = myProjectData.ventricleVolumeRight[i];
2987     data[counter][6] = tempDouble;
2988     tempDouble= (double) (Math.round(tempDouble*100))/100;
2989     tableData[counter][6] = tempDouble;
2990
2991     for (int udaCounter=0;udaCounter<myProjectData.numberOfUDA;udaCounter++)
2992     {
2993         tempDouble = myProjectData.udaVolume[i][udaCounter];
2994         data[counter][7+udaCounter] = tempDouble;
2995         tempDouble= (double) (Math.round(tempDouble*100))/100;
2996         tableData[counter][7+udaCounter] = tempDouble;
2997     }
2998     counter++;
2999 }
3000 }
3001
3002 this.removeAll();
3003 table = new JTable(tableData, columnNames);
3004 table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
3005
3006 TableColumn column;
3007 for (int i=0; i<7+myProjectData.numberOfUDA ; i++)
3008 {
3009     column = table.getColumnModel().getColumn(i);
3010     column.setPreferredWidth(150);
3011 }
3012
3013 JScrollPane scrollPane = new JScrollPane(table);
3014
3015 firstSplitPane.setLeftComponent(scrollPane);
3016 int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
3017 firstSplitPane.setDividerLocation(screenWidth*14/20);
3018 middleScrollPane.repaint();
3019 }
3020
3021 public void whenClickedInTree()
3022 {
3023     myMethods.setWaitCursor();
3024
3025     this.updateTable();
3026     middleScrollPane.getViewport().add(this);
3027     rightScrollPane.getViewport().add(subjectResults_rightSidePanel);
3028
3029     if(myProjectData.conversionIsDefined)
3030     {
3031         subjectResults_rightSidePanel.conversionWarning.setForeground(Color.green);
3032         subjectResults_rightSidePanel.conversionWarning.setText("Conversion defined");
3033     }
3034     else

```

```

3035     {
3036         subjectResults_rightSidePanel.conversionWarning.setForeground(Color.red);
3037         subjectResults_rightSidePanel.conversionWarning.setText("Conversion not defined");
3038     }
3039
3040     if(myProjectData.allLevelsContinuous)
3041     {
3042         subjectResults_rightSidePanel.continuousLevelsWarning.setForeground(Color.green);
3043         subjectResults_rightSidePanel.continuousLevelsWarning.setText("All levels continuous");
3044     }
3045     else
3046     {
3047         subjectResults_rightSidePanel.continuousLevelsWarning.setForeground(Color.red);
3048         subjectResults_rightSidePanel.continuousLevelsWarning.setText("Levels have to be continuous");
3049     }
3050
3051     if(myProjectData.allImagesOfSameSize)
3052     {
3053         subjectResults_rightSidePanel.sameSizeWarning.setForeground(Color.green);
3054         subjectResults_rightSidePanel.sameSizeWarning.setText("All images of same size");
3055     }
3056     else
3057     {
3058         subjectResults_rightSidePanel.sameSizeWarning.setForeground(Color.orange);
3059         subjectResults_rightSidePanel.sameSizeWarning.setText("Warning: All images are not of same
size");
3060     }
3061
3062     myMethods.setCustomCursor();
3063 }
3064
3065 public void whenLeftInTree()
3066 {
3067     firstSplitPane.setLeftComponent(middleScrollPane);
3068     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
3069     firstSplitPane.setDividerLocation(screenWidth*14/20);
3070 }
3071
3072 public void copyToClipboard()
3073 {
3074     String string = "";
3075     String temp;
3076
3077     for (int i=0;i<7+myProjectData.numberOfUDA;i++)
3078     {
3079         try
3080         {
3081             temp=columnNames[i];
3082             string= string + temp;
3083             if (i!=7+myProjectData.numberOfUDA-1)
3084                 string = string + '\t';
3085         }catch(Exception e){}
3086     }
3087
3088     for (int i=0;i<myProjectData.numberOfSubjects;i++)
3089     {
3090         string=string+'\n';
3091         for(int j=0;j<7+myProjectData.numberOfUDA;j++)
3092         {
3093             try

```

```

3094     {
3095         temp=data[i][j].toString();
3096         string= string + temp;
3097         if (j!=7+myProjectData.numberOfUDA-1)
3098             string = string + '\t';
3099     }catch(Exception e){}
3100     }
3101 }
3102
3103 StringSelection stringToClipboard = new StringSelection(string);
3104 Toolkit toolkit = Toolkit.getDefaultToolkit();
3105 Clipboard cp = toolkit.getSystemClipboard();
3106 cp.setContents(stringToClipboard, null);
3107 }
3108
3109 @Override
3110 public String toString()
3111 {
3112     return "Subject Results";
3113 }
3114 }
3115
3116 @SuppressWarnings("LeakingThisInConstructor")
3117 public class SubjectResults_RightSidePanel extends JPanel implements ActionListener
3118 {
3119     JButton copyToClipboardButton;
3120
3121     JLabel conversionWarning = new JLabel("Conversion not updated");
3122     JLabel continousLevelsWarning = new JLabel("All levels continous");
3123     JLabel sameSizeWarning = new JLabel("");
3124
3125     public SubjectResults_RightSidePanel()
3126     {
3127         copyToClipboardButton=new JButton("Copy to clipboard");
3128
3129         conversionWarning.setForeground(Color.red);
3130
3131         this.setLayout(new GridBagLayout());
3132         GridBagConstraints c = new GridBagConstraints();
3133
3134         c.gridx = 0;
3135         c.gridy = 0;
3136         c.gridwidth = 1;
3137         c.insets = new Insets(10,5,5,5);
3138         c.anchor = GridBagConstraints.PAGE_START;
3139         this.add(copyToClipboardButton, c);
3140
3141         c.insets = new Insets(30,5,5,5);
3142         c.gridy++;
3143         this.add(conversionWarning, c);
3144
3145         c.insets = new Insets(30,5,5,5);
3146         c.gridy++;
3147         this.add(continousLevelsWarning, c);
3148
3149         c.insets = new Insets(30,5,5,5);
3150         c.gridy++;
3151         c.weighty=1;
3152         this.add(sameSizeWarning, c);
3153

```

```

3154     copyToClipboardButton.addActionListener(this);
3155 }
3156
3157 @Override
3158 public void actionPerformed(ActionEvent e)
3159 {
3160     if (e.getSource() == copyToClipboardButton)
3161     {
3162         subjectResults_middlePanel.copyToClipboard();
3163     }
3164 }
3165 }
3166
3167 @SuppressWarnings("LeakingThisInConstructor")
3168 public class TissueDetection_MiddlePanel extends JPanel implements MouseMotionListener,
MouseListener
3169 {
3170     //Lets the user find suitable thresholds for tissue detection. Moving the mouse over the image
3171     //displays RGB values and whether the pixels is currently interpreted as tissue or background.
3172     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
3173     BufferedImage displayImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
3174
3175     int displaySubject = 0;
3176     int displaySection = 0;
3177
3178     public TissueDetection_MiddlePanel()
3179     {
3180         addMouseMotionListener(this);
3181         addMouseListener(this);
3182     }
3183
3184     public void whenClickedInTree()
3185     {
3186         myMethods.setWaitCursor();
3187
3188         currentSectionData=new SectionData(myProjectData.imageWidth, myProjectData.imageHeight);
3189         currentSectionData.colorThreshold=myProjectData.colorThresholdGeneral;
3190         currentSectionData.intensityThreshold=myProjectData.intensityThresholdGeneral;
3191
3192         tissueDetection_rightSidePanel.colorThresholdTextField.setText(""+myProjectData.colorThresholdGeneral);
3193         tissueDetection_rightSidePanel.intensityThresholdTextField.setText(""+myProjectData.intensityThresholdGeneral);
3194
3195         tissueDetection_rightSidePanel.colorWarningLabel.setText(""+myProjectData.colorThresholdGeneral);
3196         tissueDetection_rightSidePanel.colorWarningLabel.setForeground(Color.green);
3197         tissueDetection_rightSidePanel.intensityWarningLabel.setText(""+myProjectData.intensityThresholdGeneral);
3198         tissueDetection_rightSidePanel.intensityWarningLabel.setForeground(Color.green);
3199         tissueDetection_rightSidePanel.colorThresholdOk=true;
3200         tissueDetection_rightSidePanel.intensityThresholdOk=true;
3201         tissueDetection_rightSidePanel.applyToAllButton.setEnabled(true);
3202
3203         tissueDetection_middlePanel.updateDisplayImage();
3204         tissueDetection_middlePanel.setPreferredSize(new Dimension(displayImage.getWidth(),
displayImage.getHeight()));
3205         middleScrollPane.getViewPort().add(tissueDetection_middlePanel);
3206         rightScrollPane.getViewPort().add(tissueDetection_rightSidePanel);

```

```

3207
tissueDetection_rightSidePanel.displayedImageLabel.setText(myProjectData.sectionName[displaySubject][displaySection]);
3208
3209     myMethods.setCustomCursor();
3210 }
3211
3212 public void updateDisplayImage()
3213 {
3214     int[] tissueColor = {255,0,122};
3215     try
3216     {
3217         File originalImageFile= new File(myMethods.getOriginalImageFileName(displaySubject,
displaySection));
3218         originalImage = ImageIO.read(originalImageFile);
3219         displayImage = ImageIO.read(originalImageFile);
3220     } catch (IOException event) { }
3221     boolean[][] tissue = myMethods.getTissuePixels(displayImage);
3222     myMethods.changePixelsInImage(displayImage, tissue, tissueColor);
3223
3224
tissueDetection_rightSidePanel.displayedImageLabel.setText(myProjectData.sectionName[displaySubject][displaySection]);
3225     }
3226
3227     @Override
3228     public void mouseMoved(MouseEvent event)
3229     {
3230         Point currentPoint = event.getPoint();
3231         int[] color = new int[3];
3232
3233         currentPoint.x=100*currentPoint.x/myProjectData.zoomFactor;
3234         currentPoint.y=100*currentPoint.y/myProjectData.zoomFactor;
3235
3236         if (currentPoint.x<originalImage.getWidth()
3237             && currentPoint.y<originalImage.getHeight())
3238         {
3239             //Mouse is over the top image
3240             WritableRaster raster = originalImage.getRaster();
3241
3242             color=raster.getPixel(currentPoint.x,currentPoint.y,color);
3243
3244             tissueDetection_rightSidePanel.redValue.setText("Red: "+color[0]);
3245             tissueDetection_rightSidePanel.greenValue.setText("Green: "+color[1]);
3246             tissueDetection_rightSidePanel.blueValue.setText("Blue: "+color[2]);
3247
3248             int firstInt = color[0]+color[2];
3249             int secondInt = color[0]+color[1]+color[2];
3250             double tempDouble = (double) firstInt/secondInt;
3251             tempDouble = tempDouble*1000;
3252             tempDouble = Math.round(tempDouble);
3253             tempDouble = tempDouble/10;
3254
3255             tissueDetection_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
3256             tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
3257             if(myMethods.pixelIsTissue(color))
3258             {
3259                 tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
3260             }
3261             else

```



```

3262     {
3263         tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Background");
3264     }
3265
3266     tissueDetection_rightSidePanel.repaint();
3267 }
3268 else if (currentPoint.x<originalImage.getWidth()
3269         && currentPoint.y<2*originalImage.getHeight()
3270         && currentPoint.y>originalImage.getHeight())
3271 {
3272     //Mouse is over the lower image
3273     currentPoint.y=currentPoint.y-originalImage.getHeight();
3274
3275     WritableRaster raster = originalImage.getRaster();
3276
3277     color=raster.getPixel(currentPoint.x,currentPoint.y,color);
3278
3279     tissueDetection_rightSidePanel.redValue.setText("Red: "+color[0]);
3280     tissueDetection_rightSidePanel.greenValue.setText("Green: "+color[1]);
3281     tissueDetection_rightSidePanel.blueValue.setText("Blue: "+color[2]);
3282
3283     int firstInt = color[0]+color[2];
3284     int secondInt = color[0]+color[1]+color[2];
3285     double tempDouble = (double) firstInt/secondInt;
3286     tempDouble = tempDouble*1000;
3287     tempDouble = Math.round(tempDouble);
3288     tempDouble = tempDouble/10;
3289
3290     tissueDetection_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
3291     tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
3292     if(myMethods.pixellIsTissue(color))
3293     {
3294         tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
3295     }
3296     else
3297     {
3298         tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Background");
3299     }
3300
3301     tissueDetection_rightSidePanel.repaint();
3302 }
3303
3304 else
3305 {
3306     //Mouse is over middle panel but not over one of the panels
3307     tissueDetection_rightSidePanel.redValue.setText("Red: ---");
3308     tissueDetection_rightSidePanel.greenValue.setText("Green: ---");
3309     tissueDetection_rightSidePanel.blueValue.setText("Blue: ---");
3310     tissueDetection_rightSidePanel.colorValueLabel.setText("Color: ---");
3311     tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
3312     tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: ---");
3313
3314     tissueDetection_rightSidePanel.repaint();
3315 }
3316 }
3317
3318 @Override
3319 public void mouseExited(MouseEvent e)
3320 {
3321     tissueDetection_rightSidePanel.redValue.setText("Red: ---");

```



```

3322     tissueDetection_rightSidePanel.greenValue.setText("Green: ---");
3323     tissueDetection_rightSidePanel.blueValue.setText("Blue: ---");
3324     tissueDetection_rightSidePanel.colorValueLabel.setText("Color: ---");
3325     tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
3326     tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: ---");
3327
3328     tissueDetection_rightSidePanel.repaint();
3329 }
3330
3331 @Override
3332 public void mouseClicked(MouseEvent e) {}
3333 @Override
3334 public void mousePressed(MouseEvent e) {}
3335 @Override
3336 public void mouseReleased(MouseEvent event) {}
3337 @Override
3338 public void mouseDragged(MouseEvent e) {}
3339 @Override
3340 public void mouseEntered(MouseEvent e) {}
3341
3342 @Override
3343 public void paint(Graphics g)
3344 {
3345     super.paint(g);
3346
3347     int width = displayImage.getWidth();
3348     int height = displayImage.getHeight();
3349
3350     int drawWidth = width*myProjectData.zoomFactor/100;
3351     int drawHeight = height*myProjectData.zoomFactor/100;
3352
3353     tissueDetection_middlePanel.setPreferredSize(new Dimension(drawWidth+300, 2*drawHeight));
3354
3355     g.drawString("Original image", drawWidth+10, 15);
3356     g.drawImage(originalImage,0,0, drawWidth, drawHeight,this);
3357
3358     g.drawString("Detected tissue", drawWidth+10, drawHeight+15);
3359     g.drawImage(displayImage, 0,drawHeight, drawWidth, drawHeight, this);
3360
3361     this.revalidate();
3362 }
3363 @Override
3364 public String toString()
3365 {
3366     //return new String("Tissue Detection");
3367     return "Tissue Detection";
3368 }
3369 }
3370
3371 @SuppressWarnings("LeakingThisInConstructor")
3372 public class TissueDetection_RightSidePanel extends JPanel implements ActionListener
3373 {
3374     JButton helpButton;
3375
3376     JLabel colorTresholdLabel = new JLabel("Color treshold");
3377     JTextField colorTresholdTextField = new JTextField(""+myProjectData.colorTresholdGeneral, 8);
3378     JLabel colorWarningLabel = new JLabel(""+myProjectData.colorTresholdGeneral);
3379     JLabel intensityTresholdLabel = new JLabel("Intensity treshold");
3380     JTextField intensityTresholdTextField = new JTextField(""+myProjectData.intensityTresholdGeneral,
8);

```

```

3381 JLabel intensityWarningLabel = new JLabel(""+myProjectData.intensityTresholdGeneral);
3382
3383 JLabel redValue = new JLabel("Red: ---");
3384 JLabel greenValue = new JLabel("Green: ---");
3385 JLabel blueValue = new JLabel("Blue: ---");
3386
3387 JLabel colorValueLabel = new JLabel("Color: ");
3388 JLabel intensityValueLabel = new JLabel("Intensity: ");
3389 JLabel resultLabel = new JLabel("Pixel is: ");
3390
3391 JButton applyToAllButton;
3392 JButton previousButton;
3393 JLabel displayedImageLabel = new JLabel("---");
3394 JButton nextButton;
3395
3396 boolean colorTresholdOk=true;
3397 boolean intensityTresholdOk=true;
3398
3399 public TissueDetection_RightSidePanel()
3400 {
3401     helpButton=new JButton("Help");
3402     applyToAllButton=new JButton("Apply To All");
3403     previousButton=new JButton("Previous image");
3404     nextButton=new JButton("Next image");
3405
3406     this.setLayout(new GridBagLayout());
3407     GridBagConstraints c = new GridBagConstraints();
3408
3409     c.gridx = 0;
3410     c.gridy = 0;
3411     c.gridwidth = 1;
3412     c.insets = new Insets(15,5,25,5);
3413     c.anchor = GridBagConstraints.PAGE_START;
3414     this.add(helpButton, c);
3415
3416     c.gridy++;
3417     c.insets = new Insets(5,5,5,5);
3418     this.add(colorTresholdLabel, c);
3419
3420     c.gridy++;
3421     this.add(colorTresholdTextField, c);
3422
3423     c.gridy++;
3424     this.add(colorWarningLabel, c);
3425
3426     c.gridy++;
3427     this.add(intensityTresholdLabel, c);
3428
3429     c.insets = new Insets(5,5,5,5);
3430     c.gridy++;
3431     this.add(intensityTresholdTextField, c);
3432
3433     c.gridy++;
3434     c.insets = new Insets(5,5,5,5);
3435     this.add(intensityWarningLabel, c);
3436
3437     c.gridy++;
3438     c.insets = new Insets(25,5,5,5);
3439     this.add(redValue, c);
3440

```

```

3441     c.insets = new Insets(5,5,5,5);
3442     c.gridy++;
3443     this.add(greenValue, c);
3444
3445     c.gridy++;
3446     this.add(blueValue, c);
3447
3448     c.gridy++;
3449     c.insets = new Insets(25,5,5,5);
3450     this.add(colorValueLabel,c);
3451
3452     c.gridy++;
3453     c.insets = new Insets(5,5,5,5);
3454     this.add(intensityValueLabel,c);
3455
3456     c.gridy++;
3457     this.add(resultLabel, c);
3458
3459     c.gridy++;
3460     c.insets = new Insets(25,5,30,5);
3461     this.add(applyToAllButton,c);
3462
3463     c.gridy++;
3464     c.insets = new Insets(5,5,5,5);
3465     this.add(previousButton,c);
3466
3467     c.gridy++;
3468     this.add(displayedImageLabel,c);
3469
3470     c.gridy++;
3471     c.weighty = 1;
3472     this.add(nextButton,c);
3473
3474     colorTresholdTextField.addActionListener(this);
3475     intensityTresholdTextField.addActionListener(this);
3476     helpButton.addActionListener(this);
3477     applyToAllButton.addActionListener(this);
3478     previousButton.addActionListener(this);
3479     nextButton.addActionListener(this);
3480 }
3481
3482 @Override
3483 public void actionPerformed(ActionEvent e)
3484 {
3485     if (e.getSource()==helpButton)
3486     {
3487         String helpText = myMethods.getHelpTextTissueDetection();
3488         helpDialog.newText(helpText);
3489     }
3490
3491     if (e.getSource()==colorTresholdTextField)
3492     {
3493         try
3494         {
3495             double tempDouble = Double.parseDouble(colorTresholdTextField.getText());
3496             if (tempDouble>=0 && tempDouble<=100)
3497             {
3498                 myProjectData.colorTresholdGeneral = tempDouble;
3499                 currentSectionData.colorTreshold = tempDouble;
3500                 colorWarningLabel.setText(""+myProjectData.colorTresholdGeneral);

```

```

3501         colorWarningLabel.setForeground(Color.green);
3502         colorThresholdOk=true;
3503     }
3504     else
3505     {
3506         colorWarningLabel.setForeground(Color.red);
3507         colorWarningLabel.setText("Use range 0-100");
3508         colorThresholdOk=false;
3509     }
3510 }catch(Exception error)
3511 {
3512     colorWarningLabel.setForeground(Color.red);
3513     colorWarningLabel.setText("Not a number");
3514     colorThresholdOk=false;
3515 }
3516
3517 if (colorThresholdOk && intensityThresholdOk)
3518 {
3519     applyToAllButton.setEnabled(true);
3520 }
3521 else
3522 {
3523     applyToAllButton.setEnabled(false);
3524 }
3525
3526 upperScrollPane.requestFocus();
3527
3528 tissueDetection_middlePanel.updateDisplayImage();
3529 tissueDetection_middlePanel.repaint();
3530 }
3531 if (e.getSource()==intensityThresholdTextField)
3532 {
3533     try
3534     {
3535         double tempDouble = Double.parseDouble(intensityThresholdTextField.getText());
3536         if (tempDouble>=0 && tempDouble<=765)
3537         {
3538             myProjectData.intensityThresholdGeneral = tempDouble;
3539             currentSectionData.intensityThreshold = tempDouble;
3540             intensityWarningLabel.setText(""+myProjectData.intensityThresholdGeneral);
3541             intensityWarningLabel.setForeground(Color.green);
3542             intensityThresholdOk=true;
3543         }
3544         else
3545         {
3546             intensityWarningLabel.setForeground(Color.red);
3547             intensityWarningLabel.setText("Use range 0-765");
3548             intensityThresholdOk=false;
3549         }
3550     }catch(Exception error)
3551     {
3552         intensityWarningLabel.setForeground(Color.red);
3553         intensityWarningLabel.setText("Not a number");
3554         intensityThresholdOk=false;
3555     }
3556
3557 if (colorThresholdOk && intensityThresholdOk)
3558 {
3559     applyToAllButton.setEnabled(true);
3560 }

```

