

```

1 /*
2
3 * SectionToVolume is made calculate volumes based on serial sections
4 *
5 * Version 1.0
6 *
7 * Created by Anders Hånell in 2011
8
9 This software is not subject to copyright protection and is in the public domain.
10 SectionToVolume is an experimental system and the author assumes no responsibility
11 whatsoever for its use by other parties, and makes no guarantees, expressed or
12 implied, about its quality, reliability, or any other characteristic.
13
14 Send questions and suggestions to SectionToVolume@gmail.com
15
16 * The first part of the program declares variables and sets up the window.
17 *
18 * Button panel attach code to the buttons in the upper panel.
19 *
20 * PressedW/A/S/D is used for the keyboard control.
21 *
22 * HelpDialog, AboutDialog and AboutBox displays information.
23 *
24 * The CreateProjectDialogs collects information and makes some controls
25 * when creating a new project.
26 *
27 * The class Project holds code used when creating and opening projects.
28 *
29 * TreePanel and the TreeObject classes is used to execute the correct action
30 * when the user clicks the tree panel in the left panel.
31 *
32 * The MiddlePanel and RightSidePanel classes are displayed when selected in
33 * the tree panel
34 *
35 * The ProjectData and SectionData holds the data which is saved to disk.
36 * A copy of the original image file is also saved to disk
37 *
38 * MyMethods holds various methods used in the program.
39
40 */
41 package LesionVolume43_Version1p0;
42
43 import java.awt.*;
44 import java.awt.Cursor;
45 import java.awt.datatransfer.*;
46 import java.awt.event.*;
47 import java.awt.event.KeyEvent;
48 import java.awt.event.KeyListener;
49 import java.awt.geom.Ellipse2D;
50 import java.awt.Graphics2D;
51 import java.awt.image.*;
52
53 import java.io.*;
54 import java.io.File;
55 import java.io.FileOutputStream;
56 import java.io.IOException;

```

```

57 import java.io.ObjectOutputStream;
58 import java.io.Serializable;
59
60 import javax.imageio.ImageIO;
61
62 import javax.swing.*;
63 import javax.swing.JFrame;
64 import javax.swing.JTable;
65 import javax.swing.table.TableColumn;
66 import javax.swing.JTree;
67 import javax.swing.tree.*;
68 import javax.swing.event.TreeSelectionEvent;
69 import javax.swing.event.TreeSelectionListener;
70
71 public class LesionVolume43_Version1p0 extends JFrame implements WindowListener
72 {
73     public static void main(String[] args)
74     {
75         javax.swing.SwingUtilities.invokeLater(new Runnable()
76         {
77             public void run()
78             {
79                 createAndShowGUI();
80             }
81         });
82     }
83
84     static LesionVolume43_Version1p0 frame;
85
86     //These scroll panes are added to split panes when the program is initiated
87     //The upper scrollPane always holds the new, open and save buttons
88     //The left scrollPane always holds the tree structure
89     //The content of the middle and right scrollPane depends on user action
90     static JScrollPane upperScrollPane= new JScrollPane();
91     static JScrollPane leftScrollPane= new JScrollPane();
92     static JScrollPane middleScrollPane = new JScrollPane();
93     static JScrollPane rightScrollPane= new JScrollPane();
94
95     static JSplitPane firstSplitPane;
96     static JSplitPane secondSplitPane;
97     static JSplitPane highLowSplitPane;
98
99     //These to variables are saved to disk
100     //One copy of SectionData is saved for each section in the project
101     //The original image files and the adjusted image files are also saved to disk
102     ProjectData myProjectData = new ProjectData(1,1,1,1);
103     SectionData currentSectionData = new SectionData(1,1);
104
105     LR_DivideMiddlePanel lr_divideMiddlePanel = new LR_DivideMiddlePanel();
106     LR_DivideRightSidePanel lr_DivideRightSidePanel = new LR_DivideRightSidePanel();
107
108     Subject_MiddlePanel subject_middlePanel = new Subject_MiddlePanel();
109     Subject_RightSidePanel subject_rightSidePanel = new Subject_RightSidePanel();
110
111     Section_MiddlePanel section_middlePanel = new Section_MiddlePanel();
112     Section_RightSidePanel section_rightSidePanel = new Section_RightSidePanel();