```

3561     else
3562     {
3563         applyToAllButton.setEnabled(false);
3564     }
3565
3566     upperScrollPane.requestFocus();
3567
3568     tissueDetection_middlePanel.updateDisplayImage();
3569     tissueDetection_middlePanel.repaint();
3570 }
3571
3572 if (e.getSource()==applyToAllButton)
3573 {
3574     //The settings for color treshold and intensity treshold
3575     //is applied to all sections. All areas and volumes are
3576     //also recalculated to match the new tresholds.
3577
3578     int numberOfSections=myProjectData.totalNumberOfSections;
3579
3580     progressMonitor = new ProgressMonitor(null,
3581         "Analyzing sections",
3582         "Sections analyzed: 0", 0, numberOfSections);
3583
3584     progressMonitor.setMillisToDecideToPopup(0);
3585     progressMonitor.setMillisToPopup(0);
3586     progressMonitor.setProgress(0);
3587     progressMonitor.setNote("Sections analyzed: 0");
3588
3589     AnalyzeAllSections analyzeAllSections = new AnalyzeAllSections();
3590     analyzeAllSections.execute();
3591 }
3592
3593 if (e.getSource()==previousButton)
3594 {
3595     int subject=tissueDetection_middlePanel.displaySubject;
3596     int section=tissueDetection_middlePanel.displaySection;
3597
3598     if (section>0)
3599     {
3600         tissueDetection_middlePanel.displaySection--;
3601     }
3602     else
3603     {
3604         if (subject>0)
3605         {
3606             tissueDetection_middlePanel.displaySubject--;
3607             for (int i=0;i<myProjectData.numberOfLevels;i++)
3608             {
3609                 if (myProjectData.sectionExists[subject-1][i])
3610                     tissueDetection_middlePanel.displaySection=i;
3611             }
3612         }
3613         else
3614         {
3615             tissueDetection_middlePanel.displaySubject=myProjectData.numberOfSubjects-1;
3616             for (int i=0;i<myProjectData.numberOfLevels;i++)
3617             {
3618                 if (myProjectData.sectionExists[myProjectData.numberOfSubjects-1][i])
3619                     tissueDetection_middlePanel.displaySection=i;
3620             }

```

```

3621     }
3622 }
3623
3624 tissueDetection_middlePanel.updateDisplayImage();
3625 tissueDetection_middlePanel.repaint();
3626 }
3627
3628 if (e.getSource()==nextButton)
3629 {
3630     int subject=tissueDetection_middlePanel.displaySubject;
3631     int section=tissueDetection_middlePanel.displaySection;
3632
3633     if (subject==myProjectData.numberOfSubjects-1)
3634     {
3635         if (section ==myProjectData.numberOfLevels-1)
3636         {
3637             tissueDetection_middlePanel.displaySubject=0;
3638             tissueDetection_middlePanel.displaySection=0;
3639         }
3640         else
3641         {
3642             if (myProjectData.sectionExists[subject][section+1])
3643             {
3644                 tissueDetection_middlePanel.displaySection++;
3645             }
3646             else
3647             {
3648                 tissueDetection_middlePanel.displaySubject=0;
3649                 tissueDetection_middlePanel.displaySection=0;
3650             }
3651         }
3652     }
3653     else
3654     {
3655         if (section ==myProjectData.numberOfLevels-1)
3656         {
3657             tissueDetection_middlePanel.displaySubject++;
3658             tissueDetection_middlePanel.displaySection=0;
3659         }
3660         else
3661         {
3662             if (myProjectData.sectionExists[subject][section+1])
3663             {
3664                 tissueDetection_middlePanel.displaySection++;
3665             }
3666             else
3667             {
3668                 tissueDetection_middlePanel.displaySubject++;
3669                 tissueDetection_middlePanel.displaySection=0;
3670             }
3671         }
3672     }
3673     tissueDetection_middlePanel.updateDisplayImage();
3674     tissueDetection_middlePanel.repaint();
3675 }
3676 }
3677
3678 public class AnalyzeAllSections extends SwingWorker<Void, Void>
3679 {
3680     @Override

```

```

3681     public Void doInBackground()
3682     {
3683         frame.setVisible(false);
3684         int counter=0;
3685         progressMonitor.setProgress(counter);
3686         for (int subject=0; subject<myProjectData.numberOfSubjects; subject++)
3687         {
3688             for (int section=0; section<myProjectData.numberOfLevels; section++)
3689             {
3690                 if (myProjectData.sectionExists[subject][section])
3691                 {
3692                     currentSectionData=myMethods.readSectionData(subject, section);
3693
3694                     currentSectionData.colorThreshold=myProjectData.colorThresholdGeneral;
3695                     currentSectionData.intensityThreshold=myProjectData.intensityThresholdGeneral;
3696
3697                     myMethods.saveSectionData(subject, section, currentSectionData);
3698
3699                     myMethods.analyzeSection(subject, section);
3700                     myMethods.updateSectionResultsObject(subject, section);
3701                     counter++;
3702                     progressMonitor.setProgress(counter);
3703                     progressMonitor.setNote("Sections analyzed: "+counter);
3704
3705                     if (progressMonitor.isCanceled())
3706                     {
3707                         break;
3708                     }
3709                 }
3710             }
3711             myMethods.analyzeSubject(subject);
3712             if (progressMonitor.isCanceled())
3713             {
3714                 errorDialog.newText(myMethods.getErrorTextApplyToAllInterrupted());
3715                 errorDialog.setVisible(true);
3716                 break;
3717             }
3718         }
3719         frame.setVisible(true);
3720         return null;
3721     }
3722
3723     @Override
3724     public void done() {}
3725 }
3726
3727
3728 @SuppressWarnings("LeakingThisInConstructor")
3729 private class Conversion_MiddlePanel extends JPanel implements MouseListener {
3730     //Lets the user click the conversion image to select a distance in pixels.
3731     //The corresponding length in millimeters can then be entered in the right
3732     //side panel. This is used to convert the area measurments in pixels to
3733     //square millimeters and cubic millimeters.
3734
3735     BufferedImage conversionImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
3736
3737     public Conversion_MiddlePanel() {
3738         addMouseListener(this);
3739     }
3740

```

```

3741 public void whenClickedInTree() {
3742     middleScrollPane.getViewPort().add(conversion_middlePanel);
3743     rightScrollPane.getViewPort().add(conversion_rightSidePanel);
3744
3745     generateConversionImage();
3746
3747     if (myProjectData.distancePixels==0) {
3748         conversion_rightSidePanel.distanceLabel.setText("Click image");
3749         conversion_rightSidePanel.distanceLabel.setForeground(Color.red);
3750     }
3751     else {
3752         conversion_rightSidePanel.distanceLabel.setText(""+myProjectData.distancePixels);
3753         conversion_rightSidePanel.distanceLabel.setForeground(Color.green);
3754     }
3755
3756     if (myProjectData.distanceMM<=0) {
3757         conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.red);
3758         conversion_rightSidePanel.numberOfMMLabel.setText("Must be more than 0");
3759     }
3760     else {
3761         conversion_rightSidePanel.distancemmTextField.setText(""+myProjectData.distanceMM);
3762         conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.green);
3763         conversion_rightSidePanel.numberOfMMLabel.setText(""+myProjectData.distanceMM);
3764     }
3765
3766     if (myProjectData.conversionIsDefined) {
3767         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.green);
3768
3769         conversion_rightSidePanel.conversionFactorLabel.setText(""+Math.round(myProjectData.conversionFactor));
3770     }
3771     else {
3772         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.red);
3773         conversion_rightSidePanel.conversionFactorLabel.setText("Not defined");
3774     }
3775 }
3776 public void whenLeftInTree() {}
3777
3778 public void generateConversionImage() {
3779     String conversionFileName = myProjectData.projectLocation+"/ConversionImage";
3780     File conversionFile = new File(conversionFileName);
3781
3782     try
3783     {
3784         conversionImage = (ImageIO.read(conversionFile));
3785     }catch(Exception error){}
3786
3787     boolean[][] linePixels = new boolean[conversionImage.getWidth()][conversionImage.getHeight()];
3788     myMethods.setBooleanArrayValue(linePixels, false);
3789
3790     linePixels = myMethods.drawLineBoolean(linePixels,
3791         myProjectData.firstConversionPoint, myProjectData.secondConversionPoint);
3792
3793     linePixels = myMethods.widenLine(linePixels, myProjectData.lineWidth);
3794     myMethods.changePixelsInImage(conversionImage, linePixels, myProjectData.lineColor);
3795 }
3796
3797 public void calculateDistanceInPixels() {
3798     double distanceX;
3799     double distanceY;
3800     double distance;

```



```

3800
3801     distanceX = Math.abs(myProjectData.firstConversionPoint.x-
myProjectData.secondConversionPoint.x);
3802     distanceY = Math.abs(myProjectData.firstConversionPoint.y-
myProjectData.secondConversionPoint.y);
3803     distance = Math.sqrt(distanceX*distanceX+distanceY*distanceY);
3804     distance = Math.round(distance);
3805
3806     myProjectData.distancePixels= (int) distance;
3807
3808     conversion_rightSidePanel.distanceLabel.setForeground(Color.green);
3809     conversion_rightSidePanel.distanceLabel.setText(""+myProjectData.distancePixels);
3810
3811     if (myProjectData.distancePixels==0) {
3812         conversion_rightSidePanel.distanceLabel.setForeground(Color.red);
3813         conversion_rightSidePanel.distanceLabel.setText("Click image");
3814     }
3815
3816     calculateConversionFactor();
3817 }
3818
3819 public void getValueFromTextField() {
3820     double distanceMM;
3821
3822     String newValue = conversion_rightSidePanel.distancemmTextField.getText();
3823     try {
3824         distanceMM = Double.parseDouble(newValue);
3825         myProjectData.distanceMM = distanceMM;
3826         conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.green);
3827         conversion_rightSidePanel.numberOfMMLabel.setText(""+distanceMM);
3828
3829         if (distanceMM<=0) {
3830             conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.red);
3831             conversion_rightSidePanel.numberOfMMLabel.setText("Must be more than 0");
3832         }
3833     }
3834     catch (NumberFormatException error) {
3835         conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.red);
3836         conversion_rightSidePanel.numberOfMMLabel.setText("Not a number");
3837     }
3838
3839     calculateConversionFactor();
3840 }
3841
3842 public void calculateConversionFactor() {
3843     if(myProjectData.distancePixels>0 && myProjectData.distanceMM>0) {
3844         myProjectData.conversionFactor =
(myProjectData.distancePixels*myProjectData.distancePixels)
3845         /(myProjectData.distanceMM*myProjectData.distanceMM);
3846
3847         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.green);
3848
3849         conversion_rightSidePanel.conversionFactorLabel.setText(""+Math.round(myProjectData.conversionFactor));
3850         myProjectData.conversionIsDefined=true;
3851         conversion_rightSidePanel.repaint();
3852     }
3853     else {
3854         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.red);
3855         conversion_rightSidePanel.conversionFactorLabel.setText("Not defined");
3856         myProjectData.conversionFactor = 1;

```

```

3856         myProjectData.conversionIsDefined = false;
3857     }
3858
3859     for (int subject=0;subject<myProjectData.numberOfSubjects;subject++) {
3860         for (int section=0; section<myProjectData.numberOfLevels; section++) {
3861             if (myProjectData.sectionExists[subject][section]) {
3862                 myMethods.updateSectionResultsObject(subject, section);
3863             }
3864         }
3865         myMethods.analyzeSubject(subject);
3866     }
3867 }
3868
3869 @Override
3870 public void paint(Graphics g)
3871 {
3872     super.paint(g);
3873
3874     int width = conversionImage.getWidth();
3875     int height = conversionImage.getHeight();
3876
3877     conversion_middlePanel.setPreferredSize(new
Dimension(width*myProjectData.zoomFactor/100, height*myProjectData.zoomFactor/100));
3878
3879     g.drawImage(conversionImage,
3880         0,0, width*myProjectData.zoomFactor/100, height*myProjectData.zoomFactor/100,
3881         0,0, width, height, this);
3882
3883     this.revalidate();
3884 }
3885
3886 @Override
3887 public void mouseClicked(MouseEvent event) {
3888     Point currentPoint=event.getPoint();
3889
3890     currentPoint.x=100*currentPoint.x/myProjectData.zoomFactor;
3891     currentPoint.y=100*currentPoint.y/myProjectData.zoomFactor;
3892
3893     if (currentPoint.x<conversionImage.getWidth() && currentPoint.y<conversionImage.getHeight()) {
3894         if (event.getButton()==MouseEvent.BUTTON1) {
3895             myProjectData.firstConversionPoint = currentPoint;
3896         }
3897         if (event.getButton()==MouseEvent.BUTTON3) {
3898             myProjectData.secondConversionPoint = currentPoint;
3899         }
3900
3901         this.calculateDistanceInPixels();
3902         this.generateConversionImage();
3903         repaint();
3904     }
3905 }
3906
3907 @Override
3908 public void mousePressed(MouseEvent event) {}
3909 @Override
3910 public void mouseEntered(MouseEvent event) {}
3911 @Override
3912 public void mouseExited(MouseEvent event) {}
3913 @Override
3914 public void mouseReleased(MouseEvent event) {}

```

```

3915
3916     @Override
3917     public String toString()
3918     {
3919         return "Conversion Panel";
3920     }
3921 }
3922
3923 @SuppressWarnings("LeakingThisInConstructor")
3924 public class Conversion_RightSidePanel extends JPanel implements ActionListener {
3925     JButton helpButton;
3926
3927     JLabel descriptionLabel1a = new JLabel("Left click for first point");
3928     JLabel descriptionLabel1b = new JLabel("Right click for second point");
3929     JLabel descriptionLabel2 = new JLabel("Distance in pixels will be calculated");
3930     JLabel descriptionLabel3 = new JLabel("Enter corresponding distance in mm");
3931
3932     JLabel infoLabel1 = new JLabel("Distance (pixels):");
3933     JLabel distanceLabel = new JLabel("0");
3934     JLabel infoLabel2 = new JLabel("Distance (mm):");
3935     JTextField distancemmTextField = new JTextField("0",8);
3936     JLabel numberOfMMLabel = new JLabel("Cant be zero");
3937     JLabel infoLabel3 = new JLabel("Pixels/mm2:");
3938     JLabel conversionFactorLabel = new JLabel("0");
3939
3940     public Conversion_RightSidePanel()
3941     {
3942         helpButton = new JButton("Help");
3943
3944         this.setLayout(new GridBagLayout());
3945         GridBagConstraints c = new GridBagConstraints();
3946
3947         c.gridx = 0;
3948         c.gridy = 0;
3949         c.gridwidth = 1;
3950         c.anchor = GridBagConstraints.PAGE_START;
3951         c.insets = new Insets(10,5,20,5);
3952
3953         this.add(helpButton, c);
3954
3955         c.gridy ++;
3956         c.insets = new Insets(5,5,5,5);
3957         this.add(descriptionLabel1a, c);
3958
3959         c.gridy ++;
3960         this.add(descriptionLabel1b, c);
3961
3962         c.gridy ++;
3963         this.add(descriptionLabel2, c);
3964
3965         c.gridy ++;
3966         this.add(descriptionLabel3, c);
3967
3968         c.gridy ++;
3969         c.insets = new Insets(20,5,5,5);
3970         this.add(infoLabel1, c);
3971
3972         c.gridy ++;
3973         c.insets = new Insets(5,5,5,5);
3974         this.add(distanceLabel, c);

```

```

3975
3976     c.gridy ++;
3977     c.insets = new Insets(20,5,5,5);
3978     this.add(infoLabel2, c);
3979
3980     c.gridy ++;
3981     c.insets = new Insets(5,5,5,5);
3982     this.add(distancemmTextField, c);
3983
3984     c.gridy ++;
3985     this.add(numberOfMMLLabel, c);
3986
3987     c.gridy ++;
3988     c.insets = new Insets(20,5,5,5);
3989     this.add(infoLabel3, c);
3990
3991     c.weighty = 1;
3992     c.gridy ++;
3993     c.insets = new Insets(5,5,5,5);
3994     this.add(conversionFactorLabel, c);
3995
3996     helpButton.addActionListener(this);
3997     distancemmTextField.addActionListener(this);
3998 }
3999
4000 @Override
4001 public void actionPerformed(ActionEvent e) {
4002     if (e.getSource()==distancemmTextField) {
4003         upperScrollPane.requestFocus();
4004         conversion_middlePanel.getValueFromTextField();
4005     }
4006
4007     if (e.getSource()==helpButton) {
4008         String helpText = myMethods.getHelpTextChoseConversionImage();
4009         helpDialog.newText(helpText);
4010     }
4011 }
4012 }
4013
4014 public class Subject_MiddlePanel extends JPanel
4015 {
4016     //Displays the original images for all sections in this subject
4017     //to get an overview
4018     int subject=0;
4019
4020     BufferedImage[] images = new BufferedImage[100];
4021     int imageHeigth=10;
4022     int imageWidth=10;
4023     int sectionsInSubject;
4024
4025     public void whenClickedInTree(int currentSubject)
4026     {
4027         myMethods.setWaitCursor();
4028
4029         subject=currentSubject;
4030         sectionsInSubject=0;
4031         File file;
4032         String fileName;
4033         for (int i=0;i<myProjectData.numberOfLevels;i++)
4034         {

```

```

4035         if (myProjectData.sectionExists[subject][i])
4036         {
4037             try
4038             {
4039                 fileName = myMethods.getOriginalImageFileName(subject, i);
4040                 file = new File(fileName);
4041                 images[i]=ImageIO.read(file);
4042                 imageWidth=images[i].getWidth();
4043                 imageHeigth=images[i].getHeight();
4044                 sectionsInSubject++;
4045             } catch (IOException event) { }
4046         }
4047     }
4048
4049     middleScrollPane.setViewportView(subject_middlePanel);
4050     rightScrollPane.setViewportView(subject_rightSidePanel);
4051     middleScrollPane.repaint();
4052
4053     myMethods.setCustomCursor();
4054 }
4055
4056 @Override
4057 public void paint(Graphics g)
4058 {
4059     super.paint(g);
4060
4061     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4062     int drawHeight = imageHeigth*myProjectData.zoomFactor/100;
4063
4064     FontMetrics fontMetrics = g.getFontMetrics();
4065     int fontHeight = fontMetrics.getHeight();
4066     int subjectNumber;
4067
4068     subject_middlePanel.setPreferredSize(new Dimension(drawWidth+300,
sectionsInSubject*drawHeight));
4069
4070     for (int i=0;i<sectionsInSubject;i++)
4071     {
4072         subjectNumber=i+1;
4073         g.drawImage(images[i], 0, drawHeight*i, drawWidth, drawHeight, this);
4074         g.drawString("Section "+subjectNumber, drawWidth+5, drawHeight*i+fontHeight);
4075         g.drawString(""+myProjectData.sectionName[subject][i], drawWidth+5,
drawHeight*i+2*fontHeight);
4076         g.drawString("Level: "+myProjectData.bregmaLevels[subject][i], drawWidth+5,
drawHeight*i+3*fontHeight);
4077     }
4078
4079     revalidate();
4080 }
4081 }
4082
4083 public class Subject_RightSidePanel extends JPanel
4084 {
4085     JLabel infoLabel = new JLabel("Subject overview");
4086
4087     public Subject_RightSidePanel()
4088     {
4089         this.setLayout(new GridBagLayout());
4090         GridBagConstraints c = new GridBagConstraints();
4091

```

```

4092     c.gridx = 0;
4093     c.gridy = 0;
4094     c.gridwidth = 1;
4095     c.weighty=1;
4096     c.insets = new Insets(10,5,5,5);
4097     c.anchor = GridBagConstraints.PAGE_START;
4098     this.add(infoLabel, c);
4099 }
4100 }
4101
4102 private class Section_MiddlePanel extends JPanel
4103 {
4104     //Displays the original image and derived images to get
4105     //an overview of this section.
4106     int subject=0;
4107     int section=0;
4108
4109     BufferedImage originalImage;
4110     BufferedImage adjustedImage;
4111     BufferedImage areasImage;
4112     BufferedImage[] UDImages;
4113
4114     int imageHeight=10;
4115     int imageWidth=10;
4116
4117     public void whenClickedInTree(int currentSubject, int currentSection)
4118     {
4119         myMethods.setWaitCursor();
4120
4121         subject=currentSubject;
4122         section=currentSection;
4123
4124         UDImages=new BufferedImage[myProjectData.numberOfUDA];
4125
4126         currentSectionData=myMethods.readSectionData(subject, section);
4127
4128         originalImage = myMethods.readImageFile(subject, section);
4129         openSection.originalImage = originalImage;
4130         openSection.tissuePixels = myMethods.getTissuePixels(originalImage);
4131
4132         section_rightSidePanel.levelLabel.setText(""+ myProjectData.bregmaLevels[subject][section]);
4133         section_rightSidePanel.bregmaLevelTextField.setText(""+
myProjectData.bregmaLevels[subject][section]);
4134
4135         section_rightSidePanel.updateBregmaLevel();
4136
4137         this.generateImages();
4138
4139         middleScrollPane.getViewPort().add(section_middlePanel);
4140         rightScrollPane.getViewPort().add(section_rightSidePanel);
4141
4142         repaint();
4143         myMethods.setCustomCursor();
4144     }
4145     public void whenLeftInTree(int currentSubject, int currentSection) {}
4146
4147     public void generateImages()
4148     {
4149         adjustedImage = myMethods.copyBufferedImage(originalImage);

```

```

4150     myMethods.changePixelsInImage(adjustedImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
4151     myMethods.changePixelsInImage(adjustedImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
4152
4153     areasImage = myMethods.copyBufferedImage(originalImage);
4154
4155     imageWidth=originalImage.getWidth();
4156     imageHeigth=originalImage.getHeight();
4157
4158     for (int i=0;i<myProjectData.numberOfUDA;i++) {
4159         UDAimages[i] = myMethods.copyBufferedImage(originalImage);
4160     }
4161
4162     boolean[][] ventriclePixels = myMethods.newGetVentriclePixels2(openSection.tissuePixels);
4163
4164     boolean[][] dividePixels = myMethods.getBooleanArrayFromPointList(
4165         currentSectionData.pointList,
4166         currentSectionData.imageWidth,
4167         currentSectionData.imageHeight);
4168
4169     boolean[][] isLeft = myMethods.isLeftFromArray(dividePixels);
4170
4171     WritableRaster destinationRaster=areasImage.getRaster();
4172
4173     int[] black = {0,0,0};
4174     int[] red = {255,0,0};
4175     int[] blue = {0,0,255};
4176     int[] green = {0,255,0};
4177
4178     for (int i=0; i<imageWidth;i++)
4179         for (int j=0; j<imageHeigth;j++)
4180         {
4181             if (openSection.tissuePixels[i][j] && isLeft[i][j])
4182             {
4183                 destinationRaster.setPixel(i, j, red);
4184             }
4185             if (openSection.tissuePixels[i][j] && !isLeft[i][j])
4186             {
4187                 destinationRaster.setPixel(i, j, black);
4188             }
4189             if (ventriclePixels[i][j] && isLeft[i][j])
4190             {
4191                 destinationRaster.setPixel(i, j, blue);
4192             }
4193             if (ventriclePixels[i][j] && !isLeft[i][j])
4194             {
4195                 destinationRaster.setPixel(i, j, green);
4196             }
4197         }
4198
4199     for (int UDAnumber=0; UDAnumber<myProjectData.numberOfUDA; UDAnumber++)
4200     {
4201         myMethods.changePixelsInImage(UDAimages[UDAnumber],
currentSectionData.markedAsBackground, myProjectData.addBackgroundColor);
4202         myMethods.changePixelsInImage(UDAimages[UDAnumber],
currentSectionData.markedAsTissue, myProjectData.addTissueColor);
4203
4204         boolean[][] linePixels = myMethods.getBooleanArrayFromPointList(
4205             currentSectionData.UDApointList[UDAnumber],

```



```

4206         currentSectionData.imageWidth,
4207         currentSectionData.imageHeight);
4208
4209         boolean[][] UDAoutlinePixels = myMethods.widenLine(linePixels, myProjectData.lineWidth);
4210         myMethods.changePixelsInImage(UDAIMages[UDANumber], UDAoutlinePixels, black );
4211
4212         if (currentSectionData.haveUdaOrigin[UDANumber])
4213         {
4214             Point UDAstartPosition = currentSectionData.udaOrigin[UDANumber];
4215
4216             for (int i=0; i<imageWidth;i++)
4217             {
4218                 linePixels[i][0]=true;
4219                 linePixels[i][imageHeight-1]=true;
4220             }
4221             for (int i=0; i<imageHeight;i++)
4222             {
4223                 linePixels[0][i]=true;
4224                 linePixels[imageWidth-1][i]=true;
4225             }
4226
4227             boolean[][] udaPixels = myMethods.newFillInside(linePixels, UDAstartPosition);
4228
4229             boolean includeTissue = myProjectData.includeTissue[UDANumber];
4230             boolean includeBackground = myProjectData.includeBackground[UDANumber];
4231
4232             for (int i=0; i<imageWidth;i++)
4233                 for (int j=0; j<imageHeight;j++)
4234                 {
4235                     if (openSection.tissuePixels[i][j] && !includeTissue) udaPixels[i][j]=false;
4236                     if (!openSection.tissuePixels[i][j] && !includeBackground) udaPixels[i][j]=false;
4237                 }
4238
4239             myMethods.changePixelsInImage(UDAIMages[UDANumber], udaPixels, green );
4240         }
4241     }
4242 }
4243
4244 @Override
4245 public void paint(Graphics g)
4246 {
4247     super.paint(g);
4248
4249     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4250     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4251     FontMetrics fontMetrics = g.getFontMetrics();
4252     int fontHeight = fontMetrics.getHeight();
4253
4254     section_middlePanel.setPreferredSize(new Dimension(drawWidth+300,
4255 (3+myProjectData.numberOfUDA)*drawHeight));
4256
4257     g.drawImage(originalImage, 0, 0, drawWidth, drawHeight, this);
4258     g.drawString("Original image", drawWidth+5, fontHeight);
4259     g.drawImage(adjustedImage, 0, drawHeight, drawWidth, drawHeight, this);
4260     g.drawString("Adjusted image", drawWidth+5, drawHeight+fontHeight);
4261     g.drawImage(areasImage, 0, 2*drawHeight, drawWidth, drawHeight, this);
4262     g.drawString("Detected areas", drawWidth+5, 2*drawHeight+fontHeight);
4263
4264     for (int UDANumber=0; UDANumber<myProjectData.numberOfUDA; UDANumber++)
4265     {

```



```

4265         g.drawImage(UDAIMAGES[UDANumber], 0, (3+UDANumber)*drawHeigh, drawWidth,
drawHeigh, this);
4266         g.drawString(" "+myProjectData.UDANames[UDANumber], drawWidth+5,
(3+UDANumber)*drawHeigh+fontHeight);
4267     }
4268
4269     revalidate();
4270 }
4271 }
4272
4273 @SuppressWarnings("LeakingThisInConstructor")
4274 private class Section_RightSidePanel extends JPanel implements ActionListener
4275 {
4276     //The level used for this section can be changed here. If the levels
4277     //in a subject are not continuous a warning will be generated here and
4278     //when the subject results and the section results are displayed.
4279
4280     JLabel infoLabel = new JLabel("Current level");
4281     JLabel levelLabel = new JLabel("");
4282     JButton updateBregmaLevelButton;
4283     JTextField bregmaLevelTextField = new JTextField("",8);
4284     JLabel warningLabel= new JLabel("");
4285
4286     public Section_RightSidePanel()
4287     {
4288         updateBregmaLevelButton=new JButton("Update Bregma Level");
4289
4290         this.setLayout(new GridBagLayout());
4291         GridBagConstraints c = new GridBagConstraints();
4292
4293         c.gridx = 0;
4294         c.gridy = 0;
4295         c.gridwidth = 1;
4296         c.insets = new Insets(10,5,5,5);
4297         c.anchor = GridBagConstraints.PAGE_START;
4298         this.add(infoLabel, c);
4299
4300         c.gridy++;
4301         c.insets = new Insets(5,5,5,5);
4302         this.add(levelLabel, c);
4303
4304         c.gridy++;
4305         c.insets = new Insets(5,5,5,5);
4306         this.add(updateBregmaLevelButton, c);
4307
4308         c.gridy++;
4309         c.insets = new Insets(5,5,5,5);
4310         this.add(bregmaLevelTextField, c);
4311
4312         c.gridy++;
4313         c.weighty=1;
4314         this.add(warningLabel, c);
4315
4316         updateBregmaLevelButton.addActionListener(this);
4317         bregmaLevelTextField.addActionListener(this);
4318     }
4319
4320     public void updateBregmaLevel()
4321     {
4322         boolean isNumber=false;

```

```

4323
4324     String bregmaLevel = bregmaLevelTextField.getText();
4325     try
4326     {
4327         myProjectData.bregmaLevels[section_middlePanel.subject][section_middlePanel.section] =
Double.parseDouble(bregmaLevel);
4328         isNumber=true;
4329         warningLabel.setText("");
4330         levelLabel.setText(""+
myProjectData.bregmaLevels[section_middlePanel.subject][section_middlePanel.section]);
4331     } catch (Exception error) {}
4332
4333     if (!isNumber)
4334     {
4335         warningLabel.setForeground(Color.red);
4336         warningLabel.setText("Not a number");
4337     }
4338     else
4339     {
4340         boolean allIncreasing=true;
4341         boolean allDecreasing=true;
4342
4343         int subject= section_middlePanel.subject;
4344
4345         for(int i=0;i<myProjectData.numberOfLevels-1;i++)
4346         {
4347             if (myProjectData.sectionExists[subject][i+1])
4348             {
4349                 if
(myProjectData.bregmaLevels[subject][i]>=myProjectData.bregmaLevels[subject][i+1]) allIncreasing=false;
4350                 if
(myProjectData.bregmaLevels[subject][i]<=myProjectData.bregmaLevels[subject][i+1]) allDecreasing=false;
4351             }
4352         }
4353
4354         if (!(allIncreasing || allDecreasing))
4355         {
4356             warningLabel.setForeground(Color.red);
4357             warningLabel.setText("Values must continously increase or decrease");
4358             myProjectData.continuousLevels[subject]=false;
4359         }
4360         else
4361         {
4362             warningLabel.setForeground(Color.red);
4363             warningLabel.setText("");
4364             myProjectData.continuousLevels[subject]=true;
4365         }
4366     }
4367
4368     myProjectData.allLevelsContinous=true;
4369     for (int i=0; i<myProjectData.numberOfSubjects;i++)
4370     {
4371         if (!myProjectData.continuousLevels[i]) myProjectData.allLevelsContinous=false;
4372     }
4373
4374     myMethods.analyzeSubject(section_middlePanel.subject);
4375 }
4376
4377 @Override
4378 public void actionPerformed(ActionEvent e)

```

```

4379     {
4380         if (e.getSource()==updateBregmaLevelButton)
4381         {
4382             this.updateBregmaLevel();
4383         }
4384         if (e.getSource()==bregmaLevelTextField)
4385         {
4386             upperScrollPane.requestFocus();
4387             this.updateBregmaLevel();
4388         }
4389     }
4390 }
4391
4392 @SuppressWarnings("LeakingThisInConstructor")
4393 private class AdjustImage_MiddlePanel extends JPanel implements MouseListener,
4394 MouseMotionListener
4395 {
4396     //Here the user can remove unwanted tissue and add tissue to fill for example tears.
4397     //It is also possible to change the color treshold and intensity treshold
4398     //for this section only.
4399     int subject=0;
4400     int section=0;
4401
4402     boolean addBackground=true;
4403
4404     int size = 10;
4405     int circleDrawSize=10;
4406     int[] drawColor = {255,255,255};
4407     Cursor transparentCursor = new Cursor(Cursor.HAND_CURSOR);
4408     boolean cursorOverPanel = false;
4409
4410     Point currentPoint = new Point(100,100);
4411
4412     int circlePosX=100;
4413     int circlePosY=100;
4414     int circleSize=20;
4415
4416     int cursorPosX=100;
4417     int cursorPosY=100;
4418
4419     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
4420     BufferedImage imageToAdjust = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
4421     BufferedImage modifiedImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
4422
4423     boolean somethingChanged = false;
4424
4425     public AdjustImage_MiddlePanel()
4426     {
4427         this.setTransparentCursor();
4428         addMouseListener(this);
4429         addMouseMotionListener(this);
4430     }
4431
4432     @Override
4433     public void mouseClicked(MouseEvent event)
4434     {
4435         if (event.getButton() == MouseEvent.BUTTON1)
4436         {
4437             //Adds or remove tissue in a circle with radius size.

```

```

4438     int imageWidth = originalImage.getWidth();
4439     int imageHeight = originalImage.getHeight();
4440     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4441     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4442
4443     currentPoint=event.getPoint();
4444     currentPoint.y = currentPoint.y-drawHeight;
4445     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4446     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4447
4448     if (currentPoint.x<imageWidth && currentPoint.x>-1
4449         && currentPoint.y<imageHeight && currentPoint.y>-1)
4450         this.newDrawPoint(currentPoint);
4451
4452     currentPoint=event.getPoint();
4453     currentPoint.y = currentPoint.y-2*drawHeight;
4454     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4455     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4456
4457     if (currentPoint.x<imageWidth && currentPoint.x>-1
4458         && currentPoint.y<imageHeight && currentPoint.y>-1)
4459         this.newDrawPoint(currentPoint);
4460
4461     repaint();
4462
4463     currentPoint=event.getPoint();
4464     cursorPosX=currentPoint.x;
4465     cursorPosY=currentPoint.y;
4466
4467     circleDrawSize=size*myProjectData.zoomFactor/100;
4468
4469     circlePosX=cursorPosX-circleDrawSize;
4470     circlePosY=cursorPosY-circleDrawSize;
4471     circleSize=circleDrawSize*2;
4472 }
4473 if (event.getButton() == MouseEvent.BUTTON3)
4474 {
4475     //Recalculates the result image.
4476
4477     adjustImage_middlePanel.generateAdjustedImage();
4478     adjustImage_middlePanel.generateResultImage();
4479     adjustImage_middlePanel.repaint();
4480 }
4481 }
4482
4483 @Override
4484 public void mouseDragged(MouseEvent event)
4485 {
4486     //Modifies image in the same way as if a mouseClicked was triggered.
4487
4488     currentPoint=event.getPoint();
4489
4490     int imageWidth = originalImage.getWidth();
4491     int imageHeight = originalImage.getHeight();
4492     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4493     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4494
4495     currentPoint.y = currentPoint.y-drawHeight;
4496     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4497

```

```

4498     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4499
4500
4501     if (currentPoint.x<imageWidth && currentPoint.x>-1
4502         && currentPoint.y<imageHeight && currentPoint.y>-1)
4503         this.newDrawPoint(currentPoint);
4504
4505     currentPoint=event.getPoint();
4506     currentPoint.y = currentPoint.y-2*drawHeight;
4507     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4508     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4509
4510     if (currentPoint.x<imageWidth && currentPoint.x>-1
4511         && currentPoint.y<imageHeight && currentPoint.y>-1)
4512         this.newDrawPoint(currentPoint);
4513
4514     repaint();
4515
4516     currentPoint=event.getPoint();
4517     cursorPosX=currentPoint.x;
4518     cursorPosY=currentPoint.y;
4519
4520     circleDrawSize=size*myProjectData.zoomFactor/100;
4521
4522     circlePosX=cursorPosX-circleDrawSize;
4523     circlePosY=cursorPosY-circleDrawSize;
4524     circleSize=circleDrawSize*2;
4525 }
4526
4527 @Override
4528 public void mouseMoved(MouseEvent event)
4529 {
4530     //Displays RGB values when mouse is over an image. Also displays
4531     //the values for color and intensity and whether the pixel
4532     //is considered to be tissue or background.
4533     int[] color = new int[3];
4534
4535     int imageWidth = originalImage.getWidth();
4536     int imageHeight = originalImage.getHeight();
4537
4538     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4539     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4540
4541     Point cursorPosition = event.getPoint();
4542     Point imagePosition = new Point();
4543
4544     middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
4545     cursorOverPanel=false;
4546     adjustImage_rightSidePanel.redValue.setText("Red: ---");
4547     adjustImage_rightSidePanel.greenValue.setText("Green: ---");
4548     adjustImage_rightSidePanel.blueValue.setText("Blue: ---");
4549     adjustImage_rightSidePanel.colorValueLabel.setText("Color: ---");
4550     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
4551     adjustImage_rightSidePanel.resultLabel.setText("Pixel is: ---");
4552
4553     if (
4554         cursorPosition.y<drawHeight &&
4555         cursorPosition.x<drawWidth )
4556     {
4557

```

```

4558 imagePosition.x=cursorPosition.x*100/myProjectData.zoomFactor;
4559 imagePosition.y=cursorPosition.y;
4560 imagePosition.y=imagePosition.y*100/myProjectData.zoomFactor;
4561
4562 WritableRaster raster = originalImage.getRaster();
4563 color=raster.getPixel(imagePosition.x,imagePosition.y,color);
4564 adjustImage_rightSidePanel.redValue.setText("Red: "+color[0]);
4565 adjustImage_rightSidePanel.greenValue.setText("Green: "+color[1]);
4566 adjustImage_rightSidePanel.blueValue.setText("Blue: "+color[2]);
4567
4568 int firstInt = color[0]+color[2];
4569 int secondInt = color[0]+color[1]+color[2];
4570 double tempDouble = (double) firstInt/secondInt;
4571 tempDouble = tempDouble*1000;
4572 tempDouble = Math.round(tempDouble);
4573 tempDouble = tempDouble/10;
4574
4575 adjustImage_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
4576 adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
4577 if(myMethods.pixellIsTissue(color))
4578 {
4579     adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
4580 }
4581 else
4582 {
4583     adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Background");
4584 }
4585
4586 adjustImage_rightSidePanel.repaint();
4587 }
4588
4589 if (cursorPosition.y>drawHeight &&
4590     cursorPosition.y<2*drawHeight &&
4591     cursorPosition.x< drawWidth )
4592 {
4593     middleScrollPane.setCursor(transparentCursor);
4594     cursorOverPanel=true;
4595
4596     imagePosition.x=cursorPosition.x*100/myProjectData.zoomFactor;
4597     imagePosition.y=cursorPosition.y-drawHeight;
4598     imagePosition.y=imagePosition.y*100/myProjectData.zoomFactor;
4599
4600     WritableRaster raster = originalImage.getRaster();
4601     color=raster.getPixel(imagePosition.x,imagePosition.y,color);
4602
4603     adjustImage_rightSidePanel.redValue.setText("Red: "+color[0]);
4604     adjustImage_rightSidePanel.greenValue.setText("Green: "+color[1]);
4605     adjustImage_rightSidePanel.blueValue.setText("Blue: "+color[2]);
4606
4607     int firstInt = color[0]+color[2];
4608     int secondInt = color[0]+color[1]+color[2];
4609     double tempDouble = (double) firstInt/secondInt;
4610     tempDouble = tempDouble*1000;
4611     tempDouble = Math.round(tempDouble);
4612     tempDouble = tempDouble/10;
4613
4614     adjustImage_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
4615     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
4616     if(myMethods.pixellIsTissue(color))
4617     {

```

```

4618         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
4619     }
4620     else
4621     {
4622         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Background");
4623     }
4624
4625     adjustImage_rightSidePanel.repaint();
4626 }
4627
4628 if (cursorPosition.y>2*drawHeight &&
4629     cursorPosition.y<3*drawHeight &&
4630     cursorPosition.x< drawWidth )
4631 {
4632     middleScrollPane.setCursor(transparentCursor);
4633     cursorOverPanel=true;
4634
4635     imagePosition.x=cursorPosition.x*100/myProjectData.zoomFactor;
4636     imagePosition.y=cursorPosition.y-2*drawHeight;
4637     imagePosition.y=imagePosition.y*100/myProjectData.zoomFactor;
4638
4639     WritableRaster raster = originalImage.getRaster();
4640     color=raster.getPixel(imagePosition.x,imagePosition.y,color);
4641
4642     adjustImage_rightSidePanel.redValue.setText("Red: "+color[0]);
4643     adjustImage_rightSidePanel.greenValue.setText("Green: "+color[1]);
4644     adjustImage_rightSidePanel.blueValue.setText("Blue: "+color[2]);
4645
4646     int firstInt = color[0]+color[2];
4647     int secondInt = color[0]+color[1]+color[2];
4648     double tempDouble = (double) firstInt/secondInt;
4649     tempDouble = tempDouble*1000;
4650     tempDouble = Math.round(tempDouble);
4651     tempDouble = tempDouble/10;
4652
4653     adjustImage_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
4654     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
4655     if(myMethods.pixelIsTissue(color))
4656     {
4657         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
4658     }
4659     else
4660     {
4661         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Background");
4662     }
4663
4664     adjustImage_rightSidePanel.repaint();
4665 }
4666
4667 currentPoint=event.getPoint();
4668 cursorPosX=currentPoint.x;
4669 cursorPosY=currentPoint.y;
4670
4671 circleDrawSize=size*myProjectData.zoomFactor/100;
4672
4673 circlePosX=cursorPosX-circleDrawSize;
4674 circlePosY=cursorPosY-circleDrawSize;
4675 circleSize=circleDrawSize*2;
4676
4677 adjustImage_rightSidePanel.repaint();

```