```

```

113
114 AdjustImage_MiddlePanel adjustImage_middlePanel = new AdjustImage_MiddlePanel();
115
116 //The top panel which holds for example the open and save buttons
117 ButtonPanel buttonPanel = new ButtonPanel();
118
119 SectionResults_MiddlePanel sectionResults_middlePanel = new SectionResults_MiddlePanel();
120 SectionResults_RightSidePanel sectionResults_rightSidePanel = new SectionResults_RightSidePanel();
121
122 SubjectResults_MiddlePanel subjectResults_middlePanel = new SubjectResults_MiddlePanel();
123 SubjectResults_RightSidePanel subjectResults_rightSidePanel = new SubjectResults_RightSidePanel();
124
125 UDA_MiddlePanel uda_middlePanel = new UDA_MiddlePanel();
126 UDA_RightSidePanel uda_rightSidePanel = new UDA_RightSidePanel();
127
128 TissueDetection_MiddlePanel tissueDetection_middlePanel = new TissueDetection_MiddlePanel();
129 TissueDetection_RightSidePanel tissueDetection_rightSidePanel = new TissueDetection_RightSidePanel();
130
131 Conversion_MiddlePanel conversion_middlePanel = new Conversion_MiddlePanel();
132 Conversion_RightSidePanel conversion_rightSidePanel = new Conversion_RightSidePanel();
133
134 AdjustImage_RightSidePanel adjustImage_rightSidePanel = new AdjustImage_RightSidePanel();
135
136 TreePanel treePanel = new TreePanel(1,1,1);
137
138 //This class holds methods for creating new projects and opening existing projects
139 Project myProject = new Project();
140
141 //Various methods are stored here
142 MyMethods myMethods = new MyMethods();
143
144 CreateProjectDialog1 createProjectDialog1 = new CreateProjectDialog1();
145 CreateProjectDialog2 createProjectDialog2 = new CreateProjectDialog2();
146 CreateProjectDialog3 createProjectDialog3 = new CreateProjectDialog3();
147 CreateProjectDialog4 createProjectDialog4 = new CreateProjectDialog4(1);
148 CreateProjectDialog5 createProjectDialog5 = new CreateProjectDialog5();
149
150 HelpDialog helpDialog = new HelpDialog();
151 ErrorDialog errorDialog = new ErrorDialog();
152 AboutBox aboutBox = new AboutBox();
153
154 //Used to keep track of the progress in long tasks
155 ProgressMonitor progressMonitor;
156 ProgressMonitor analyzeFolderMonitor;
157
158 private static void createAndShowGUI()
159 {
160     //Creates the main window and adds split panes and scroll panes
161     frame = new LesionVolume43_Version1p0();
162     frame.setTitle("SectionToVolume");
163     frame.setSize(600,400);
164     frame.setExtendedState(MAXIMIZED_BOTH);
165     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
166
167     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
168     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();

```

```

169
170 firstSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
171     middleScrollPane, rightScrollPane);
172 firstSplitPane.setResizeWeight(0.75);
173 firstSplitPane.setDividerLocation( (int) (screenWidth*14)/20);
174
175 secondSplitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
176     leftScrollPane, firstSplitPane);
177 secondSplitPane.setDividerLocation( (int) (screenWidth*3)/20);
178
179 highLowSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
180     upperScrollPane, secondSplitPane);
181
182 highLowSplitPane.setDividerLocation( (int) screenHeight/10);
183
184 frame.add(highLowSplitPane);
185 firstSplitPane.resetToPreferredSizes();
186 frame.setVisible(true);
187 }
188
189 public LesionVolume43_Version1p0()
190 {
191     //Sets the scroll speed when using the mouse wheel
192     middleScrollPane.getVerticalScrollBar().setUnitIncrement(32);
193
194     upperScrollPane.getViewport().add(buttonPanel);
195
196     //The letters W and S can be used to zoom in and out
197     //The letters A and D can be used to move up and down in the tree structure
198     PressedW pressedW = new PressedW();
199     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
200         KeyStroke.getKeyStroke("W"), "w");
201     upperScrollPane.getActionMap().put("w", pressedW);
202
203     PressedA pressedA = new PressedA();
204     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
205         KeyStroke.getKeyStroke("A"), "a");
206     upperScrollPane.getActionMap().put("a", pressedA);
207
208     PressedS pressedS = new PressedS();
209     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
210         KeyStroke.getKeyStroke("S"), "s");
211     upperScrollPane.getActionMap().put("s", pressedS);
212
213     PressedD pressedD = new PressedD();
214     upperScrollPane.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(
215         KeyStroke.getKeyStroke("D"), "d");
216     upperScrollPane.getActionMap().put("d", pressedD);
217
218     upperScrollPane.requestFocus();
219 }
220
221 public void windowClosing(WindowEvent e)
222 {
223     //The project is always saved before closing
224     myMethods.saveProjectData();