```

4678     }
4679
4680     @Override
4681     public void mouseEntered(MouseEvent event)
4682     {
4683         cursorOverPanel=true;
4684     }
4685
4686     @Override
4687     public void mouseExited(MouseEvent event)
4688     {
4689         cursorOverPanel=false;
4690
4691         adjustImage_rightSidePanel.redValue.setText("Red: ---");
4692         adjustImage_rightSidePanel.greenValue.setText("Green: ---");
4693         adjustImage_rightSidePanel.blueValue.setText("Blue: ---");
4694         adjustImage_rightSidePanel.colorValueLabel.setText("Color: ---");
4695         adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
4696         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: ---");
4697     }
4698
4699     @Override
4700     public void mousePressed(MouseEvent event) {}
4701
4702     @Override
4703     public void mouseReleased(MouseEvent event) {}
4704
4705     public void whenClickedInTree(int currentSubject, int currentSection)
4706     {
4707         myMethods.setWaitCursor();
4708
4709         openSection.resultsNeedUpdating = false;
4710
4711         subject=currentSubject;
4712         section=currentSection;
4713
4714         currentSectionData=myMethods.readSectionData(subject, section);
4715
4716         adjustImage_rightSidePanel.updateTextFields();
4717         adjustImage_rightSidePanel.colorWarningLabel.setText("");
4718         adjustImage_rightSidePanel.intensityWarningLabel.setText("");
4719
4720         this.generateOriginalImage();
4721         this.generateAdjustedImage();
4722         this.generateResultImage();
4723
4724         middleScrollPane.getViewport().add(adjustImage_middlePanel);
4725
4726         rightScrollPane.getViewport().add(adjustImage_rightSidePanel);
4727         adjustImage_middlePanel.setPreferredSize(new Dimension(originalImage.getWidth(),
4728 3*originalImage.getHeight()+300));
4729
4729         myMethods.setCustomCursor();
4730     }
4731
4732     public void whenClickedInTreeFromSameSection(int currentSubject, int currentSection)
4733     {
4734         myMethods.setWaitCursor();
4735
4736         subject=currentSubject;

```



```

4737     section=currentSection;
4738
4739     adjustImage_rightSidePanel.updateTextFields();
4740     adjustImage_rightSidePanel.colorWarningLabel.setText("");
4741     adjustImage_rightSidePanel.intensityWarningLabel.setText("");
4742
4743     this.generateOriginalImage();
4744     this.generateAdjustedImage();
4745     this.generateResultImage();
4746
4747     middleScrollPane.getViewPort().add(adjustImage_middlePanel);
4748
4749     rightScrollPane.getViewPort().add(adjustImage_rightSidePanel);
4750     adjustImage_middlePanel.setPreferredSize(new Dimension(originalImage.getWidth(),
4751 3*originalImage.getHeight()+300));
4752     myMethods.setCursor();
4753 }
4754
4755 public void whenLeftInTree()
4756 {
4757     myMethods.setCursor();
4758     myMethods.saveSectionData(subject, section, currentSectionData);
4759     if (somethingChanged) {
4760         myMethods.analyzeSection(subject, section);
4761         openSection.resultsNeedUpdating = true;
4762     }
4763     if (openSection.resultsNeedUpdating) {
4764         myMethods.updateSectionResultsObject(subject, section);
4765         myMethods.analyzeSubject(subject);
4766     }
4767     myMethods.saveSectionData(subject, section, currentSectionData);
4768     myMethods.setCursor();
4769 }
4770
4771 public void whenLeftInTreeForSameSection() {
4772     myMethods.setCursor();
4773
4774     if (somethingChanged) {
4775         myMethods.saveSectionData(subject, section, currentSectionData);
4776         myMethods.analyzeSection(subject, section);
4777         openSection.resultsNeedUpdating = true;
4778         openSection.tissuePixels = myMethods.getTissuePixels(originalImage);
4779     }
4780     myMethods.setCursor();
4781 }
4782
4783 public void newDrawPoint(Point newPoint)
4784 {
4785     somethingChanged = true;
4786
4787     WritableRaster adjustRaster = imageToAdjust.getRaster();
4788     WritableRaster resultRaster = modifiedImage.getRaster();
4789
4790     for (int i=0;i<size;i++)
4791         for (int j=0;j<size;j++)
4792         {
4793             if (Math.sqrt(i*i+j*j)<size)
4794             {
4795                 try

```

```

4796         {
4797             if(addBackground)
4798             {
4799                 currentSectionData.markedAsBackground[newPoint.x-i][newPoint.y-j]=true;
4800                 currentSectionData.markedAsTissue[newPoint.x-i][newPoint.y-j]=false;
4801                 adjustRaster.setPixel(newPoint.x-i,newPoint.y-j,
myProjectData.addBackgroundColor);
4802                 resultRaster.setPixel(newPoint.x-i,newPoint.y-j, myProjectData.addBackgroundColor);
4803             }
4804             else
4805             {
4806                 currentSectionData.markedAsBackground[newPoint.x-i][newPoint.y-j]=false;
4807                 currentSectionData.markedAsTissue[newPoint.x-i][newPoint.y-j]=true;
4808                 adjustRaster.setPixel(newPoint.x-i,newPoint.y-j, myProjectData.addTissueColor);
4809                 resultRaster.setPixel(newPoint.x-i,newPoint.y-j, myProjectData.addTissueColor);
4810             }
4811         } catch(Exception error){ }
4812
4813     try
4814     {
4815         if(addBackground)
4816         {
4817             currentSectionData.markedAsBackground[newPoint.x-i][newPoint.y+j]=true;
4818             currentSectionData.markedAsTissue[newPoint.x-i][newPoint.y+j]=false;
4819             adjustRaster.setPixel(newPoint.x-i,newPoint.y+j,
myProjectData.addBackgroundColor);
4820             resultRaster.setPixel(newPoint.x-i,newPoint.y+j,
myProjectData.addBackgroundColor);
4821         }
4822         else
4823         {
4824             currentSectionData.markedAsBackground[newPoint.x-i][newPoint.y+j]=false;
4825             currentSectionData.markedAsTissue[newPoint.x-i][newPoint.y+j]=true;
4826             adjustRaster.setPixel(newPoint.x-i,newPoint.y+j, myProjectData.addTissueColor);
4827             resultRaster.setPixel(newPoint.x-i,newPoint.y+j, myProjectData.addTissueColor);
4828         }
4829     } catch(Exception error){ }
4830
4831     try
4832     {
4833         if(addBackground)
4834         {
4835             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y-j]=true;
4836             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y-j]=false;
4837             adjustRaster.setPixel(newPoint.x+i,newPoint.y-j,
myProjectData.addBackgroundColor);
4838             resultRaster.setPixel(newPoint.x+i,newPoint.y-j,
myProjectData.addBackgroundColor);
4839         }
4840         else
4841         {
4842             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y-j]=false;
4843             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y-j]=true;
4844             adjustRaster.setPixel(newPoint.x+i,newPoint.y-j, myProjectData.addTissueColor);
4845             resultRaster.setPixel(newPoint.x+i,newPoint.y-j, myProjectData.addTissueColor);
4846         }
4847     } catch(Exception error){ }
4848
4849     try
4850     {

```

```

4851         if(addBackground)
4852         {
4853             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y+j]=true;
4854             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y+j]=false;
4855             adjustRaster.setPixel(newPoint.x+i,newPoint.y+j,
myProjectData.addBackgroundColor);
4856             resultRaster.setPixel(newPoint.x+i,newPoint.y+j,
myProjectData.addBackgroundColor);
4857         }
4858         else
4859         {
4860             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y+j]=false;
4861             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y+j]=true;
4862             adjustRaster.setPixel(newPoint.x+i,newPoint.y+j, myProjectData.addTissueColor);
4863             resultRaster.setPixel(newPoint.x+i,newPoint.y+j, myProjectData.addTissueColor);
4864         }
4865     } catch(Exception error){ }
4866 }
4867 }
4868 }
4869
4870 public void generateOriginalImage()
4871 {
4872     try
4873     {
4874         File originalImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
4875         originalImage = ImageIO.read(originalImageFile);
4876     } catch (IOException event) { }
4877 }
4878
4879 public void generateAdjustedImage()
4880 {
4881     try
4882     {
4883         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
4884         imageToAdjust = ImageIO.read(adjustedImageFile);
4885     } catch (IOException event) { }
4886
4887     myMethods.changePixelsInImage(imageToAdjust, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
4888     myMethods.changePixelsInImage(imageToAdjust, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
4889 }
4890
4891 public void generateResultImage()
4892 {
4893     try
4894     {
4895         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
4896         modifiedImage = ImageIO.read(adjustedImageFile);
4897     } catch (IOException event) { }
4898
4899     boolean[][] tissuePixels = myMethods.getTissuePixels(modifiedImage);
4900
4901     WritableRaster destinationRaster=modifiedImage.getRaster();
4902
4903     int[] red = {255,0,0};
4904
4905     for (int i=0; i<modifiedImage.getWidth();i++)
4906         for (int j=0; j<modifiedImage.getHeight();j++)

```

```

4907         {
4908             if (tissuePixels[i][j])
4909             {
4910                 destinationRaster.setPixel(i, j, red);
4911             }
4912         }
4913     }
4914
4915     @Override
4916     public void paint(Graphics g)
4917     {
4918         super.paint(g);
4919
4920         int imageWidth = originalImage.getWidth();
4921         int imageHeight = originalImage.getHeight();
4922
4923         int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4924         int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4925
4926         adjustImage_middlePanel.setPreferredSize(new Dimension(drawWidth+300, 3*drawHeight));
4927
4928         g.drawString("Original image", drawWidth+10, 15);
4929         g.drawImage(originalImage, 0, 0, drawWidth, drawHeight, this);
4930
4931         g.drawString("Adjust this image", drawWidth+10, 15+drawHeight);
4932         g.drawImage(imageToAdjust, 0, drawHeight, drawWidth, drawHeight, this);
4933
4934         g.drawString("Detected tissue pixels", drawWidth+10, 15+2*drawHeight);
4935         g.drawImage(modifiedImage, 0, 2*drawHeight, drawWidth, drawHeight, this);
4936
4937         if (cursorOverPanel)
4938         {
4939             Graphics2D g2 = (Graphics2D) g;
4940             Color tempDrawColor = new Color (drawColor[0],drawColor[1],drawColor[2]);
4941             g2.setColor(tempDrawColor);
4942             g2.draw(new Ellipse2D.Double(circlePosX, circlePosY,
4943                 circleSize, circleSize));
4944             g2.setColor(Color.BLACK);
4945             g2.draw(new Ellipse2D.Double(circlePosX-1, circlePosY-1,
4946                 circleSize+2, circleSize+2));
4947         }
4948
4949         repaint();
4950         revalidate();
4951     }
4952
4953     public void setTransparentCursor()
4954     {
4955         //Windows can't use a cursor larger than 32x32 pixels.
4956         //To get around this problem this method creates a transparent cursor.
4957         //Whenever the cursor is over an image the transparent cursor is used
4958         //and a circle is drawn at the cursor position.
4959         Toolkit toolkit = Toolkit.getDefaultToolkit();
4960         BufferedImage bufferedImage = new BufferedImage(32,32,BufferedImage.TYPE_INT_ARGB);
4961         WritableRaster raster = bufferedImage.getRaster();
4962         int[] transparent = {0,0,0,0};
4963
4964         for (int i=0;i<32;i++)
4965             for (int j=0;j<32;j++)
4966                 raster.setPixel(i,j,transparent);

```

```

4967
4968     Image tempImage = bufferedImage;
4969     Point hotSpot = new Point();
4970     hotSpot.x=0;
4971     hotSpot.y=0;
4972     transparentCursor = toolkit.createCustomCursor(tempImage , hotSpot, "Circel");
4973 }
4974
4975 }
4976
4977 @SuppressWarnings("LeakingThisInConstructor")
4978 public class AdjustImage_RightSidePanel extends JPanel implements ActionListener
4979 {
4980     JButton helpButton;
4981     JButton revertToOriginalButton;
4982     JButton updateImageButton;
4983     JButton increaseButton;
4984     JButton decreaseButton;
4985     JButton addTissueButton;
4986     JButton removeTissueButton;
4987
4988     JLabel colorTreshholdLabel = new JLabel("Color treshhold");
4989     JTextField colorTreshholdTextField = new JTextField(""+myProjectData.colorTreshholdGeneral, 8);
4990     JLabel colorWarningLabel = new JLabel("");
4991     JLabel intensityTreshholdLabel = new JLabel("Intensity treshhold");
4992     JTextField intensityTreshholdTextField = new JTextField(""+myProjectData.intensityTreshholdGeneral,
4993 8);
4994     JLabel intensityWarningLabel = new JLabel("");
4995
4996     int size=10;
4997     int[] white = {255,255,255};
4998     int[] tissueColor = {255,0,255};
4999
5000     JLabel redValue = new JLabel("Red: ---");
5001     JLabel greenValue = new JLabel("Green: ---");
5002     JLabel blueValue = new JLabel("Blue: ---");
5003
5004     JLabel colorValueLabel = new JLabel("Color: ");
5005     JLabel intensityValueLabel = new JLabel("Intensity: ");
5006     JLabel resultLabel = new JLabel("Pixel is: ");
5007
5008     JLabel testLabel = new JLabel("");
5009
5010     public AdjustImage_RightSidePanel()
5011     {
5012         helpButton = new JButton ("Help");
5013         revertToOriginalButton = new JButton ("Revert To Original");
5014         updateImageButton = new JButton("Update Image");
5015         decreaseButton = new JButton("- Draw Size");
5016         increaseButton = new JButton("+ Draw Size");
5017         addTissueButton = new JButton("Add tissue");
5018         removeTissueButton = new JButton("Remove tissue");
5019
5020         this.setLayout(new GridBagLayout());
5021         GridBagConstraints c = new GridBagConstraints();
5022
5023         c.gridx = 0;
5024         c.gridy = 0;
5025         c.gridwidth = 1;
5026         c.insets = new Insets(10,5,20,5);

```

```
5026     c.anchor = GridBagConstraints.PAGE_START;
5027     this.add(helpButton , c);
5028
5029     c.insets = new Insets(5,5,5,5);
5030     c.gridy ++;
5031     this.add(revertToOriginalButton , c);
5032
5033     c.gridy ++;
5034     this.add(updateImageButton, c);
5035
5036     c.insets = new Insets(20,5,5,5);
5037     c.gridy ++;
5038     this.add(decreaseButton, c);
5039
5040     c.insets = new Insets(5,5,5,5);
5041     c.gridy ++;
5042     this.add(increaseButton, c);
5043
5044     c.gridy ++;
5045     this.add(addTissueButton, c);
5046
5047     c.gridy ++;
5048     this.add(removeTissueButton, c);
5049
5050     c.gridy++;
5051     c.insets = new Insets(25,5,5,5);
5052     this.add(colorTresholdLabel, c);
5053
5054     c.gridy++;
5055     c.insets = new Insets(5,5,5,5);
5056     this.add(colorTresholdTextField, c);
5057
5058     c.gridy++;
5059     this.add(colorWarningLabel, c);
5060
5061     c.gridy++;
5062     this.add(intensityTresholdLabel, c);
5063
5064     c.insets = new Insets(5,5,25,5);
5065     c.gridy++;
5066     this.add(intensityTresholdTextField, c);
5067
5068     c.gridy++;
5069     c.insets = new Insets(5,5,5,5);
5070     this.add(intensityWarningLabel, c);
5071
5072     c.gridy++;
5073     c.insets = new Insets(0,5,15,5);
5074     this.add(redValue, c);
5075
5076     c.gridy++;
5077     this.add(greenValue, c);
5078
5079     c.gridy++;
5080     this.add(blueValue, c);
5081
5082     c.gridy++;
5083     c.insets = new Insets(25,5,5,5);
5084     this.add(colorValueLabel,c);
5085
```

```

5086     c.gridy++;
5087     c.insets = new Insets(5,5,5,5);
5088     this.add(intensityValueLabel,c);
5089
5090     c.gridy++;
5091     c.weighty = 1;
5092     this.add(resultLabel, c);
5093
5094     helpButton.addActionListener(this);
5095     revertToOriginalButton.addActionListener(this);
5096     updateImageButton.addActionListener(this);
5097     decreaseButton.addActionListener(this);
5098     increaseButton.addActionListener(this);
5099     addTissueButton.addActionListener(this);
5100     removeTissueButton.addActionListener(this);
5101     colorTresholdTextField.addActionListener(this);
5102     intensityTresholdTextField.addActionListener(this);
5103 }
5104
5105 @Override
5106 public void actionPerformed(ActionEvent e)
5107 {
5108     if (e.getSource() == helpButton)
5109     {
5110         String helpText = myMethods.getHelpTextAdjustImage();
5111         helpDialog.newText(helpText);
5112     }
5113
5114     if (e.getSource() == revertToOriginalButton)
5115     {
5116         myMethods.setBooleanArrayValue(currentSectionData.markedAsBackground, false);
5117         myMethods.setBooleanArrayValue(currentSectionData.markedAsTissue, false);
5118
5119         openSection.tissuePixels =
5120 myMethods.getTissuePixels(adjustImage_middlePanel.originalImage);
5121         openSection.originalImage =
5122 myMethods.copyBufferedImage(adjustImage_middlePanel.originalImage);
5123         adjustImage_middlePanel.generateOriginalImage();
5124         adjustImage_middlePanel.generateAdjustedImage();
5125         adjustImage_middlePanel.generateResultImage();
5126         adjustImage_middlePanel.repaint();
5127         adjustImage_middlePanel.somethingChanged = true;
5128     }
5129
5130     if (e.getSource() == updateImageButton)
5131     {
5132         adjustImage_middlePanel.generateAdjustedImage();
5133         adjustImage_middlePanel.generateResultImage();
5134         adjustImage_middlePanel.repaint();
5135         adjustImage_middlePanel.somethingChanged = true;
5136     }
5137
5138     if (e.getSource() == decreaseButton)
5139     {
5140         size=adjustImage_middlePanel.size;
5141
5142         if (size<5)
5143         {
5144             size -=1;

```

```

5144     }
5145     else if (size<12)
5146     {
5147         size -= 3;
5148     }
5149     else
5150     {
5151         size -= 6;
5152     }
5153
5154     if (size<1) size=1;
5155     adjustImage_middlePanel.size =size;
5156 }
5157
5158 if (e.getSource() == increaseButton)
5159 {
5160     size=adjustImage_middlePanel.size;
5161
5162     if (size<5)
5163     {
5164         size +=1;
5165     }
5166     else if (size<12)
5167     {
5168         size += 3;
5169     }
5170     else
5171     {
5172         size += 6;
5173     }
5174
5175     adjustImage_middlePanel.size =size;
5176 }
5177
5178 if (e.getSource() == addTissueButton)
5179 {
5180     adjustImage_middlePanel.drawColor =tissueColor;
5181     adjustImage_middlePanel.addBackground=false;
5182 }
5183 if (e.getSource() == removeTissueButton)
5184 {
5185     adjustImage_middlePanel.drawColor = white;
5186     adjustImage_middlePanel.addBackground=true;
5187 }
5188
5189 if (e.getSource()==colorTresholdTextField)
5190 {
5191     upperScrollPane.requestFocus();
5192
5193     try
5194     {
5195         double tempDouble = Double.parseDouble(colorTresholdTextField.getText());
5196         if (tempDouble>=0 && tempDouble<=100)
5197         {
5198             currentSectionData.colorTreshold = tempDouble;
5199             colorWarningLabel.setText("");
5200         }
5201         else
5202         {
5203             colorWarningLabel.setForeground(Color.red);

```



```

5204         colorWarningLabel.setText("Use range 0-100");
5205     }
5206 }catch(Exception error)
5207 {
5208     colorWarningLabel.setForeground(Color.red);
5209     colorWarningLabel.setText("Not a number");
5210 }
5211
5212 adjustImage_middlePanel.generateResultImage();
5213 adjustImage_middlePanel.repaint();
5214 adjustImage_middlePanel.somethingChanged = true;
5215 }
5216 if (e.getSource()==intensityTresholdTextField)
5217 {
5218     upperScrollPane.requestFocus();
5219
5220     try
5221     {
5222         double tempDouble = Double.parseDouble(intensityTresholdTextField.getText());
5223         if (tempDouble>=0 && tempDouble<=765)
5224         {
5225             currentSectionData.intensityTreshold = tempDouble;
5226             intensityWarningLabel.setText("");
5227         }
5228         else
5229         {
5230             intensityWarningLabel.setForeground(Color.red);
5231             intensityWarningLabel.setText("Use range 0-765");
5232         }
5233     }catch(Exception error)
5234     {
5235         intensityWarningLabel.setForeground(Color.red);
5236         intensityWarningLabel.setText("Not a number");
5237     }
5238
5239     adjustImage_middlePanel.generateResultImage();
5240     adjustImage_middlePanel.repaint();
5241     adjustImage_middlePanel.somethingChanged = true;
5242 }
5243 }
5244
5245 public void updateTextFields()
5246 {
5247     colorTresholdTextField.setText(""+currentSectionData.colorTreshold);
5248     intensityTresholdTextField.setText(""+currentSectionData.intensityTreshold);
5249 }
5250 }
5251
5252 @SuppressWarnings("LeakingThisInConstructor")
5253 private class LR_DivideMiddlePanel extends JPanel implements MouseListener, MouseMotionListener
5254 {
5255     //Lets the user draw a line which separates the left and right hemispheres
5256     int subject=0;
5257     int section=0;
5258
5259     BufferedImage LR_LinesImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5260     BufferedImage LR_ResultsImage = new
BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5261
5262     boolean[][] dividePixels;

```

```

5263
5264     boolean button1pressed = false;
5265     boolean button3pressed = false;
5266
5267     public LR_DivideMiddlePanel()
5268     {
5269         addMouseListener(this);
5270         addMouseMotionListener(this);
5271     }
5272
5273     @Override
5274     public void mouseClicked(MouseEvent event)
5275     {
5276         if (event.getButton() == MouseEvent.BUTTON1)
5277         {
5278             this.newDrawPoint(event.getPoint());
5279         }
5280         if (event.getButton() == MouseEvent.BUTTON3)
5281         {
5282             lr_divideMiddlePanel.generateLR_LinesImage();
5283             lr_divideMiddlePanel.generateLR_AreasImage();
5284             lr_divideMiddlePanel.repaint();
5285         }
5286     }
5287
5288     @Override
5289     public void mouseDragged(MouseEvent event)
5290     {
5291         if (button1pressed)
5292         {
5293             this.newDrawPoint(event.getPoint());
5294         }
5295         if (button3pressed)
5296         {
5297             lr_divideMiddlePanel.generateLR_LinesImage();
5298             lr_divideMiddlePanel.generateLR_AreasImage();
5299             lr_divideMiddlePanel.repaint();
5300         }
5301     }
5302
5303     @Override
5304     public void mouseMoved(MouseEvent event) {}
5305
5306     @Override
5307     public void mousePressed(MouseEvent event) {
5308         if (event.getButton() == MouseEvent.BUTTON1)
5309         {
5310             button1pressed = true;
5311         }
5312         if (event.getButton() == MouseEvent.BUTTON3)
5313         {
5314             button3pressed = true;
5315         }
5316     }
5317
5318     @Override
5319     public void mouseReleased(MouseEvent event){
5320         if (event.getButton() == MouseEvent.BUTTON1)
5321         {
5322             button1pressed = false;

```

```

5323     }
5324     if (event.getButton() == MouseEvent.BUTTON3)
5325     {
5326         button3pressed = false;
5327     }
5328 }
5329
5330 @Override
5331 public void mouseEntered(MouseEvent event) {}
5332
5333 @Override
5334 public void mouseExited(MouseEvent event) {}
5335
5336 public void whenClickedInTree(int currentSubject, int currentSection)
5337 {
5338     myMethods.setWaitCursor();
5339
5340     openSection.resultsNeedUpdating = false;
5341
5342     subject=currentSubject;
5343     section=currentSection;
5344     currentSectionData=myMethods.readSectionData(subject, section);
5345
5346     this.generateLR_LinesImage();
5347     this.generateLR_AreasImage();
5348
5349     rightScrollPane.getViewport().add(lr_DivideRightSidePanel);
5350     middleScrollPane.getViewport().add(lr_divideMiddlePanel);
5351     lr_divideMiddlePanel.setPreferredSize(new Dimension(LR_LinesImage.getWidth(),
5352 2*LR_LinesImage.getHeight()+100));
5352
5353     myMethods.setCustomCursor();
5354 }
5355
5356 public void whenClickedInTreeFromSameSection(int currentSubject, int currentSection) {
5357     myMethods.setWaitCursor();
5358
5359     subject=currentSubject;
5360     section=currentSection;
5361
5362     this.generateLR_LinesImage();
5363     this.generateLR_AreasImage();
5364     rightScrollPane.getViewport().add(lr_DivideRightSidePanel);
5365     middleScrollPane.getViewport().add(lr_divideMiddlePanel);
5366     lr_divideMiddlePanel.setPreferredSize(new Dimension(LR_LinesImage.getWidth(),
5367 2*LR_LinesImage.getHeight()+100));
5367
5368     myMethods.setCustomCursor();
5369 }
5370
5371 public void whenLeftInTree()
5372 {
5373     //Calculates areas and saves the data before leaving
5374     myMethods.setWaitCursor();
5375     this.generateLR_AreasImage();
5376     myMethods.saveSectionData(subject, section, currentSectionData);
5377
5378     if (openSection.resultsNeedUpdating) {
5379         myMethods.updateSectionResultsObject(subject, section);
5380         myMethods.analyzeSubject(subject);

```

```

5381     }
5382
5383     myMethods.setCustomCursor();
5384 }
5385
5386 public void whenLeftInTreeForSameSection() {
5387     myMethods.setWaitCursor();
5388     this.generateLR_AreasImage();
5389     myMethods.setCustomCursor();
5390 }
5391
5392 public void newDrawPoint(Point newPoint)
5393 {
5394     newPoint.x=100*newPoint.x/myProjectData.zoomFactor;
5395     newPoint.y=100*newPoint.y/myProjectData.zoomFactor;
5396
5397     if (newPoint.x<LR_LinesImage.getWidth() && newPoint.y<LR_LinesImage.getHeight()
5398         && newPoint.x>-1 && newPoint.y>-1)
5399     {
5400         currentSectionData.pointList.add(newPoint);
5401
5402         int size = currentSectionData.pointList.size();
5403         if (size>1) {
5404             Point lastPoint = (Point) currentSectionData.pointList.get(size-2);
5405             Point newestPoint = (Point) currentSectionData.pointList.get(size-1);
5406             dividePixels = myMethods.drawLineBoolean(dividePixels, lastPoint, newestPoint);
5407         }
5408
5409         boolean[][] linePixels = myMethods.widenLine(dividePixels, myProjectData.lineWidth);
5410         myMethods.changePixelsInImage(LR_LinesImage, linePixels, myProjectData.lineColor);
5411
5412         repaint();
5413     }
5414 }
5415
5416 public void clearLines()
5417 {
5418     currentSectionData.pointList = new ArrayList();
5419     myMethods.setBooleanArrayValue(dividePixels, false);
5420
5421     myProjectData.numberOfTissuePixelsLeft[subject][section] =0;
5422     myProjectData.numberOfTissuePixelsRight[subject][section] =0;
5423     myProjectData.numberOfVentriclePixelsLeft[subject][section] =0;
5424     myProjectData.numberOfVentriclePixelsRight[subject][section] =0;
5425
5426     LR_LinesImage = myMethods.copyBufferedImage(openSection.originalImage);
5427     LR_ResultsImage = myMethods.copyBufferedImage(openSection.originalImage);
5428
5429     openSection.resultsNeedUpdating = true;
5430
5431     repaint();
5432 }
5433
5434 public void generateLR_LinesImage()
5435 {
5436     LR_LinesImage = myMethods.readImageFile(subject, section);
5437
5438     openSection.originalImage = myMethods.copyBufferedImage(LR_LinesImage);
5439

```

```

5440     myMethods.changePixelsInImage(LR_LinesImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5441     myMethods.changePixelsInImage(LR_LinesImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5442
5443     dividePixels = myMethods.getBooleanArrayFromPointList(currentSectionData.pointList,
5444                                     LR_LinesImage.getWidth(), LR_LinesImage.getHeight());
5445
5446     boolean[][] linePixels = myMethods.widenLine(dividePixels, myProjectData.lineWidth);
5447     myMethods.changePixelsInImage(LR_LinesImage, linePixels, myProjectData.lineColor);
5448 }
5449
5450 public void generateLR_AreasImage()
5451 {
5452     LR_ResultsImage = myMethods.copyBufferedImage(openSection.originalImage);
5453
5454     boolean[][] tissuePixels = myMethods.getTissuePixels(LR_ResultsImage);
5455
5456     openSection.tissuePixels = tissuePixels;
5457
5458     boolean[][] ventriclePixels = myMethods.newGetVentriclePixels2(tissuePixels);
5459
5460     boolean[][] isLeft = myMethods.isLeftFromArray(dividePixels);
5461
5462     myMethods.changePixelsInImage(LR_ResultsImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5463     myMethods.changePixelsInImage(LR_ResultsImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5464
5465     WritableRaster destinationRaster=LR_ResultsImage.getRaster();
5466
5467     int tissuePixelsLeft=0;
5468     int tissuePixelsRight=0;
5469     int ventriclePixelsLeft=0;
5470     int ventriclePixelsRight=0;
5471
5472     int[] black = {255,255,0};
5473     int[] red = {255,0,0};
5474     int[] blue = {0,0,255};
5475     int[] green = {0,255,0};
5476
5477     for (int i=0; i<LR_ResultsImage.getWidth();i++)
5478         for (int j=0; j<LR_ResultsImage.getHeight();j++)
5479         {
5480             if (tissuePixels[i][j] && isLeft[i][j])
5481             {
5482                 destinationRaster.setPixel(i, j, red);
5483                 tissuePixelsLeft++;
5484             }
5485             if (tissuePixels[i][j] && !isLeft[i][j])
5486             {
5487                 destinationRaster.setPixel(i, j, black);
5488                 tissuePixelsRight++;
5489             }
5490             if (ventriclePixels[i][j] && isLeft[i][j])
5491             {
5492                 destinationRaster.setPixel(i, j, blue);
5493                 ventriclePixelsLeft++;
5494             }
5495             if (ventriclePixels[i][j] && !isLeft[i][j])

```

```

5496         {
5497             destinationRaster.setPixel(i, j, green);
5498             ventriclePixelsRight++;
5499         }
5500     }
5501
5502     myProjectData.numberOfTissuePixelsLeft[subject][section] = tissuePixelsLeft;
5503     myProjectData.numberOfTissuePixelsRight[subject][section] = tissuePixelsRight;
5504
5505     myProjectData.numberOfVentriclePixelsLeft[subject][section] = ventriclePixelsLeft;
5506     myProjectData.numberOfVentriclePixelsRight[subject][section] = ventriclePixelsRight;
5507
5508     openSection.resultsNeedUpdating = true;
5509 }
5510
5511 @Override
5512 public void paint(Graphics g)
5513 {
5514     super.paint(g);
5515
5516     int imageWidth = LR_LinesImage.getWidth();
5517     int imageHeight = LR_LinesImage.getHeight();
5518
5519     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
5520     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5521
5522     lr_divideMiddlePanel.setPreferredSize(new Dimension(drawWidth+300, 2*drawHeight));
5523
5524     g.drawString("Draw line here", drawWidth+10, 15);
5525     g.drawImage(LR_LinesImage, 0,0, drawWidth, drawHeight, this);
5526
5527     g.drawString("Detected areas", drawWidth+10, drawHeight+15);
5528     g.drawImage(LR_ResultsImage,0, drawHeight, drawWidth, drawHeight, this);
5529
5530     this.revalidate();
5531 }
5532 }
5533
5534 @SuppressWarnings("LeakingThisInConstructor")
5535 public class LR_DivideRightSidePanel extends JPanel implements ActionListener
5536 {
5537     JLabel info1Label = new JLabel("Left click or drag to draw outline");
5538     JLabel info1Label2 = new JLabel("Right click to update detected areas");
5539
5540     JButton clearLinesButton;
5541     JButton updateSectionButton;
5542
5543     public LR_DivideRightSidePanel()
5544     {
5545         clearLinesButton=new JButton("Clear Lines");
5546         updateSectionButton=new JButton("Update Section");
5547
5548         this.setLayout(new GridBagLayout());
5549         GridBagConstraints c = new GridBagConstraints();
5550
5551         c.gridx = 0;
5552         c.gridy = 0;
5553         c.gridwidth = 1;
5554         c.insets = new Insets(10,5,5,5);
5555         c.anchor = GridBagConstraints.PAGE_START;

```

```

5556     this.add(info1Label, c);
5557
5558     c.gridy++;
5559     c.insets = new Insets(5,5,5,5);
5560     this.add(info1Label2, c);
5561
5562     c.gridy++;
5563     c.insets = new Insets(20,5,5,5);
5564     this.add(clearLinesButton, c);
5565
5566     c.insets = new Insets(5,5,5,5);
5567     c.gridy++;
5568     c.weighty=1;
5569     this.add(updateSectionButton, c);
5570
5571     clearLinesButton.addActionListener(this);
5572     updateSectionButton.addActionListener(this);
5573 }
5574
5575 @Override
5576 public void actionPerformed(ActionEvent e)
5577 {
5578     if (e.getSource() == clearLinesButton)
5579     {
5580         lr_divideMiddlePanel.clearLines();
5581     }
5582
5583     if (e.getSource() == updateSectionButton)
5584     {
5585         lr_divideMiddlePanel.generateLR_LinesImage();
5586         lr_divideMiddlePanel.generateLR_AreasImage();
5587         lr_divideMiddlePanel.repaint();
5588     }
5589 }
5590 }
5591
5592 @SuppressWarnings("LeakingThisInConstructor")
5593 public class UDA_MiddlePanel extends JPanel implements MouseListener, MouseMotionListener
5594 {
5595     //Lets the user draw a line around an area. When the user right clicks the
5596     //size of the area is calculated.
5597     int subject=0;
5598     int section=0;
5599     int UDAnumber=0;
5600
5601     boolean[][] linePixels;
5602
5603     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5604     BufferedImage udaPanelImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5605
5606     boolean button1pressed = false;
5607     boolean button3pressed = false;
5608
5609     public UDA_MiddlePanel()
5610     {
5611         addMouseListener(this);
5612         addMouseMotionListener(this);
5613     }
5614
5615     @Override

```

```

5616 public void mouseClicked(MouseEvent event)
5617 {
5618     if (event.getButton() == MouseEvent.BUTTON1)
5619     {
5620         this.newDrawPoint(event.getPoint());
5621     }
5622     if (event.getButton() == MouseEvent.BUTTON3)
5623     {
5624         this.newUdaOrigin(event.getPoint());
5625     }
5626 }
5627
5628 @Override
5629 public void mouseDragged(MouseEvent event)
5630 {
5631     if (button1pressed)
5632     {
5633         this.newDrawPoint(event.getPoint());
5634     }
5635     if (button3pressed)
5636     {
5637         this.newUdaOrigin(event.getPoint());
5638     }
5639 }
5640
5641 @Override
5642 public void mouseMoved(MouseEvent event) {}
5643
5644 @Override
5645 public void mousePressed(MouseEvent event) {
5646     if (event.getButton() == MouseEvent.BUTTON1)
5647     {
5648         button1pressed = true;
5649     }
5650     if (event.getButton() == MouseEvent.BUTTON3)
5651     {
5652         button3pressed = true;
5653     }
5654 }
5655
5656 @Override
5657 public void mouseEntered(MouseEvent event) {}
5658
5659 @Override
5660 public void mouseExited(MouseEvent event) {}
5661
5662 @Override
5663 public void mouseReleased(MouseEvent event) {
5664     if (event.getButton() == MouseEvent.BUTTON1)
5665     {
5666         button1pressed = false;
5667     }
5668     if (event.getButton() == MouseEvent.BUTTON3)
5669     {
5670         button3pressed = false;
5671     }
5672 }
5673
5674 public void whenClickedInTree(int currentSubject, int currentSection, int currentUDANumber)
5675 {

```



```

5676     myMethods.setWaitCursor();
5677
5678     openSection.resultsNeedUpdating = false;
5679
5680     subject=currentSubject;
5681     section=currentSection;
5682     UDANumber=currentUDANumber;
5683
5684     currentSectionData=myMethods.readSectionData(subject, section);
5685
5686     originalImage = myMethods.readImageFile(subject, section);
5687
5688     udaPanelImage = myMethods.copyBufferedImage(originalImage);
5689
5690     openSection.originalImage = originalImage;
5691     openSection.tissuePixels = myMethods.getTissuePixels(udaPanelImage);
5692
5693     this.generateUdaPanelImage();
5694     uda_middlePanel.setPreferredSize(new Dimension(udaPanelImage.getWidth(),
udaPanelImage.getHeight()));
5695     middleScrollPane.setViewportView().add(uda_middlePanel);
5696     this.repaint();
5697
5698     rightScrollPane.setViewportView().add(uda_rightSidePanel);
5699     myMethods.setCustomCursor();
5700 }
5701
5702 public void whenClickedFromSameSection(int currentSubject, int currentSection, int
currentUDANumber) {
5703     myMethods.setWaitCursor();
5704
5705     subject=currentSubject;
5706     section=currentSection;
5707     UDANumber=currentUDANumber;
5708
5709     originalImage = openSection.originalImage;
5710
5711     udaPanelImage = myMethods.copyBufferedImage(originalImage);
5712
5713     this.generateUdaPanelImage();
5714     uda_middlePanel.setPreferredSize(new Dimension(udaPanelImage.getWidth(),
udaPanelImage.getHeight()));
5715     middleScrollPane.setViewportView().add(uda_middlePanel);
5716     this.repaint();
5717
5718     rightScrollPane.setViewportView().add(uda_rightSidePanel);
5719     myMethods.setCustomCursor();
5720 }
5721
5722 public void whenLeftInTree()
5723 {
5724     myMethods.setWaitCursor();
5725     myMethods.saveSectionData(subject, section, currentSectionData);
5726     if (openSection.resultsNeedUpdating) {
5727         myMethods.updateSectionResultsObject(subject, section);
5728         myMethods.analyzeSubject(subject);
5729     }
5730     myMethods.setCustomCursor();
5731 }
5732

```

```

5733 public void whenLeftInTreeForSameSection() {}
5734
5735 public void newDrawPoint(Point newPoint) {
5736
5737     int imageHeight = originalImage.getHeight();
5738     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5739
5740     newPoint.y=newPoint.y-drawHeight;
5741     newPoint.x=100*newPoint.x/myProjectData.zoomFactor;
5742     newPoint.y=100*newPoint.y/myProjectData.zoomFactor;
5743
5744     if (newPoint.x<udaPanelImage.getWidth() && newPoint.y<udaPanelImage.getHeight()
5745         && newPoint.x>-1 && newPoint.y>-1) {
5746
5747         currentSectionData.UDApointList[UDANumber].add(newPoint);
5748         int listLength = currentSectionData.UDApointList[UDANumber].size();
5749
5750         if (listLength>1) {
5751             Point oldPoint = (Point) currentSectionData.UDApointList[UDANumber].get(listLength-2);
5752             Point newestPoint = (Point)
currentSectionData.UDApointList[UDANumber].get(listLength-1);
5753             linePixels = myMethods.drawLineBoolean(linePixels, oldPoint, newestPoint);
5754         }
5755
5756         boolean[][] outlinePixels = myMethods.widenLine(linePixels, myProjectData.lineWidth);
5757         myMethods.changePixelsInImage(udaPanelImage, outlinePixels, myProjectData.lineColor );
5758         repaint();
5759     }
5760 }
5761
5762 public void newUdaOrigin(Point newPoint)
5763 {
5764     int imageHeight = originalImage.getHeight();
5765     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5766
5767     newPoint.y=newPoint.y-drawHeight;
5768     newPoint.x=100*newPoint.x/myProjectData.zoomFactor;
5769     newPoint.y=100*newPoint.y/myProjectData.zoomFactor;
5770
5771     if (newPoint.x<udaPanelImage.getWidth() && newPoint.y<udaPanelImage.getHeight()
5772         && newPoint.x>-1 && newPoint.y>-1)
5773     {
5774         currentSectionData.udaOrigin[UDANumber] = newPoint;
5775         currentSectionData.haveUdaOrigin[UDANumber] = true;
5776         udaPanelImage = myMethods.copyBufferedImage(originalImage);
5777         this.generateUdaPanelImage();
5778     }
5779 }
5780
5781 public void clearLines()
5782 {
5783     currentSectionData.UDApointList[UDANumber] = new ArrayList();
5784     currentSectionData.haveUdaOrigin[UDANumber] = false;
5785     udaPanelImage = myMethods.copyBufferedImage(originalImage);
5786
5787     myMethods.setBooleanArrayValue(linePixels, false);
5788
5789     myProjectData.numberOfUdaPixels[subject][section][UDANumber] = 0;
5790     openSection.resultsNeedUpdating = true;
5791     repaint();

```