```

```

225     System.exit(0);
226 }
227 public void windowClosed(WindowEvent e) {}
228 public void windowOpened(WindowEvent e) {}
229 public void windowIconified(WindowEvent e) {}
230 public void windowDeiconified(WindowEvent e) {}
231 public void windowActivated(WindowEvent e) {}
232 public void windowDeactivated(WindowEvent e) {}
233 public void windowGainedFocus(WindowEvent e) {}
234 public void windowLostFocus(WindowEvent e) {}
235 public void windowStateChanged(WindowEvent e) {}
236
237 private class ButtonPanel extends JPanel implements ActionListener
238 {
239     //This panel is located on the upper scroll pane
240     JButton newProjectButton;
241     JButton saveProjectButton;
242     JButton openProjectButton;
243     JButton zoomInButton;
244     JButton zoomOutButton;
245     JButton decLineWidthButton;
246     JButton incLineWidthButton;
247     JButton blackLinesButton;
248     JButton whiteLinesButton;
249     JButton helpButton;
250     JButton aboutButton;
251
252     public ButtonPanel()
253     {
254         newProjectButton=new JButton("New Project");
255         newProjectButton.addActionListener(this);
256         saveProjectButton=new JButton("Save Project");
257         saveProjectButton.addActionListener(this);
258         openProjectButton=new JButton("Open Project");
259         openProjectButton.addActionListener(this);
260         zoomInButton=new JButton("Zoom In");
261         zoomInButton.addActionListener(this);
262         zoomOutButton=new JButton("Zoom Out");
263         zoomOutButton.addActionListener(this);
264         decLineWidthButton=new JButton("- Line Width");
265         decLineWidthButton.addActionListener(this);
266         incLineWidthButton=new JButton("+ Line Width");
267         incLineWidthButton.addActionListener(this);
268         blackLinesButton =new JButton("Black lines");
269         blackLinesButton.addActionListener(this);
270         whiteLinesButton=new JButton("White lines");
271         whiteLinesButton.addActionListener(this);
272         helpButton=new JButton("Help");
273         helpButton.addActionListener(this);
274         aboutButton=new JButton("About");
275         aboutButton.addActionListener(this);
276
277         this.setLayout(new GridBagLayout());
278         GridBagConstraints c = new GridBagConstraints();
279
280         c.gridx = 0;

```

```

281     c.gridy = 0;
282     c.gridwidth = 1;
283     c.insets = new Insets(5,5,5,5);
284     this.add(newProjectButton, c);
285
286     c.gridx++;
287     this.add(saveProjectButton, c);
288
289     c.gridx++;
290     this.add(openProjectButton, c);
291
292     c.insets = new Insets(5,40,5,5);
293     c.gridx++;
294     this.add(zoomInButton, c);
295
296     c.insets = new Insets(5,5,5,5);
297     c.gridx++;
298     this.add(zoomOutButton, c);
299
300     c.insets = new Insets(5,40,5,5);
301     c.gridx++;
302     this.add(decLineWidthButton, c);
303
304     c.insets = new Insets(5,5,5,5);
305     c.gridx++;
306     this.add(incLineWidthButton, c);
307
308     c.insets = new Insets(5,40,5,5);
309     c.gridx++;
310     this.add(blackLinesButton, c);
311
312     c.insets = new Insets(5,5,5,5);
313     c.gridx++;
314     this.add(whiteLinesButton, c);
315
316     c.insets = new Insets(5,40,5,5);
317     c.gridx++;
318     this.add(helpButton, c);
319
320     c.insets = new Insets(5,5,5,5);
321     c.gridx++;
322     this.add(aboutButton, c);
323 }
324
325 public void actionPerformed(ActionEvent e)
326 {
327     if (e.getSource() == newProjectButton)
328     {
329         addWindowListener(frame);
330         createProjectDialog1.setVisible(true);
331     }
332
333     if (e.getSource() == openProjectButton)
334     {
335         addWindowListener(frame);
336

```

```

337 //The file dialog uses a file filter to hide any file except main project files
338 FileFilter fileFilter = new FileFilter();
339 File file = new File("");
340 JFileChooser fc = new JFileChooser(file);
341 fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
342 fc.setFileFilter(fileFilter);
343 int dialogResult = fc.showOpenDialog(this);
344
345 if (dialogResult==JFileChooser.APPROVE_OPTION)
346 {
347     File projectFile = fc.getSelectedFile();
348
349     FileInputStream fis = null;
350     ObjectInputStream in = null;
351     try
352     {
353         fis = new FileInputStream(projectFile);
354         in = new ObjectInputStream(fis);
355         myProjectData = (ProjectData)in.readObject();
356         in.close();
357         myProjectData.projectLocation=projectFile.getParent();
358     }
359     catch(IOException ex)
360     {
361         ex.printStackTrace();
362     }
363     catch(ClassNotFoundException ex){ }
364
365     myProject.refreshWhenOpened(); //Sets up the tree structure for example
366 }
367 }
368
369 if (e.getSource() == saveProjectButton)
370 {
371     myMethods.saveProjectData();
372 }
373
374 // If the zoom factor is for example 47 images will be shown at 47% of their original size
375 if (e.getSource() == zoomInButton)
376 {
377     myProjectData.zoomFactor= myProjectData.zoomFactor+10;
378     middleScrollPane.repaint();
379     middleScrollPane.revalidate();
380 }
381 if (e.getSource() == zoomOutButton)
382 {
383     if (myProjectData.zoomFactor>10) myProjectData.zoomFactor= myProjectData.zoomFactor-10;
384     middleScrollPane.repaint();
385     middleScrollPane.revalidate();
386 }
387
388 //When zoomed out thin lines might not be displayed
389 //but by increasing the line width this is avoided
390 //The setting for the line width does not affect
391 //the area calculations
392 if (e.getSource() == decLineWidthButton)