```

5792     }
5793
5794     public void generateUdaPanelImage() {
5795         myMethods.changePixelsInImage(originalImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5796         myMethods.changePixelsInImage(originalImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5797
5798         udaPanelImage = myMethods.copyBufferedImage(originalImage);
5799         myMethods.changePixelsInImage(udaPanelImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5800         myMethods.changePixelsInImage(udaPanelImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5801
5802         linePixels =
myMethods.getBooleanArrayFromPointList(currentSectionData.UDApoinList[UDANumber],
5803                                         currentSectionData.imageWidth,
5804                                         currentSectionData.imageHeight);
5805
5806         if (currentSectionData.haveUdaOrigin[UDANumber])
5807         {
5808             Point startPosition = currentSectionData.udaOrigin[UDANumber];
5809             boolean[][] udaPixels = myMethods.newFillInside(linePixels, startPosition);
5810
5811             boolean includeTissue = myProjectData.includeTissue[UDANumber];
5812             boolean includeBackground = myProjectData.includeBackground[UDANumber];
5813
5814             for (int i=0; i<udaPanelImage.getWidth();i++)
5815                 for (int j=0; j<udaPanelImage.getHeight();j++)
5816                 {
5817                     if (openSection.tissuePixels[i][j] && !includeTissue) udaPixels[i][j]=false;
5818                     if (!openSection.tissuePixels[i][j] && !includeBackground) udaPixels[i][j]=false;
5819                 }
5820
5821             int[] green = {0,255,0};
5822             myMethods.changePixelsInImage(udaPanelImage, udaPixels, green );
5823
5824             int numberOfUdaPixels=myMethods.countPixels(udaPixels);
5825             myProjectData.numberOfUdaPixels[subject][section][UDANumber] = numberOfUdaPixels;
5826         }
5827
5828         boolean[][] outlinePixels = myMethods.widenLine(linePixels, myProjectData.lineWidth);
5829
5830         myMethods.changePixelsInImage(udaPanelImage, outlinePixels, myProjectData.lineColor );
5831         openSection.resultsNeedUpdating = true;
5832     }
5833
5834     @Override
5835     public void paint(Graphics g)
5836     {
5837         super.paint(g);
5838
5839         int imageWidth = udaPanelImage.getWidth();
5840         int imageHeight = udaPanelImage.getHeight();
5841
5842         int drawWidth = imageWidth*myProjectData.zoomFactor/100;
5843         int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5844
5845         uda_middlePanel.setPreferredSize(new Dimension(drawWidth+300, 2*drawHeight));
5846

```

```

5847     g.drawString("Original image", drawWidth+10, 15);
5848     g.drawImage(originalImage,0,0, drawWidth, drawHeight, this);
5849
5850     g.drawString("Draw here", drawWidth+10, drawHeight+15);
5851     g.drawImage(udaPanelImage,0,drawHeight, drawWidth, drawHeight, this);
5852
5853     repaint();
5854     revalidate();
5855 }
5856 }
5857
5858 @SuppressWarnings("LeakingThisInConstructor")
5859 public class UDA_RightSidePanel extends JPanel implements ActionListener
5860 {
5861     JButton helpButton;
5862
5863     JLabel info1Label = new JLabel("Left click to draw outline");
5864     JLabel info2Label = new JLabel("Right click inside area");
5865     JLabel info3Label = new JLabel("when outline is complete");
5866
5867     JButton clearLinesButton;
5868
5869     public UDA_RightSidePanel()
5870     {
5871         helpButton=new JButton("Help");
5872         clearLinesButton=new JButton("Clear Lines");
5873
5874         this.setLayout(new GridBagLayout());
5875         GridBagConstraints c = new GridBagConstraints();
5876
5877         c.gridx = 0;
5878         c.gridy = 0;
5879         c.gridwidth = 1;
5880         c.insets = new Insets(10,5,20,5);
5881         c.anchor = GridBagConstraints.PAGE_START;
5882         this.add(helpButton, c);
5883
5884         c.gridy++;
5885         c.insets = new Insets(10,5,5,5);
5886         this.add(info1Label, c);
5887
5888         c.gridy++;
5889         c.insets = new Insets(20,5,5,5);
5890         this.add(info2Label, c);
5891
5892         c.gridy++;
5893         c.insets = new Insets(5,5,5,5);
5894         this.add(info3Label, c);
5895
5896         c.insets = new Insets(20,5,5,5);
5897         c.gridy++;
5898         c.weighty=1;
5899         this.add(clearLinesButton, c);
5900
5901         helpButton.addActionListener(this);
5902         clearLinesButton.addActionListener(this);
5903     }
5904
5905     @Override
5906     public void actionPerformed(ActionEvent e)

```

```

5907     {
5908         if (e.getSource() == helpButton)
5909         {
5910             String helpText = myMethods.getHelpTextUDA();
5911             helpDialog.newText(helpText);
5912         }
5913
5914         if (e.getSource() == clearLinesButton)
5915         {
5916             uda_middlePanel.clearLines();
5917         }
5918     }
5919 }
5920
5921 static public class ProjectData implements Serializable
5922 {
5923     //One object of this class is created and saved to disk.
5924     String projectLocation;
5925
5926     int[][] positionInProject;
5927
5928     double conversionFactor=1;
5929     double conversionNumber=1;
5930     boolean conversionIsDefined=false;
5931
5932     //Keeps track of whether the levels for a
5933     //subject are continuously increasing or decreasing
5934     boolean[] continousLevels;
5935     boolean allLevelsContinous=true;
5936
5937     int numberOfSubjects = 0;
5938     int totalNumberOfSections=0;
5939     int numberOfLevels=0;
5940     String[] subjectNameList;
5941
5942     boolean[][] sectionExists;
5943     String[][] sectionName;
5944
5945     double[][] bregmaLevels;
5946
5947     //Needed in the conversion panel
5948     Point firstConversionPoint = new Point(0,0);
5949     Point secondConversionPoint = new Point(0,0);
5950     int distancePixels=0;
5951     double distanceMM = 0;
5952
5953     int numberOfUDA;
5954     String[] UDAnames;
5955     boolean[] includeTissue;
5956     boolean[] includeBackground;
5957
5958     int[][] numberOfTissuePixels;
5959     int[][] numberOfTissuePixelsLeft;
5960     int[][] numberOfTissuePixelsRight;
5961
5962     double[][] tissueArea;
5963     double[][] leftTissueArea;
5964     double[][] rightTissueArea;
5965
5966     int[][] numberOfVentriclePixels;

```

```

5967     int[][] numberOfVentriclePixelsLeft;
5968     int[][] numberOfVentriclePixelsRight;
5969
5970     double[] tissueVolume;
5971     double[] tissueVolumeLeft;
5972     double[] tissueVolumeRight;
5973
5974     double[] ventricleVolume;
5975     double[] ventricleVolumeLeft;
5976     double[] ventricleVolumeRight;
5977
5978     int[][][] numberOfUdaPixels;
5979     double[][] udaVolume;
5980
5981     int zoomFactor=100;
5982     int lineWidth=2;
5983     int[] lineColor = {0,0,0};
5984
5985     double colorTresholdGeneral=70;
5986     double intensityTresholdGeneral=50;
5987
5988     Object[][] sectionResultsObject;
5989     Object[][] roundedSectionResultsObject;
5990
5991     int imageWidth=1;
5992     int imageHeight=1;
5993
5994     int[] addTissueColor = {255,0,255};
5995     int[] addBackgroundColor = {255,255,255};
5996
5997     boolean allImagesOfSameSize;
5998
5999     ProjectData(int newNumberOfSubject, int newNumberOfLevels, int newNumberOfUda, int
newTotalNumberOfSections)
6000     {
6001         numberOfSubjects = newNumberOfSubject;
6002         numberOfLevels = newNumberOfLevels;
6003         numberOfUDA = newNumberOfUda;
6004         totalNumberOfSections = newTotalNumberOfSections;
6005
6006         subjectNameList = new String[newNumberOfSubject];
6007
6008         continousLevels = new boolean[newNumberOfSubject];
6009
6010         positionInProject = new int[newNumberOfSubject][newNumberOfLevels];
6011
6012         sectionExists = new boolean[newNumberOfSubject][newNumberOfLevels];
6013         for (int i =0; i<newNumberOfSubject;i++)
6014         {
6015             continousLevels[i]=true;
6016             for (int j=0; j<newNumberOfLevels; j++)
6017             {
6018                 sectionExists[i][j]=false;
6019             }
6020         }
6021
6022         sectionName = new String[newNumberOfSubject][newNumberOfLevels];
6023
6024         bregmaLevels = new double[newNumberOfSubject][newNumberOfLevels];
6025

```

```

6026     UDAnames = new String[newNumberOfUda];
6027     includeTissue = new boolean[newNumberOfUda];
6028     includeBackground = new boolean[newNumberOfUda];
6029
6030     numberOfTissuePixels = new int[newNumberOfSubject][newNumberOfLevels];
6031     numberOfTissuePixelsLeft = new int[newNumberOfSubject][newNumberOfLevels];
6032     numberOfTissuePixelsRight = new int[newNumberOfSubject][newNumberOfLevels];
6033
6034     tissueArea = new double [newNumberOfSubject][newNumberOfLevels];
6035     leftTissueArea = new double [newNumberOfSubject][newNumberOfLevels];
6036     rightTissueArea = new double [newNumberOfSubject][newNumberOfLevels];
6037
6038     numberOfVentriclePixels = new int[newNumberOfSubject][newNumberOfLevels];
6039     numberOfVentriclePixelsLeft = new int[newNumberOfSubject][newNumberOfLevels];
6040     numberOfVentriclePixelsRight = new int[newNumberOfSubject][newNumberOfLevels];
6041
6042     tissueVolume = new double[newNumberOfSubject];
6043     tissueVolumeLeft = new double[newNumberOfSubject];
6044     tissueVolumeRight = new double[newNumberOfSubject];
6045
6046     ventricleVolume = new double[newNumberOfSubject];
6047     ventricleVolumeLeft = new double[newNumberOfSubject];
6048     ventricleVolumeRight = new double[newNumberOfSubject];
6049
6050     numberOfUdaPixels = new int[newNumberOfSubject][newNumberOfLevels][newNumberOfUda];
6051     udaVolume = new double[newNumberOfSubject][newNumberOfUda];
6052
6053     sectionResultsObject = new Object[totalNumberOfSections][15+2*numberOfUDA];
6054     roundedSectionResultsObject = new Object[totalNumberOfSections][15+2*numberOfUDA];
6055 }
6056 }
6057
6058 static public class SectionData implements Serializable
6059 {
6060     //One object for each section in the project is created and saved to disk
6061     int highestNumberOfUDA=100;
6062
6063     int imageWidth=0;
6064     int imageHeight=0;
6065
6066     Point[] udaOrigin = new Point[highestNumberOfUDA];
6067     boolean[] haveUdaOrigin = new boolean[highestNumberOfUDA];
6068
6069     double colorTreshold=75;
6070     double intensityTreshold=50;
6071
6072     ArrayList pointList = new ArrayList();
6073
6074     ArrayList[] UDApointList;
6075
6076     boolean[][] markedAsBackground;
6077     boolean[][] markedAsTissue;
6078
6079     public SectionData(int imageWidht, int imageHeight)
6080     {
6081         for (int i=0; i<haveUdaOrigin.length; i++) haveUdaOrigin[i]=false;
6082
6083         markedAsBackground= new boolean[imageWidht][imageHeight];
6084         markedAsTissue= new boolean[imageWidht][imageHeight];
6085

```



```

6086     for (int i=0; i<imageWidth; i++)
6087         for (int j=0; j<imageHeight; j++)
6088         {
6089             markedAsBackground[i][j]=false;
6090             markedAsTissue[i][j]=false;
6091         }
6092     }
6093 }
6094 }
6095
6096 public class OpenSection {
6097
6098     //Holds this data to avoid recreating them when swithcing between panels
6099     //within the same section.
6100
6101     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
6102     boolean[][] tissuePixels= new boolean[100][100];
6103     boolean resultsNeedUpdating = false;
6104 }
6105
6106 public class MyMethods
6107 {
6108     //Sets the normal cursor
6109     public void setCustomCursor()
6110     {
6111         buttonPanel.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
6112         leftScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
6113         middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
6114         rightScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
6115     }
6116
6117     //The hourglass cursor is mainly used when swithing between panels
6118     public void setWaitCursor()
6119     {
6120         buttonPanel.setCursor(new Cursor(Cursor.WAIT_CURSOR));
6121         leftScrollPane.setCursor(new Cursor(Cursor.WAIT_CURSOR));
6122         middleScrollPane.setCursor(new Cursor(Cursor.WAIT_CURSOR));
6123         rightScrollPane.setCursor(new Cursor(Cursor.WAIT_CURSOR));
6124     }
6125
6126     //They keys A, S, D and W can be used to zoom in and out and move up
6127     //and down in the tree. This is blocked when a text field has the focus.
6128     public boolean textFieldIsFocusOwner()
6129     {
6130         boolean textFieldIsFocusOwner=false;
6131
6132         if (tissueDetection_rightSidePanel.colorTresholdTextField.isFocusOwner())
        textFieldIsFocusOwner=true;
6133         if (tissueDetection_rightSidePanel.intensityTresholdTextField.isFocusOwner())
        textFieldIsFocusOwner=true;
6134         if (conversion_rightSidePanel.distancemmTextField.isFocusOwner()) textFieldIsFocusOwner=true;
6135         if (section_rightSidePanel.bregmaLevelTextField.isFocusOwner()) textFieldIsFocusOwner=true;
6136         if (adjustImage_rightSidePanel.colorTresholdTextField.isFocusOwner())
        textFieldIsFocusOwner=true;
6137         if (adjustImage_rightSidePanel.intensityTresholdTextField.isFocusOwner())
        textFieldIsFocusOwner=true;
6138
6139         return textFieldIsFocusOwner;
6140     }
6141

```



```

6142 //Makes a complete analysis of the section
6143 public void analyzeSection(int subject, int section)
6144 {
6145     BufferedImage originalImage = myMethods.readImageFile(subject, section);
6146
6147     boolean[][] tissuePixels = myMethods.getTissuePixels(originalImage);
6148
6149     boolean[][] ventriclePixels = myMethods.newGetVentriclePixels2(tissuePixels);
6150
6151     boolean[][] isLeft = myMethods.getBooleanArrayFromPointList(
6152         currentSectionData.pointList,
6153         currentSectionData.imageWidth,
6154         currentSectionData.imageHeight);
6155
6156     int numberOfTissuePixels=0;
6157     int numberOfTissuePixelsLeft=0;
6158     int numberOfTissuePixelsRight=0;
6159
6160     int numberOfVentriclePixels=0;
6161     int numberOfVentriclePixelsLeft=0;
6162     int numberOfVentriclePixelsRight=0;
6163
6164     for (int i=0; i<originalImage.getWidth(); i++)
6165         for (int j=0; j<originalImage.getHeight(); j++)
6166         {
6167             if (tissuePixels[i][j]) numberOfTissuePixels++;
6168             if (tissuePixels[i][j] && isLeft[i][j]) numberOfTissuePixelsLeft++;
6169             if (tissuePixels[i][j] && !isLeft[i][j]) numberOfTissuePixelsRight++;
6170
6171             if (ventriclePixels[i][j]) numberOfVentriclePixels++;
6172             if (ventriclePixels[i][j] && isLeft[i][j]) numberOfVentriclePixelsLeft++;
6173             if (ventriclePixels[i][j] && !isLeft[i][j]) numberOfVentriclePixelsRight++;
6174         }
6175
6176     myProjectData.numberOfTissuePixels[subject][section] = numberOfTissuePixels;
6177     myProjectData.numberOfTissuePixelsLeft[subject][section] = numberOfTissuePixelsLeft;
6178     myProjectData.numberOfTissuePixelsRight[subject][section] = numberOfTissuePixelsRight;
6179
6180     myProjectData.numberOfVentriclePixels[subject][section] = numberOfVentriclePixels;
6181     myProjectData.numberOfVentriclePixelsLeft[subject][section] = numberOfVentriclePixelsLeft;
6182     myProjectData.numberOfVentriclePixelsRight[subject][section] = numberOfVentriclePixelsRight;
6183
6184     int[] numberOfUDApixels = new int[myProjectData.numberOfUDA];
6185
6186     for (int UDACounter=0; UDACounter<myProjectData.numberOfUDA; UDACounter++)
6187     {
6188         numberOfUDApixels[UDACounter] = myMethods.pixelsInUDA(subject, section, UDACounter,
6189             tissuePixels);
6190         myProjectData.numberOfUdaPixels[subject][section][UDACounter] =
6191             numberOfUDApixels[UDACounter];
6192     }
6193 }
6194
6195 //Counts the pixels in a user defined area (UDA)
6196 public int pixelsInUDA(int subject, int section, int UDANumber, boolean[][] tissue)
6197 {
6198     int numberOfPixels=0;
6199

```

```

6200     if (currentSectionData.haveUdaOrigin[UDANumber])
6201     {
6202
6203         boolean[][] linePixels = myMethods.getBooleanArrayFromPointList(
6204             currentSectionData.UDApoinList[UDANumber],
6205             currentSectionData.imageWidth,
6206             currentSectionData.imageHeight);
6207
6208
6209         Point startPosition = currentSectionData.udaOrigin[UDANumber];
6210
6211         boolean[][] UDAPixels = myMethods.newFillInside(linePixels, startPosition);
6212
6213         boolean includeTissue = myProjectData.includeTissue[UDANumber];
6214         boolean includeBackground = myProjectData.includeBackground[UDANumber];
6215
6216         for (int i=0; i<tissue.length;i++)
6217             for (int j=0; j<tissue[0].length;j++)
6218             {
6219                 if (tissue[i][j] && !includeTissue) UDAPixels[i][j]=false;
6220                 if (!tissue[i][j] && !includeBackground) UDAPixels[i][j]=false;
6221             }
6222         numberOfPixels=myMethods.countPixels(UDAPixels);
6223     }
6224     return numberOfPixels;
6225 }
6226
6227 //Updates the data used in the section results table
6228 public void updateSectionResultsObject(int subject, int section)
6229 {
6230     double tempDouble;
6231     int counter;
6232
6233     if (myProjectData.sectionExists[subject][section])
6234     {
6235         counter = myProjectData.positionInProject[subject][section];
6236
6237         myProjectData.sectionResultsObject[counter][0] = myProjectData.subjectNameList[subject];
6238         myProjectData.roundedSectionResultsObject[counter][0] =
6239 myProjectData.subjectNameList[subject];
6240         myProjectData.sectionResultsObject[counter][1] =
6241 myProjectData.sectionName[subject][section];
6242         myProjectData.roundedSectionResultsObject[counter][1] =
6243 myProjectData.sectionName[subject][section];
6244         myProjectData.sectionResultsObject[counter][2] =
6245 myProjectData.bregmaLevels[subject][section];
6246         myProjectData.roundedSectionResultsObject[counter][2] =
6247 myProjectData.bregmaLevels[subject][section];
6248         myProjectData.sectionResultsObject[counter][3] =
6249 myProjectData.numberOfTypePixels[subject][section];
6250         myProjectData.roundedSectionResultsObject[counter][3] =
6251 myProjectData.numberOfTypePixels[subject][section];
6252         tempDouble=( double)
6253 myProjectData.numberOfTypePixels[subject][section]/myProjectData.conversionFactor;
6254         myProjectData.sectionResultsObject[counter][4] = tempDouble;
6255         tempDouble= (double) (Math.round(tempDouble*100))/100;

```

```

6252         myProjectData.roundedSectionResultsObject[counter][4] = tempDouble;
6253
6254         myProjectData.sectionResultsObject[counter][5] =
myProjectData.numberOfTissuePixelsLeft[subject][section];
6255         myProjectData.roundedSectionResultsObject[counter][5] =
myProjectData.numberOfTissuePixelsLeft[subject][section];
6256
6257         tempDouble = (double)
myProjectData.numberOfTissuePixelsLeft[subject][section]/myProjectData.conversionFactor;
6258         myProjectData.sectionResultsObject[counter][6] = tempDouble;
6259         tempDouble= (double) (Math.round(tempDouble*100))/100;
6260         myProjectData.roundedSectionResultsObject[counter][6] = tempDouble;
6261
6262         myProjectData.sectionResultsObject[counter][7] =
myProjectData.numberOfTissuePixelsRight[subject][section];
6263         myProjectData.roundedSectionResultsObject[counter][7] =
myProjectData.numberOfTissuePixelsRight[subject][section];
6264
6265         tempDouble = (double)
myProjectData.numberOfTissuePixelsRight[subject][section]/myProjectData.conversionFactor;
6266         myProjectData.sectionResultsObject[counter][8] = tempDouble;
6267         tempDouble= (double) (Math.round(tempDouble*100))/100;
6268         myProjectData.roundedSectionResultsObject[counter][8] = tempDouble;
6269
6270         myProjectData.sectionResultsObject[counter][9] =
myProjectData.numberOfVentriclePixels[subject][section];
6271         myProjectData.roundedSectionResultsObject[counter][9] =
myProjectData.numberOfVentriclePixels[subject][section];
6272
6273         tempDouble = (double)
myProjectData.numberOfVentriclePixels[subject][section]/myProjectData.conversionFactor;
6274         myProjectData.sectionResultsObject[counter][10] = tempDouble;
6275         tempDouble= (double) (Math.round(tempDouble*100))/100;
6276         myProjectData.roundedSectionResultsObject[counter][10] = tempDouble;
6277
6278         myProjectData.sectionResultsObject[counter][11] =
myProjectData.numberOfVentriclePixelsLeft[subject][section];
6279         myProjectData.roundedSectionResultsObject[counter][11] =
myProjectData.numberOfVentriclePixelsLeft[subject][section];
6280
6281         tempDouble = (double)
myProjectData.numberOfVentriclePixelsLeft[subject][section]/myProjectData.conversionFactor;
6282         myProjectData.sectionResultsObject[counter][12] = tempDouble;
6283         tempDouble= (double) (Math.round(tempDouble*100))/100;
6284         myProjectData.roundedSectionResultsObject[counter][12] = tempDouble;
6285
6286         myProjectData.sectionResultsObject[counter][13] =
myProjectData.numberOfVentriclePixelsRight[subject][section];
6287         myProjectData.roundedSectionResultsObject[counter][13] =
myProjectData.numberOfVentriclePixelsRight[subject][section];
6288
6289         tempDouble = (double)
myProjectData.numberOfVentriclePixelsRight[subject][section]/myProjectData.conversionFactor;
6290         myProjectData.sectionResultsObject[counter][14] = tempDouble;
6291         tempDouble= (double) (Math.round(tempDouble*100))/100;
6292         myProjectData.roundedSectionResultsObject[counter][14] = tempDouble;
6293
6294         for (int k=0; k<myProjectData.numberOfUDA;k++)
6295         {

```