```

```

393     {
394         myMethods.setWaitCursor();
395
396         if (myProjectData.lineWidth>0) myProjectData.lineWidth--;
397         lr_divideMiddlePanel.generateLR_LinesImage();
398         lr_divideMiddlePanel.generateLR_AreasImage();
399         uda_middlePanel.generateUdaPanelImage();
400         conversion_middlePanel.generateConversionImage();
401         middleScrollPane.repaint();
402         myMethods.setCustomCursor();
403     }
404     if (e.getSource() == incLineWidthButton)
405     {
406         myMethods.setWaitCursor();
407         myProjectData.lineWidth++;
408         lr_divideMiddlePanel.generateLR_LinesImage();
409         lr_divideMiddlePanel.generateLR_AreasImage();
410         uda_middlePanel.generateUdaPanelImage();
411         conversion_middlePanel.generateConversionImage();
412         middleScrollPane.repaint();
413         myMethods.setCustomCursor();
414     }
415
416     if (e.getSource() == blackLinesButton)
417     {
418         myMethods.setWaitCursor();
419         int[] black = {0,0,0};
420         myProjectData.lineColor= black;
421
422         try
423         {
424             lr_divideMiddlePanel.generateLR_LinesImage();
425             lr_divideMiddlePanel.generateLR_AreasImage();
426         }catch (Exception error){ }
427
428         try
429         {
430             uda_middlePanel.generateUdaPanelImage();
431         }catch (Exception error){ }
432
433         try
434         {
435             conversion_middlePanel.generateConversionImage();
436         }catch (Exception error){ }
437
438         middleScrollPane.repaint();
439         myMethods.setCustomCursor();
440     }
441
442     if (e.getSource() == whiteLinesButton)
443     {
444         myMethods.setWaitCursor();
445         int[] white = {255,255,255};
446         myProjectData.lineColor= white;
447
448         try

```



```

449     {
450         lr_divideMiddlePanel.generateLR_LinesImage();
451         lr_divideMiddlePanel.generateLR_AreasImage();
452     }catch (Exception error){ }
453
454     try
455     {
456         uda_middlePanel.generateUdaPanelImage();
457     }catch (Exception error){ }
458
459     try
460     {
461         conversion_middlePanel.generateConversionImage();
462     }catch (Exception error){ }
463
464     middleScrollPane.repaint();
465     myMethods.setCustomCursor();
466 }
467
468
469 if (e.getSource() == helpButton)
470 {
471     //Pressing the main help button displays the entire help text
472     String helpText=myMethods.getHelpTextChoseProjectFolder();
473     helpText=helpText+"\n'+'\n';
474     helpText=helpText+myMethods.getHelpTextChoseSaveFolder();
475     helpText=helpText+"\n'+'\n';
476     helpText=helpText+myMethods.getHelpTextChoseConversionImage();
477     helpText=helpText+"\n'+'\n';
478     helpText=helpText+myMethods.getHelpTextBregmaLevels();
479     helpText=helpText+"\n'+'\n';
480     helpText=helpText+myMethods.getHelpTextUDA();
481     helpText=helpText+"\n'+'\n';
482     helpText=helpText+myMethods.getHelpTextTissueDetection();
483     helpText=helpText+"\n'+'\n';
484     helpText=helpText+myMethods.getHelpTextAdjustImage();
485
486     helpDialog.newText(helpText);
487 }
488 if (e.getSource() == aboutButton)
489 {
490     aboutBox.newText(myMethods.getAboutText());
491 }
492 }
493
494 class FileFilter extends javax.swing.filechooser.FileFilter
495 {
496     //Used when navigating to project files when opening an existing project
497     //Only displays folders and the main project file
498     public boolean accept(File f)
499     {
500         if (f.isDirectory()) return true;
501         return (f.getName().equals("ProjectData.ser"));
502     }
503     public String getDescription()
504     {