```

6296         myProjectData.sectionResultsObject[counter][15+2*k] =
myProjectData.numberOfUdaPixels[subject][section][k];
6297         myProjectData.roundedSectionResultsObject[counter][15+2*k] =
myProjectData.numberOfUdaPixels[subject][section][k];
6298         tempDouble = (double)
myProjectData.numberOfUdaPixels[subject][section][k]/myProjectData.conversionFactor;
6299         myProjectData.sectionResultsObject[counter][16+2*k] = tempDouble;
6300         tempDouble= (double) (Math.round(tempDouble*100))/100;
6301         myProjectData.roundedSectionResultsObject[counter][16+2*k] = tempDouble;
6302     }
6303 }
6304 }
6305
6306 //Calculates the volumes for a subject. It assumes that the areas in the sections are up to date.
6307 public void analyzeSubject(int subject)
6308 {
6309     double area1;
6310     double area2;
6311     double distance;
6312     double sliceVolume;
6313
6314     double tissueVolume=0;
6315     double tissueVolumeLeft=0;
6316     double tissueVolumeRight=0;
6317
6318     double ventricleVolume=0 ;
6319     double ventricleVolumeLeft=0;
6320     double ventricleVolumeRight=0;
6321
6322     double[] udaVolume = new double[myProjectData.numberOfUDA];
6323
6324     for (int udaCounter=0; udaCounter<myProjectData.numberOfUDA; udaCounter++)
6325         udaVolume[udaCounter]=0;
6326
6327     for (int j=0;j<myProjectData.numberOfLevels-1;j++)
6328     {
6329         if (myProjectData.sectionExists[subject][j] && myProjectData.sectionExists[subject][j+1])
6330         {
6331             distance= myProjectData.bregmaLevels[subject][j+1]-
myProjectData.bregmaLevels[subject][j];
6332             distance = Math.abs(distance);
6333
6334             area1 = (double)
myProjectData.numberOfTissuePixels[subject][j]/myProjectData.conversionFactor;
6335             area2 = (double)
myProjectData.numberOfTissuePixels[subject][j+1]/myProjectData.conversionFactor;
6336
6337             sliceVolume=distance*(area1+area2)/2;
6338             tissueVolume=tissueVolume+sliceVolume;
6339
6340             area1 = (double)
myProjectData.numberOfTissuePixelsLeft[subject][j]/myProjectData.conversionFactor;
6341             area2 = (double)
myProjectData.numberOfTissuePixelsLeft[subject][j+1]/myProjectData.conversionFactor;
6342
6343             sliceVolume=distance*(area1+area2)/2;
6344             tissueVolumeLeft=tissueVolumeLeft+sliceVolume;
6345
6346             area1 = (double)
myProjectData.numberOfTissuePixelsRight[subject][j]/myProjectData.conversionFactor;

```

```

6347         area2 = (double)
myProjectData.numberOfTissuePixelsRight[subject][j+1]/myProjectData.conversionFactor;
6348
6349         sliceVolume=distance*(area1+area2)/2;
6350         tissueVolumeRight=tissueVolumeRight+sliceVolume;
6351
6352         area1 = (double)
myProjectData.numberOfVentriclePixels[subject][j]/myProjectData.conversionFactor;
6353         area2 = (double)
myProjectData.numberOfVentriclePixels[subject][j+1]/myProjectData.conversionFactor;
6354
6355         sliceVolume=distance*(area1+area2)/2;
6356         ventricleVolume=ventricleVolume+sliceVolume;
6357
6358         area1 = (double)
myProjectData.numberOfVentriclePixelsLeft[subject][j]/myProjectData.conversionFactor;
6359         area2 = (double)
myProjectData.numberOfVentriclePixelsLeft[subject][j+1]/myProjectData.conversionFactor;
6360
6361         sliceVolume=distance*(area1+area2)/2;
6362         ventricleVolumeLeft=ventricleVolumeLeft+sliceVolume;
6363
6364         area1 = (double)
myProjectData.numberOfVentriclePixelsRight[subject][j]/myProjectData.conversionFactor;
6365         area2 = (double)
myProjectData.numberOfVentriclePixelsRight[subject][j+1]/myProjectData.conversionFactor;
6366
6367         sliceVolume=distance*(area1+area2)/2;
6368         ventricleVolumeRight=ventricleVolumeRight+sliceVolume;
6369
6370         for (int udaCounter=0; udaCounter<myProjectData.numberOfUDA; udaCounter++)
6371         {
6372             area1 = (double)
myProjectData.numberOfUdaPixels[subject][j][udaCounter]/myProjectData.conversionFactor;
6373             area2 = (double)
myProjectData.numberOfUdaPixels[subject][j+1][udaCounter]/myProjectData.conversionFactor;
6374             sliceVolume=distance*(area1+area2)/2;
6375             udaVolume[udaCounter]=udaVolume[udaCounter]+sliceVolume;
6376         }
6377     }
6378 }
6379
6380 myProjectData.tissueVolume[subject] = tissueVolume;
6381 myProjectData.tissueVolumeLeft[subject] = tissueVolumeLeft;
6382 myProjectData.tissueVolumeRight[subject] = tissueVolumeRight;
6383
6384 myProjectData.ventricleVolume[subject] = ventricleVolume;
6385 myProjectData.ventricleVolumeLeft[subject] = ventricleVolumeLeft;
6386 myProjectData.ventricleVolumeRight[subject] = ventricleVolumeRight;
6387
6388 for (int udaCounter=0; udaCounter<myProjectData.numberOfUDA; udaCounter++)
6389     myProjectData.udaVolume[subject][udaCounter] = udaVolume[udaCounter];
6390 }
6391
6392 //Determines if a pixel is tissue or background based on the color and treshold values.
6393 public boolean pixellIsTissue (int[] color)
6394 {
6395     boolean isTissue=true;
6396
6397     if (color[0]+color[1]+color[2]==0)

```

```

6398     {
6399         isTissue=false;
6400     }
6401     else if (100*(color[0]+color[2])/(color[0]+color[1]+color[2])<currentSectionData.colorTreshold)
6402         isTissue=false;
6403
6404     if (color[0]+color[1]+color[2]<currentSectionData.intensityTreshold)
6405         isTissue=false;
6406
6407     return isTissue;
6408 }
6409
6410 //Returns an array of boolean with the same size as the input image.
6411 //The value is true if the pixel is tissue and false if it is background
6412 public boolean[][] getTissuePixels(BufferedImage inputImage)
6413 {
6414     boolean[][] tissuePixels = new boolean[inputImage.getWidth()][inputImage.getHeight()];
6415     Raster myRaster=inputImage.getRaster();
6416     int[] color = new int[3];
6417
6418     for (int i=0;i<inputImage.getWidth();i++)
6419         for (int j=0;j<inputImage.getHeight();j++)
6420         {
6421             color=myRaster.getPixel(i,j,color);
6422
6423             tissuePixels[i][j]=this.pixelIsTissue(color);
6424
6425             if (currentSectionData.markedAsBackground[i][j]) tissuePixels[i][j]=false;
6426             if (currentSectionData.markedAsTissue[i][j]) tissuePixels[i][j]=true;
6427
6428             if (i<3 || i>(inputImage.getWidth()-4) || j<3 || j>(inputImage.getHeight()-4))
6429             {
6430                 //Pixels close to the border of the image are set as false to avoid
6431                 //trying to access pixels outside the image when working with objects
6432                 //close to the border
6433                 tissuePixels[i][j]=false;
6434             }
6435         }
6436
6437     //If theres no tissue pixels one pixel is set as tissue to avoid errors
6438     if (myMethods.countPixels(tissuePixels) == 0) tissuePixels[3][3]=true;
6439
6440     return tissuePixels;
6441 }
6442
6443 public boolean[][] newFillInside(boolean[][] outlinePixels, Point startPoint)
6444 {
6445     int width = outlinePixels.length;
6446     int height = outlinePixels[0].length;
6447
6448     boolean[][] insidePixels = new boolean [width][height];
6449
6450     for (int i=0;i<width;i++)
6451         for (int j=0;j<height;j++)
6452         {
6453             insidePixels[i][j]=false;
6454         }
6455
6456     insidePixels[startPoint.x][startPoint.y]=true;
6457

```

```

6458     boolean foundMore=true;
6459
6460     while (foundMore)
6461     {
6462         foundMore=false;
6463
6464         for (int x=0;x<width;x++)
6465         for (int y=0;y<height;y++)
6466             if (insidePixels[x][y])
6467             {
6468                 try{
6469                     if(!insidePixels[x][y-1] && !outlinePixels[x][y-1])
6470                     {
6471                         insidePixels[x][y-1]=true;
6472                         foundMore=true;
6473                     }
6474                 }catch(Exception error) {}
6475
6476                 try{
6477                     if(!insidePixels[x][y+1] && !outlinePixels[x][y+1])
6478                     {
6479                         insidePixels[x][y+1]=true;
6480                         foundMore=true;
6481                     }
6482                 }catch(Exception error) {}
6483
6484                 try{
6485                     if(!insidePixels[x-1][y] && !outlinePixels[x-1][y])
6486                     {
6487                         insidePixels[x-1][y]=true;
6488                         foundMore=true;
6489                     }
6490                 }catch(Exception error) {}
6491
6492                 try{
6493                     if(!insidePixels[x+1][y] && !outlinePixels[x+1][y])
6494                     {
6495                         insidePixels[x+1][y]=true;
6496                         foundMore=true;
6497                     }
6498                 }catch(Exception error) {}
6499             }
6500         if (!foundMore) return insidePixels;
6501
6502         foundMore=false;
6503
6504         for (int x=width-1;x>=0;x--)
6505         for (int y=height-1;y>=0;y--)
6506             if (insidePixels[x][y])
6507             {
6508                 try{
6509                     if(!insidePixels[x][y-1] && !outlinePixels[x][y-1])
6510                     {
6511                         insidePixels[x][y-1]=true;
6512                         foundMore=true;
6513                     }
6514                 }catch(Exception error) {}
6515
6516                 try{
6517

```



```

6518         if(!insidePixels[x][y+1] && !outlinePixels[x][y+1])
6519         {
6520             insidePixels[x][y+1]=true;
6521             foundMore=true;
6522         }
6523     }catch(Exception error) {}
6524
6525     try{
6526         if(!insidePixels[x-1][y] && !outlinePixels[x-1][y])
6527         {
6528             insidePixels[x-1][y]=true;
6529             foundMore=true;
6530         }
6531     }catch(Exception error) {}
6532
6533     try{
6534         if(!insidePixels[x+1][y] && !outlinePixels[x+1][y])
6535         {
6536             insidePixels[x+1][y]=true;
6537             foundMore=true;
6538         }
6539     }catch(Exception error) {}
6540 }
6541 }
6542 return insidePixels;
6543 }
6544
6545 //Counts the number of true in an array of boolean.
6546 public int countPixels(boolean[][] inputArray)
6547 {
6548     int numberOfPixels=0;
6549
6550     for (int i=0;i<inputArray.length;i++)
6551         for (int j=0;j<inputArray[0].length;j++)
6552         {
6553             if (inputArray[i][j]) numberOfPixels++;
6554         }
6555     return numberOfPixels;
6556 }
6557
6558 //If the boolean for the pixel is true it is changed to the provided color.
6559 public void changePixelsInImage(BufferedImage image, boolean[][] outlinePixels, int[] color)
6560 {
6561     WritableRaster raster = image.getRaster();
6562
6563     for (int i=0;i<image.getWidth();i++)
6564         for (int j=0;j<image.getHeight();j++)
6565         {
6566             if (outlinePixels[i][j]) raster.setPixel(i, j, color);
6567         }
6568 }
6569
6570 //Scans all horisontal lines from left to right. All pixels in the line
6571 //are considered to be left until a pixel in the lineArray is encountered.
6572 public boolean[][] isLeftFromArray(boolean[][] lineArray)
6573 {
6574     boolean[][] isLeft = new boolean[lineArray.length][lineArray[0].length];
6575     setBooleanArrayValue(isLeft,false);
6576
6577     boolean foundDivider;

```



```

6578
6579     for (int j=0;j<lineArray[0].length;j++)
6580     {
6581         foundDivider=false;
6582         for (int i=0;i<lineArray.length;i++)
6583         {
6584             if (lineArray[i][j]) foundDivider=true;
6585             {
6586                 if (!foundDivider) isLeft[i][j]=true;
6587                 else isLeft[i][j]=false;
6588             }
6589         }
6590     }
6591     return isLeft;
6592 }
6593
6594 public String getOriginalImageFileName(int subjectNumber, int sectionNumber)
6595 {
6596     String projectLocation = myProjectData.projectLocation;
6597
6598     Integer subjectInteger= new Integer(subjectNumber);
6599     String subjectNumberString = subjectInteger.toString();
6600
6601     Integer sectionInteger= new Integer(sectionNumber);
6602     String sectionNumberString = sectionInteger.toString();
6603
6604     String imageFileName = subjectNumberString+"."
6605         +sectionNumberString+".OriginalImage";
6606
6607     imageFileName = (projectLocation+"/"+imageFileName);
6608
6609     return imageFileName;
6610 }
6611
6612 public String getSectionDataFileName(int subjectNumber, int sectionNumber)
6613 {
6614     String projectLocation = myProjectData.projectLocation;
6615
6616     Integer subjectInteger= new Integer(subjectNumber);
6617     String subjectNumberString = subjectInteger.toString();
6618
6619     Integer sectionInteger= new Integer(sectionNumber);
6620     String sectionNumberString = sectionInteger.toString();
6621
6622     String fileName = subjectNumberString+"."
6623         +sectionNumberString+".SectionData";
6624
6625     fileName = (projectLocation+"/"+fileName);
6626
6627     return fileName;
6628 }
6629
6630 public void saveSectionData(int subjectNumber, int sectionNumber, SectionData dataToSave)
6631 {
6632     String saveFileName= myMethods.getSectionDataFileName(subjectNumber, sectionNumber);
6633
6634     File saveObjectFile = new File(saveFileName);
6635
6636     FileOutputStream fos;
6637     ObjectOutputStream out;

```

```

6638     try
6639     {
6640         fos = new FileOutputStream(saveObjectFile);
6641
6642         BufferedOutputStream buffOut = new BufferedOutputStream(fos);
6643
6644         GZIPOutputStream gz = new GZIPOutputStream(buffOut);
6645
6646         out = new ObjectOutputStream(gz);
6647
6648         out.writeObject(dataToSave);
6649
6650         out.flush();
6651         out.close();
6652     }
6653     catch (java.io.FileNotFoundException er)
6654     {
6655         System.out.println(er.toString());
6656     }
6657     catch (java.io.IOException ex)
6658     {
6659         System.out.println(ex.toString());
6660     }
6661 }
6662
6663 public SectionData readSectionData(int subjectNumber, int sectionNumber)
6664 {
6665     SectionData newSectionData = new SectionData(myProjectData.imageWidth,
6666     myProjectData.imageHeight);
6667
6668     String fileName= myMethods.getSectionDataFileName(subjectNumber, sectionNumber);
6669
6670     File file = new File(fileName);
6671
6672     FileInputStream fis;
6673     ObjectInputStream in;
6674     try
6675     {
6676         fis = new FileInputStream(file);
6677
6678         BufferedInputStream buffIn = new BufferedInputStream(fis);
6679
6680         GZIPInputStream zipIn = new GZIPInputStream(buffIn);
6681
6682         in = new ObjectInputStream(zipIn);
6683         newSectionData = (SectionData)in.readObject();
6684         in.close();
6685     }
6686     catch (IOException ex)
6687     {
6688         ex.printStackTrace();
6689     }
6690     catch (ClassNotFoundException ex)
6691     {}
6692
6693     return newSectionData;
6694 }
6695
6696 public void saveProjectData()

```

```

6697 {
6698     //Triggers a WhenLeftInTree for the current panel to make sure
6699     //that any last changes are also being saved
6700     TreePath selectionPath = treePanel.tree.getSelectionPath();
6701     treePanel.tree.setSelectionRow(0);
6702     treePanel.tree.setSelectionPath(selectionPath);
6703
6704     String saveFileName = myProjectData.projectLocation;
6705     saveFileName=saveFileName+"/ProjectData.ser";
6706     File saveObjectFile = new File(saveFileName);
6707     FileOutputStream fos;
6708     ObjectOutputStream out;
6709     try
6710     {
6711         fos = new FileOutputStream(saveObjectFile);
6712         out = new ObjectOutputStream(fos);
6713         out.writeObject(myProjectData);
6714         out.close();
6715     }
6716     catch (java.io.FileNotFoundException er)
6717     {
6718     }
6719 }
6720 catch(java.io.IOException ex)
6721 {
6722 }
6723 }
6724 }
6725
6726 //Adds pixels to a line based on the setting for lineWidth to make
6727 //the line easier to see.
6728 public boolean[][] widenLine (boolean[][] linePixels, int lineWidth)
6729 {
6730     boolean[][] newLine = new boolean[linePixels.length][linePixels[0].length];
6731
6732     for (int i=0 ; i<linePixels.length ; i++)
6733         for (int j =0; j<linePixels[0].length;j++)
6734         {
6735             if (linePixels[i][j])
6736             {
6737                 for (int x=-lineWidth;x<lineWidth+1;x++)
6738                     for (int y = -lineWidth; y<lineWidth+1;y++)
6739                     {
6740                         try
6741                         {
6742                             newLine[i+x][j+y]=true;
6743                         }catch (Exception error){ }
6744                     }
6745             }
6746         }
6747     return newLine;
6748 }
6749
6750 public String getHelpTextChoseProjectFolder()
6751 {
6752     String helpText =
6753         "LOCATE SOURCE IMAGES" + "\n" + "\n" +
6754         "When you click the "Select project folder" button you will be" + "\n" +
6755         "asked to locate the folder containing your source images. " + "\n" +
6756         "The image files from each subject have to be located in a " + "\n" +

```

```

6757         "folder (subject folder). All the subject folders have to be " + '\n'+
6758         "placed in the same folder (project folder). When you locate " + '\n'+
6759         "the project folder the program will scan for usable image " + '\n'+
6760         "files and present the number of subject and sections. Any files " + '\n'+
6761         "the program can't open as images are ignored. All images have " + '\n'+
6762         "to be of the same size and the program makes sure they are. " + '\n'+
6763         "The folder names will be used as subject names and the image " + '\n'+
6764         "file names as section names in the project."+ '\n'+ '\n'+
6765         "IMPORTANT: The images for subject have to be named in strict " + '\n'+
6766         "alphabetical order, otherwise they will be placed in the wrong " + '\n'+
6767         "position. Especially note that Section12 will appear before " + '\n'+
6768         "Section2, to avoid this problem change the name to Section02.";
6769     return helpText;
6770 }
6771
6772 public String getHelpTextChoseSaveFolder()
6773 {
6774     String helpText =
6775         "CHOOSE WHERE TO SAVE PROJECT"+ '\n'+ '\n'+
6776         "You have to locate a folder where the project data should be " + '\n'+
6777         "saved (save folder). Three files will be created for each " + '\n'+
6778         "section, one that holds the original image, one that holds any " + '\n'+
6779         "modifications done to the image and one file which stores data " + '\n'+
6780         "about the section. The program checks that the folder is empty " + '\n'+
6781         "before proceeding to make sure that no files are overwritten " + '\n'+
6782         "and to avoid mixing up project files with other files. The save " + '\n'+
6783         "folder will require disk space equal to size of the original " + '\n'+
6784         "images and approximately 10 kB extra for each section.";
6785     return helpText;
6786 }
6787
6788 public String getHelpTextChoseConversionImage()
6789 {
6790     String helpText =
6791         "CHOOSE CONVERSION IMAGE"+ '\n'+ '\n'+
6792         "The conversion image is used to convert pixels to mm. This image " + '\n'+
6793         "has to be acquired using the same microscope settings as the " + '\n'+
6794         "section images and contain something of a known size, preferably " + '\n'+
6795         "a scale bar. If the microscope can't include scale bars an image " + '\n'+
6796         "of a grid or some other object of known size will work as well. " + '\n'+
6797         "The program will check to make sure that it is a readable image " + '\n'+
6798         "file of the same size as the section images."+ '\n'+ '\n'+
6799         "ENTER CONVERSION VALUES"+ '\n'+
6800         "Once a project has been created the conversion panel is used " + '\n'+
6801         "to determine the number of pixels per square millimeter. The " + '\n'+
6802         "number of pixels is entered by drawing a line in the image, " + '\n'+
6803         "left click for the start point and right click for the end point. " + '\n'+
6804         "The program will calculate the distance between the points in " + '\n'+
6805         "pixels. Enter this distance in millimeters in the text box on " + '\n'+
6806         "the right side panel. Hit enter to update the conversion factor." + '\n'+
6807         "The program will now recalculate all areas and volumes using " + '\n'+
6808         "these settings and the number of pixels per square millimeter " + '\n'+
6809         "is reported.";
6810     return helpText;
6811 }
6812
6813 public String getHelpTextBregmaLevels()
6814 {
6815     String helpText =
6816         "ENTER LEVELS"+ '\n'+ '\n'+

```