```

```

505         return "Project files";
506     }
507 }
508 }
509
510 class PressedW extends AbstractAction
511 {
512     //Keyboard "W" is pressed, zooms in on the image
513     public void actionPerformed(ActionEvent e)
514     {
515         if (!myMethods.textFieldIsFocusOwner())
516         {
517             myProjectData.zoomFactor= myProjectData.zoomFactor+10;
518             middleScrollPane.repaint();
519             middleScrollPane.revalidate();
520         }
521     }
522 }
523 class PressedA extends AbstractAction
524 {
525     //Keyboard "A" is pressed, moves one step up in the tree panel
526     public void actionPerformed(ActionEvent e)
527     {
528         if (!myMethods.textFieldIsFocusOwner())
529         {
530             treePanel.tree.removeTreeSelectionListener(treePanel);
531             TreePath selectionPath = treePanel.tree.getSelectionPath();
532
533             int rowNumber = treePanel.tree.getRowForPath(selectionPath);
534             treePanel.tree.expandRow(rowNumber);
535
536             if (rowNumber<1) rowNumber=1;
537             else rowNumber--;
538
539             TreePath newSelection = treePanel.tree.getPathForRow(rowNumber);
540
541             treePanel.tree.addTreeSelectionListener(treePanel);
542             treePanel.tree.setSelectionPath(newSelection);
543         }
544     }
545 }
546 class PressedS extends AbstractAction
547 {
548     //Keyboard "S" is pressed, zooms out in the image
549     public void actionPerformed(ActionEvent e)
550     {
551         if (!myMethods.textFieldIsFocusOwner())
552         {
553             if (myProjectData.zoomFactor>10) myProjectData.zoomFactor= myProjectData.zoomFactor-10;
554             middleScrollPane.repaint();
555             middleScrollPane.revalidate();
556         }
557     }
558 }
559 class PressedD extends AbstractAction
560 {

```

```

561 //Keyboard "D" is pressed, moves one step down in the tree panel
562 public void actionPerformed(ActionEvent e)
563 {
564     if (!myMethods.textFieldIsFocusOwner())
565     {
566         treePanel.tree.removeTreeSelectionListener(treePanel);
567         TreePath selectionPath = treePanel.tree.getSelectionPath();
568
569         int rowNumber = treePanel.tree.getRowForPath(selectionPath);
570         treePanel.tree.expandRow(rowNumber);
571         int rowsDisplayed = treePanel.tree.getRowCount();
572
573         if (rowNumber==rowsDisplayed) rowNumber=1;
574         else rowNumber++;
575
576         TreePath newSelection = treePanel.tree.getPathForRow(rowNumber);
577
578         treePanel.tree.addTreeSelectionListener(treePanel);
579         treePanel.tree.setSelectionPath(newSelection);
580     }
581 }
582 }
583
584 public class HelpDialog extends JFrame
585 {
586     //Several buttons can activate the help dialog.
587     //The Help button on the upper panel will show the entire help text.
588     //The other Help buttons will only show the section relevant for
589     //that part of the program.
590
591     JTextArea textArea = new JTextArea(30, 30);
592     JScrollPane scrollPane;
593
594     public HelpDialog()
595     {
596         this.setVisible(false);
597         this.setTitle("Help");
598
599         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
600         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
601         this.setLocation(screenWidth*50/100,screenHeight*10/100);
602         this.setSize(screenWidth*45/100,screenHeight*80/100);
603
604         this.setAlwaysOnTop(true);
605         this.setModalExclusionType(Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
606
607         textArea = new JTextArea(5, 20);
608         scrollPane = new JScrollPane(textArea);
609         textArea.setEditable(false);
610         this.add(scrollPane);
611     }
612
613     public void newText(String inputText)
614     {
615         //Updates the help text when the user press one of the help buttons
616         helpDialog.remove(scrollPane);

```

```

617
618     textArea = new JTextArea(5, 20);
619     textArea.setText(inputText);
620     scrollPane = new JScrollPane(textArea);
621     textArea.setEditable(false);
622
623     this.add(scrollPane);
624     this.setVisible(true);
625 }
626 }
627
628 public class AboutBox extends JFrame
629 {
630     JTextArea textArea = new JTextArea(30, 30);
631     JScrollPane scrollPane;
632
633     public AboutBox()
634     {
635         this.setVisible(false);
636         this.setTitle("About SectionToVolume");
637
638         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
639         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
640         this.setLocation(screenWidth*5/100,screenHeight*10/100);
641         this.setSize(screenWidth*40/100,screenHeight*80/100);
642
643         this.setAlwaysOnTop(true);
644         this.setModalExclusionType(Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
645
646         textArea = new JTextArea(5, 20);
647         scrollPane = new JScrollPane(textArea);
648         textArea.setEditable(false);
649         this.add(scrollPane);
650     }
651     public void newText(String inputText)
652     {
653         helpDialog.remove(scrollPane);
654
655         textArea = new JTextArea(5, 20);
656         textArea.setText(inputText);
657         scrollPane = new JScrollPane(textArea);
658         textArea.setEditable(false);
659
660         this.add(scrollPane);
661         this.setVisible(true);
662     }
663 }
664
665 public class ErrorDialog extends JFrame
666 {
667     JTextArea textArea = new JTextArea(30, 30);
668     JScrollPane scrollPane;
669
670     boolean firstError=true;
671     String errorText= new String("");
672

```

```

673 public ErrorDialog()
674 {
675     this.setVisible(false);
676
677     this.setTitle("An error occured");
678     this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
679
680     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
681     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
682     this.setLocation(screenWidth*5/100,screenHeight*10/100);
683     this.setSize(screenWidth*40/100,screenHeight*80/100);
684
685     this.setAlwaysOnTop(true);
686
687     this.setModalExclusionType(Dialog.ModalExclusionType.APPLICATION_EXCLUDE);
688
689     textArea = new JTextArea(5, 20);
690     scrollPane = new JScrollPane(textArea);
691     textArea.setEditable(false);
692
693     this.add(scrollPane);
694 }
695
696 public void newText(String inputText)
697 {
698     errorDialog.remove(scrollPane);
699
700     if (firstError)
701     {
702         errorText=inputText;
703         firstError=false;
704     }
705     else
706     {
707         errorText=errorText+"\n'"+'\n'+'\n'+inputText;
708     }
709
710     textArea = new JTextArea(5, 20);
711     textArea.setText(errorText);
712     scrollPane = new JScrollPane(textArea);
713     textArea.setEditable(false);
714
715     this.add(scrollPane);
716     this.setVisible(true);
717 }
718 }
719
720 private class CreateProjectDialog1 extends JDialog implements ActionListener
721 {
722     //This dialog is used to locate the folder which holds the source data
723     //It checks that it contains images which can be used to create a project
724     File sourceFileLocation;
725     File[] subjectFiles = new File[1000];
726     File[][] sectionFiles = new File[1000][1000];
727
728     int numberOfSubjects=0;

```

```

729 String[] subjectNames = new String[1000];
730 int numberOfSections=0;
731 String[][] sectionNames = new String[1000][1000];
732 int highestNumberOfSectionsInSubject;
733
734 boolean[][] sectionExists = new boolean[1000][1000];
735
736 JButton helpButton;
737 JButton cancelButton;
738 JButton nextButton;
739 JButton ImagesLocationButton;
740
741 JLabel infoLabel = new JLabel("The project folder should contain one folder for each subject (subject
folder)");
742 JLabel infoLabel2 = new JLabel("Each subject folder should contain one image file for each section");
743 JLabel imageLocationLabel = new JLabel("Selected folder:");
744 JLabel detectedSubjectsLabel = new JLabel("Detected Subjects: ");
745 JLabel detectedSectionsLabel = new JLabel("Detected Sections: ");
746 JLabel statusLabel = new JLabel("Status: No folder selected");
747
748 boolean allImagesIsOfSameSize;
749 int detectedImageWidth;
750 int detectedImageHeight;
751
752 JPanel panel = new JPanel();
753
754 public CreateProjectDialog1()
755 {
756     this.setTitle("Choose source images");
757     this.setModalityType(ModalityType.APPLICATION_MODAL);
758
759     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
760     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
761     this.setLocation(screenWidth*10/100,screenHeight*10/100);
762     this.setSize(screenWidth*35/100,screenHeight*80/100);
763
764     for (int i=0; i<1000; i++)
765         for (int j=0; j<1000; j++) sectionExists[i][j]=false;
766
767     helpButton = new JButton("Help");
768     helpButton.addActionListener(this);
769     ImagesLocationButton=new JButton("Select project folder");
770     ImagesLocationButton.addActionListener(this);
771     nextButton=new JButton("Next");
772     nextButton.addActionListener(this);
773     cancelButton=new JButton("Cancel");
774     cancelButton.addActionListener(this);
775     nextButton.setEnabled(false);
776     statusLabel.setForeground(Color.red);
777
778     panel.setLayout(new GridBagLayout());
779     GridBagConstraints c = new GridBagConstraints();
780
781     c.gridx = 0;
782     c.gridy = 0;
783     c.anchor = GridBagConstraints.PAGE_START;