```

6817         "A level has to be assigned to each section to allow the calculation " + '\n'+
6818         "of volumes. The program will check for the highest number of " + '\n'+
6819         "sections in a subject and you will be prompted to enter that " + '\n'+
6820         "many numbers. If your sections are equally spaced you can enter " + '\n'+
6821         "the first level and the distance between them and use the " + '\n'+
6822         "Calculate levels-button. In case your sections aren't equally " + '\n'+
6823         "spaced a numbers can be entered manually for each level. If " + '\n'+
6824         "neither of this works out the level can be assigned individually " + '\n'+
6825         "for each section once the project is created.";
6826     return helpText;
6827 }
6828
6829 public String getHelpTextUDA()
6830 {
6831     String helpText =
6832         "USER DEFINED AREAS" + '\n'+ '\n'+
6833         "The program will automatically detect tissue and ventricles but if you wish to " + '\n'+
6834         "measure any other brain region this can be done using "User defined areas". " + '\n'+
6835         "For each area you will be asked for a name and to specify whether the area " + '\n'+
6836         "should include background, tissue or both.";
6837
6838     return helpText;
6839 }
6840
6841 public String getHelpTextTissueDetection()
6842 {
6843     String helpText =
6844         "DETECTING TISSUE" + '\n'+ '\n'+
6845         "This panel allows you to adjust the threshold value used to " + '\n'+
6846         "separate tissue from background. Use the "Apply to all" button " + '\n'+
6847         "to use the setting on all sections. The values can also be " + '\n'+
6848         "adjusted individually for each section in the adjust image panel. " + '\n'+
6849         "Each pixel in the image has one value for red, one for green " + '\n'+
6850         "and one for blue that can range from 0 to 255. The stained tissue " + '\n'+
6851         "will have higher values for red and blue but low for green. " + '\n'+
6852         "The program calculates two values, the color value and the " + '\n'+
6853         "intensity value:" + '\n'+ '\n'+
6854         "Color = 100 * (Red + Blue) / (Red + Blue + Green)" + '\n'+ '\n'+
6855         "Intensity = Red + Blue + Green" + '\n'+ '\n'+
6856         "If both values are above the set thresholds the pixel will be " + '\n'+
6857         "considered as tissue. Move the mouse over the image to see values " + '\n'+
6858         "for red, green, blue, the color, the intensity and whether a " + '\n'+
6859         "pixel is considered to be tissue or background. The intensity " + '\n'+
6860         "value is used to filter out dark pixels which have high color " + '\n'+
6861         "value, for example (R=1;G=0;B=0) will appear black but have a " + '\n'+
6862         "color value of 100.";
6863     return helpText;
6864 }
6865
6866 public String getHelpTextAdjustImage()
6867 {
6868     String helpText =
6869         "ADJUST IMAGE" + '\n'+ '\n'+
6870         "Normally at least some sections in a project will contain folds " + '\n'+
6871         "or tears or other errors from the sectioning and staining. This " + '\n'+
6872         "can be corrected for by adding or removing tissue. Use the buttons " + '\n'+
6873         "on the right side panel to choose whether to add or remove tissue " + '\n'+
6874         "and the size used when drawing. It is also possible to change the " + '\n'+
6875         "threshold for tissue detection. This is done in the same way as in " + '\n'+
6876         "the Tissue detection panel but any changes made here will only " + '\n'+

```

```

6877         "apply to this section.";
6878     return helpText;
6879 }
6880
6881 public String getAboutText()
6882 {
6883     String aboutText =
6884         "SectionToVolume version 1.1" + '\n' + '\n' +
6885         "Originally created by Anders Hånell in 2011." + '\n' + '\n' +
6886
6887         "SectionToVolume is free to use but please cite the article by " + '\n' +
6888         "Hånell et al if you use SectionToVolume in your research." + '\n' +
6889         "This is a way to get official recognition for the hard work" + '\n' +
6890         "required to create this program." + '\n' + '\n' +
6891
6892         "Facilitated assessment of tissue loss following traumatic brain injury" + '\n' +
6893         "Hånell A, Hedin J, Clausen F, Marklund N" + '\n' +
6894         "Front Neurol. 2012;3:29. Epub 2012 Mar 14" + '\n' +
6895         "PMID: 22435063" + '\n' + '\n' +
6896
6897         "The functions in the program has been thoroughly tested " + '\n' +
6898         "but no guarantees are given for correct results." + '\n' + '\n' +
6899         "Send questions and suggestions to SectionToVolume@gmail.com" + '\n' + '\n' + '\n' +
6900
6901         "This software is not subject to copyright protection and " + '\n' +
6902         "is in the public domain. SectionToVolume is an experimental " + '\n' +
6903         "system and the author assumes no responsibility whatsoever " + '\n' +
6904         "for its use by other parties, and makes no guarantees, " + '\n' +
6905         "expressed or implied, about its quality, reliability, " + '\n' +
6906         "or any other characteristic.";
6907
6908     return aboutText;
6909 }
6910
6911 public String getErrorTextApplyToAllInterrupted()
6912 {
6913     String errorText =
6914         "AN ERROR OCCURED" + '\n' + '\n' +
6915         "Apply to all function aborted by user" + '\n' +
6916         "" + '\n' +
6917         "WARNING" + '\n' +
6918         "" + '\n' +
6919         "This can cause incorrect results." + '\n' +
6920         "Start the apply to all function again and let it finish" + '\n' +
6921         "to assure correct calculation of the results." + '\n' + '\n' + '\n';
6922
6923     return errorText;
6924 }
6925
6926 public String getErrorTextCreateProjectInterrupted()
6927 {
6928     String errorText =
6929         "AN ERROR OCCURED" + '\n' + '\n' +
6930         "Create project function aborted by user" + '\n' +
6931         "" + '\n' +
6932         "WARNING" + '\n' +
6933         "" + '\n' +
6934         "The project will most likely not function." + '\n' +
6935         "Delete any created files in the save folder and" +
6936         "start the create project function again and let it finish" + '\n' +

```

```

6937         "to create the project."+ '\n'+ '\n'+ '\n';
6938
6939     return errorText;
6940 }
6941
6942 public int countNumberOfFiles(File location)
6943 {
6944     int numberOfFiles=0;
6945
6946     File[] subjectFileList;
6947     File[][] sectionFileList = new File[1000][1000];
6948
6949     subjectFileList = location.listFiles();
6950
6951     numberOfFiles+=subjectFileList.length;
6952
6953     for (int subjectCounter = 0; subjectCounter < subjectFileList.length; subjectCounter++)
6954     {
6955         try
6956         {
6957             sectionFileList[subjectCounter] = subjectFileList[subjectCounter].listFiles();
6958             numberOfFiles+=sectionFileList[subjectCounter].length;
6959         } catch (Exception error){}
6960     }
6961
6962
6963     return numberOfFiles;
6964 }
6965
6966 //Tries to read the file and count the number of tissue pixels. If it fails the file is not used.
6967 public boolean fileIsValidImage(File testFile)
6968 {
6969     if (testFile.isDirectory()) return false;
6970
6971     //Windows crate a file named Thumbs.db which cause the JAI fileload
6972     //to generate error messages.
6973     String fileName = testFile.toString();
6974     int lastPeriodPosition = fileName.lastIndexOf('.');
6975     String extension = fileName.substring(lastPeriodPosition);
6976     if (extension.equals(".db")) return false;
6977
6978     boolean fileIsImage;
6979     BufferedImage tempBufferedImage;
6980
6981     try
6982     {
6983         FileSeekableStream seekableStream = new FileSeekableStream(testFile);
6984         RenderedImage src = (RenderedImage)JAI.create("stream", seekableStream);
6985         tempBufferedImage = PlanarImage.wrapRenderedImage(src).getAsBufferedImage();
6986         WritableRaster raster = tempBufferedImage.getRaster();
6987
6988         int imageWidth=tempBufferedImage.getWidth();
6989         int imageHeight=tempBufferedImage.getHeight();
6990
6991         if (createProjectDialog1.detectedImageWidth==-1)
6992         {
6993             createProjectDialog1.detectedImageWidth=imageWidth;
6994         }
6995         else
6996         {

```

```

6997         if (createProjectDialog1.detectedImageWidth!=imageWidth)
6998             createProjectDialog1.allImagesIsOfSameSize=false;
6999     }
7000
7001     if (createProjectDialog1.detectedImageHeight===-1)
7002     {
7003         createProjectDialog1.detectedImageHeight=imageHeight;
7004     }
7005     else
7006     {
7007         if (createProjectDialog1.detectedImageHeight!=imageHeight)
7008             createProjectDialog1.allImagesIsOfSameSize=false;
7009     }
7010
7011     fileIsImage=true;
7012 }
7013 catch (Exception event)
7014 {
7015     fileIsImage=false;
7016 }
7017
7018 return fileIsImage;
7019 }
7020
7021 //The positions between the two points are set to true.
7022 public boolean[][] drawLineBoolean (boolean[][] isLine, Point previousPoint, Point currentPoint)
7023 {
7024     int distanceX;
7025     int distanceY;
7026
7027     int tempX;
7028     int tempY;
7029
7030     distanceX=Math.abs(previousPoint.x-currentPoint.x);
7031     distanceY=Math.abs(previousPoint.y-currentPoint.y);
7032
7033     if (distanceX>distanceY)
7034     {
7035         if (previousPoint.x<currentPoint.x)
7036         {
7037             if (previousPoint.y<=currentPoint.y)
7038             {
7039                 for (int i=1;i<=distanceX;i++)
7040                 {
7041                     tempX=previousPoint.x+i;
7042                     tempY=previousPoint.y+distanceY*i/distanceX;
7043                     isLine[tempX][tempY] = true;
7044                 }
7045             }
7046             if (previousPoint.y>currentPoint.y)
7047             {
7048                 for (int i=1;i<=distanceX;i++)
7049                 {
7050                     tempX=previousPoint.x+i;
7051                     tempY=previousPoint.y-distanceY*i/distanceX;
7052                     isLine[tempX][tempY] = true;
7053                 }
7054             }
7055         }
7056         if (previousPoint.x>currentPoint.x)

```



```

7057     {
7058         if (previousPoint.y<=currentPoint.y)
7059         {
7060             for (int i=1;i<=distanceX;i++)
7061             {
7062                 tempX=previousPoint.x-i;
7063                 tempY=previousPoint.y+distanceY*i/distanceX;
7064                 isLine[tempX][tempY] = true;
7065             }
7066         }
7067         if (previousPoint.y>currentPoint.y)
7068         {
7069             for (int i=1;i<=distanceX;i++)
7070             {
7071                 tempX=previousPoint.x-i;
7072                 tempY=previousPoint.y-distanceY*i/distanceX;
7073                 isLine[tempX][tempY] = true;
7074             }
7075         }
7076     }
7077 }
7078 else
7079 {
7080     if (previousPoint.y<currentPoint.y)
7081     {
7082         if (previousPoint.x<=currentPoint.x)
7083         {
7084             for (int i=1;i<=distanceY;i++)
7085             {
7086                 tempY=previousPoint.y+i;
7087                 tempX=previousPoint.x+distanceX*i/distanceY;
7088                 isLine[tempX][tempY] = true;
7089             }
7090         }
7091         if (previousPoint.x>currentPoint.x)
7092         {
7093             for (int i=1;i<=distanceY;i++)
7094             {
7095                 tempY=previousPoint.y+i;
7096                 tempX=previousPoint.x-distanceX*i/distanceY;
7097                 isLine[tempX][tempY] = true;
7098             }
7099         }
7100     }
7101     if (previousPoint.y>currentPoint.y)
7102     {
7103         if (previousPoint.x<=currentPoint.x)
7104         {
7105             for (int i=1;i<=distanceY;i++)
7106             {
7107                 tempY=previousPoint.y-i;
7108                 tempX=previousPoint.x+distanceX*i/distanceY;
7109                 isLine[tempX][tempY] = true;
7110             }
7111         }
7112         if (previousPoint.x>currentPoint.x)
7113         {
7114             for (int i=1;i<=distanceY;i++)
7115             {
7116                 tempY=previousPoint.y-i;

```

```

7117         tempX=previousPoint.x-distanceX*i/distanceY;
7118         isLine[tempX][tempY] = true;
7119     }
7120 }
7121 }
7122 }
7123 return isLine;
7124 }
7125
7126 public boolean[][] newGetVentriclePixels2(boolean[][] tissuePixels) {
7127     boolean[][] ventriclePixels = new boolean[tissuePixels.length][tissuePixels[0].length];
7128     boolean[][] continousBackgroundArea;
7129
7130     for (int i=0; i<tissuePixels.length; i++)
7131         for (int j=0; j<tissuePixels[0].length; j++) {
7132             if (tissuePixels[i][j]) ventriclePixels[i][j]=false;
7133             else ventriclePixels[i][j] =true;
7134         }
7135
7136     continousBackgroundArea=getContinousBackgroundArea2(tissuePixels, 0, 0);
7137
7138     for (int i=0; i<tissuePixels.length; i++)
7139         for (int j=0; j<tissuePixels[0].length; j++) {
7140             if (continousBackgroundArea[i][j]) {
7141                 ventriclePixels[i][j]=false;
7142             }
7143         }
7144
7145     return ventriclePixels;
7146 }
7147
7148 public boolean[][] getContinousBackgroundArea2(boolean[][] tissuePixels, int startX, int startY) {
7149     boolean[][] backgroundArea = new boolean[tissuePixels.length][tissuePixels[0].length];
7150     for (int i=0; i<tissuePixels.length; i++)
7151         for (int j=0; j<tissuePixels[0].length; j++) {
7152             backgroundArea[i][j] = false;
7153         }
7154
7155     backgroundArea[startX][startY] = true;
7156
7157     int startColumn = 0;
7158     boolean foundStartColumn = false;
7159     int endColumn = tissuePixels[0].length;
7160     int startRow = tissuePixels.length;
7161     int endRow = 0;
7162
7163     for (int x=0; x<tissuePixels.length; x++)
7164         for (int y=0; y<tissuePixels[0].length; y++) {
7165             backgroundArea[x][y]=true;
7166             if (y<tissuePixels[0].length-1)
7167                 if (tissuePixels[x][y+1]) {
7168                     if (!foundStartColumn) {
7169                         foundStartColumn = true;
7170                         startColumn = x;
7171                     }
7172                     endColumn=x+1;
7173
7174                     if (y-1<startRow) startRow = y-1;
7175
7176                     for (int y2=tissuePixels[0].length-1; y2!=1; y2--) {

```

```

7177         backgroundArea[x][y2]=true;
7178         if (tissuePixels[x][y2-1]) {
7179             if (y2+1>endRow) endRow = y2+1;
7180             break;
7181         }
7182     }
7183     break;
7184 }
7185 }
7186 }
7187
7188 boolean foundMore=true;
7189 while (foundMore)
7190 {
7191     foundMore=false;
7192
7193     for (int x=startColumn;x<endColumn;x++)
7194     for (int y=startRow;y<endRow;y++)
7195         if (backgroundArea[x][y])
7196         {
7197
7198             try{
7199                 if(!tissuePixels[x][y-1] && !backgroundArea[x][y-1])
7200                 {
7201                     backgroundArea[x][y-1]=true;
7202                     foundMore=true;
7203                 }
7204             }catch(Exception error) {}
7205
7206             try{
7207                 if(!tissuePixels[x][y+1] && !backgroundArea[x][y+1])
7208                 {
7209                     backgroundArea[x][y+1]=true;
7210                     foundMore=true;
7211                 }
7212             }catch(Exception error) {}
7213
7214             try{
7215                 if(!tissuePixels[x-1][y] && !backgroundArea[x-1][y])
7216                 {
7217                     backgroundArea[x-1][y]=true;
7218                     foundMore=true;
7219                 }
7220             }catch(Exception error) {}
7221
7222             try{
7223                 if(!tissuePixels[x+1][y] && !backgroundArea[x+1][y])
7224                 {
7225                     backgroundArea[x+1][y]=true;
7226                     foundMore=true;
7227                 }
7228             }catch(Exception error) {}
7229
7230         }
7231
7232     for (int x=endColumn;x!=startColumn;x--)
7233     for (int y=endRow;y!=startRow;y--)
7234         if (backgroundArea[x][y])
7235         {
7236             try{

```

```

7237         if(!tissuePixels[x][y-1] && !backgroundArea[x][y-1])
7238         {
7239             backgroundArea[x][y-1]=true;
7240             foundMore=true;
7241         }
7242     }catch(Exception error) {}
7243
7244     try{
7245         if(!tissuePixels[x][y+1] && !backgroundArea[x][y+1])
7246         {
7247             backgroundArea[x][y+1]=true;
7248             foundMore=true;
7249         }
7250     }catch(Exception error) {}
7251
7252     try{
7253         if(!tissuePixels[x-1][y] && !backgroundArea[x-1][y])
7254         {
7255             backgroundArea[x-1][y]=true;
7256             foundMore=true;
7257         }
7258     }catch(Exception error) {}
7259
7260     try{
7261         if(!tissuePixels[x+1][y] && !backgroundArea[x+1][y])
7262         {
7263             backgroundArea[x+1][y]=true;
7264             foundMore=true;
7265         }
7266     }catch(Exception error) {}
7267     }
7268 }
7269
7270 return backgroundArea;
7271 }
7272
7273 public boolean[][] getBooleanArrayFromPointList(ArrayList pointList, int width, int height) {
7274
7275     boolean[][] array = new boolean[width][height];
7276
7277     for (int i=0; i<width; i++)
7278         for (int j=0; j<height; j++) {
7279             array[i][j]=false;
7280         }
7281
7282     Object[] pointArray = pointList.toArray();
7283
7284     Point firstPoint;
7285     Point secondPoint;
7286     for (int i=0; i<pointArray.length-1; i++) {
7287         firstPoint = (Point) pointArray[i];
7288         secondPoint = (Point) pointArray[i+1];
7289
7290         array = myMethods.drawLineBoolean
7291             (array, firstPoint, secondPoint);
7292     }
7293
7294     return array;
7295 }
7296

```

```

7297 public BufferedImage copyBufferedImage(BufferedImage inputImage) {
7298     ColorModel cm = inputImage.getColorModel();
7299     boolean isAlphaPremultiplied = cm.isAlphaPremultiplied();
7300     WritableRaster raster = inputImage.copyData(null);
7301     return new BufferedImage(cm, raster, isAlphaPremultiplied, null);
7302 }
7303
7304 public void setBooleanArrayValue(boolean[][] array, boolean newValue) {
7305     for(int i=0; i<array.length; i++)
7306         for (int j=0; j<array[0].length; j++)
7307             array[i][j] = newValue;
7308 }
7309
7310 public BufferedImage readImageFile(int subject, int section) {
7311     BufferedImage bufferedImage = myMethods.createErrorImage();
7312     try {
7313         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
7314         bufferedImage = ImageIO.read(adjustedImageFile);
7315     } catch (IOException event) { }
7316     return bufferedImage;
7317 }
7318
7319 public BufferedImage readOriginalImageFile(File imageFile) {
7320     BufferedImage bufferedImage = myMethods.createErrorImage();
7321
7322     try{
7323         FileSeekableStream seekableStream = new FileSeekableStream(imageFile);
7324         RenderedImage src = (RenderedImage)JAI.create("stream", seekableStream);
7325         bufferedImage = PlanarImage.wrapRenderedImage(src).getAsBufferedImage();
7326         bufferedImage = convertBufferedImage(bufferedImage);
7327     }catch (Exception event) { }
7328
7329     return bufferedImage;
7330 }
7331
7332 public BufferedImage convertBufferedImage(BufferedImage input) {
7333     //Copies the pixel values to a new BufferedImage to make sure it is of the right type
7334     BufferedImage output = new
BufferedImage(input.getWidth(),input.getHeight(),BufferedImage.TYPE_INT_RGB);
7335
7336     Raster inputRaster = input.getRaster();
7337     WritableRaster outputRaster = output.getRaster();
7338
7339     int[] tempInts = new int[3];
7340
7341     for (int x=0; x<input.getWidth(); x++)
7342         for (int y=0; y<input.getHeight(); y++) {
7343             tempInts = inputRaster.getPixel(x, y, tempInts);
7344             outputRaster.setPixel(x, y, tempInts);
7345         }
7346
7347     return output;
7348 }
7349
7350 public BufferedImage createErrorImage () {
7351     BufferedImage bufferedImage = new BufferedImage(500,500,BufferedImage.TYPE_INT_RGB);
7352     Graphics g = bufferedImage.getGraphics();
7353     g.setColor(Color.red);
7354     g.drawString("Error loading image", 10, 20);
7355 }

```

```
7356     return bufferedImage;  
7357 }  
7358  
7359 }  
7360 }  
7361  
7362
```