```

```

784     c.gridwidth=3;
785     c.insets = new Insets(20,5,5,5);
786     panel.add(infoLabel, c);
787
788     c.insets = new Insets(5,5,5,5);
789     c.gridy++;
790     panel.add(infoLabel2, c);
791
792     c.gridy++;
793     c.insets = new Insets(20,5,5,5);
794     panel.add(ImagesLocationButton, c);
795
796     c.insets = new Insets(5,5,5,5);
797     c.gridy++;
798     panel.add(imageLocationLabel, c);
799
800     c.insets = new Insets(20,5,5,5);
801     c.gridy++;
802     panel.add(detectedSubjectsLabel, c);
803
804     c.insets = new Insets(5,5,5,5);
805     c.gridy++;
806     panel.add(detectedSectionsLabel, c);
807
808     c.insets = new Insets(20,5,5,5);
809     c.gridy++;
810     panel.add(statusLabel, c);
811
812     c.weightx=0.3333;
813     c.gridy++;
814     c.gridwidth=1;
815     c.insets = new Insets(20,5,5,5);
816     c.anchor=GridBagConstraints.FIRST_LINE_END;
817     c.gridx=0;
818     panel.add(cancelButton, c);
819
820     c.anchor = GridBagConstraints.PAGE_START;
821     c.gridx=1;
822     panel.add(helpButton, c);
823
824     c.anchor = GridBagConstraints.FIRST_LINE_START;
825     c.weighty=1;
826     c.gridx=2;
827     panel.add(nextButton, c);
828
829     this.getContentPane().removeAll();
830     this.add(new JScrollPane(panel));
831 }
832
833 public void actionPerformed(ActionEvent e)
834 {
835     if (e.getSource()==helpButton)
836     {
837         String helpText = myMethods.getHelpTextChoseProjectFolder();
838         helpDialog.newText(helpText);
839     }

```

```

840     if (e.getSource()==cancelButton)
841     {
842         this.setVisible(false);
843     }
844     if (e.getSource()==ImagesLocationButton)
845     {
846         File file = new File("");
847         JFileChooser fc = new JFileChooser(file);
848         fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
849         int dialogResult = fc.showOpenDialog(this);
850
851         if (dialogResult==JFileChooser.APPROVE_OPTION)
852         {
853             createProjectDialog1.setVisible(false);
854
855             sourceFileLocation = fc.getSelectedFile();
856
857             //When the user selects a folder this makes sure that it can be used to create a project
858             analyzeFolderMonitor = new ProgressMonitor(this,
859                 "Checking folder",
860                 "Folders checked: 0", 0, myMethods.countNumberOfFiles(sourceFileLocation));
861             analyzeFolderMonitor.setMillisToDecideToPopup(0);
862             analyzeFolderMonitor.setMillisToPopup(0);
863
864             CheckForSections checkForSections = new CheckForSections();
865             checkForSections.execute();
866         }
867     }
868     if (e.getSource()==nextButton)
869     {
870         this.setVisible(false);
871         createProjectDialog2.setVisible(true);
872     }
873 }
874
875 public class CheckForSections extends SwingWorker<Void, Void>
876 {
877     @Override
878     public Void doInBackground()
879     {
880         //Checks that the folder contains images, that the images can be read
881         //and that they are of the same size
882
883         frame.setVisible(false);
884         File[] firstLevelFiles = new File[1000];
885         File[][] secondLevelFiles = new File[1000][1000];
886         int filesChecked=0;
887         highestNumberOfSectionsInSubject=0;
888
889         for (int i=0; i<1000; i++)
890             for (int j=0; j<1000; j++) sectionExists[i][j]=false;
891
892         analyzeFolderMonitor.setProgress(filesChecked);
893         analyzeFolderMonitor.setNote("Folders checked: 0");
894
895         String openLocation = sourceFileLocation.getName();

```



```

896         imageLocationLabel.setText("Selected folder: "+openLocation);
897
898     firstLevelFiles = sourceFileLocation.listFiles();
899     numberOfSubjects=0;
900     for (int i=0; i<firstLevelFiles.length; i++)
901     {
902         //Makes sure that only directories are copied to the list of subjects
903         if (firstLevelFiles[i].isDirectory())
904         {
905             subjectFiles[numberOfSubjects]=firstLevelFiles[i];
906             subjectNames[numberOfSubjects]=firstLevelFiles[i].getName();
907             numberOfSubjects++;
908         }
909         analyzeFolderMonitor.setNote("Folders checked: "+i);
910         filesChecked++;
911         analyzeFolderMonitor.setProgress(filesChecked);
912         if (analyzeFolderMonitor.isCanceled())
913         {
914             break;
915         }
916     }
917     detectedSubjectsLabel.setText("Detected Subjects: "+numberOfSubjects);
918
919     analyzeFolderMonitor.setNote("Files checked: 0");
920     int sectionsInSubjectCounter;
921
922     numberOfSections=0;
923     allImagesIsOfSameSize=true;
924     detectedImageWidth=-1;
925     detectedImageHeight=-1;
926
927     for (int i=0; i<numberOfSubjects; i++)
928     {
929         sectionsInSubjectCounter=0;
930         secondLevelFiles[i]=subjectFiles[i].listFiles();
931         for (int j=0; j<secondLevelFiles[i].length; j++)
932         {
933             if (secondLevelFiles[i][j].isFile())
934             {
935                 if (myMethods.fileIsValidImage(secondLevelFiles[i][j]))
936                 {
937                     sectionFiles[i][sectionsInSubjectCounter]=secondLevelFiles[i][j];
938                     sectionNames[i][sectionsInSubjectCounter]=secondLevelFiles[i][j].getName();
939                     sectionExists[i][sectionsInSubjectCounter]=true;
940                     numberOfSections++;
941                     sectionsInSubjectCounter++;
942                 }
943             }
944             filesChecked++;
945             analyzeFolderMonitor.setProgress(filesChecked);
946             analyzeFolderMonitor.setNote("Files checked: "+filesChecked);
947             if (sectionsInSubjectCounter>highestNumberOfSectionsInSubject)
948                 highestNumberOfSectionsInSubject=sectionsInSubjectCounter;
949             if (analyzeFolderMonitor.isCanceled())
950             {
951                 break;

```

```

952         }
953     }
954     if (analyzeFolderMonitor.isCanceled())
955     {
956         break;
957     }
958 }
959 detectedSectionsLabel.setText("Detected Sections: "+numberOfSections);
960
961 if (analyzeFolderMonitor.isCanceled())
962 {
963     nextButton.setEnabled(false);
964     statusLabel.setForeground(Color.red);
965     statusLabel.setText("Status: Cancelled by user");
966 }
967 else if (numberOfSections==0)
968 {
969     nextButton.setEnabled(false);
970     statusLabel.setForeground(Color.red);
971     statusLabel.setText("Status: No sections detected");
972 }
973 else if (!allImagesIsOfSameSize)
974 {
975     nextButton.setEnabled(false);
976     statusLabel.setForeground(Color.red);
977     statusLabel.setText("All images has to be of the same size");
978 }
979 else
980 {
981     nextButton.setEnabled(true);
982     statusLabel.setForeground(Color.green);
983     statusLabel.setText("Status: Ok to proceed");
984 }
985
986 createProjectDialog4 = new CreateProjectDialog4(highestNumberOfSectionsInSubject);
987
988
989 frame.setVisible(true);
990 frame.repaint();
991 createProjectDialog1.setVisible(true);
992
993 return null;
994 }
995
996 @Override
997 public void done() {}
998 }
999 }
1000
1001 private class CreateProjectDialog2 extends JDialog implements ActionListener
1002 {
1003     //Used to locate a folder where project data should be saved.
1004     //Makes sure that the folder is empty since a lot of files
1005     //are stored in this folder
1006     String result;
1007     File saveFileLocation;

```

```
1008
1009 JLabel infoLabel = new JLabel("The folder where the project should be saved has to be empty");
1010 JLabel saveLocationLabel = new JLabel("Selected folder:");
1011 JLabel statusLabel = new JLabel("Status: No Folder Selected");
1012
1013 JButton saveLocationButton;
1014 JButton previousButton;
1015 JButton nextButton;
1016 JButton helpButton;
1017
1018 JPanel panel = new JPanel();
1019
1020 public CreateProjectDialog2()
1021 {
1022     this.setTitle("Choose save location");
1023     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1024     int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1025     this.setLocation(screenWidth*10/100,screenHeight*10/100);
1026     this.setSize(screenWidth*35/100,screenHeight*80/100);
1027
1028     this.setModalityType(ModalityType.APPLICATION_MODAL);
1029
1030     saveLocationButton=new JButton("Select save folder");
1031     saveLocationButton.addActionListener(this);
1032     previousButton=new JButton("Previous");
1033     previousButton.addActionListener(this);
1034     nextButton=new JButton("Next");
1035     nextButton.addActionListener(this);
1036     nextButton.setEnabled(false);
1037     helpButton=new JButton("Help");
1038     helpButton.addActionListener(this);
1039
1040     statusLabel.setBackground(Color.black);
1041     statusLabel.setForeground(Color.red);
1042
1043     panel.setLayout(new GridBagLayout());
1044     GridBagConstraints c = new GridBagConstraints();
1045
1046     c.gridx = 0;
1047     c.gridy = 0;
1048     c.gridwidth = 3;
1049     c.anchor = GridBagConstraints.PAGE_START;
1050     c.insets = new Insets(20,5,5,5);
1051     panel.add(infoLabel, c);
1052
1053     c.gridy++;
1054     panel.add(saveLocationButton, c);
1055
1056     c.insets = new Insets(5,5,5,5);
1057     c.gridy++;
1058     panel.add(saveLocationLabel, c);
1059
1060     c.insets = new Insets(20,5,5,5);
1061     c.gridy++;
1062     panel.add(statusLabel, c);
1063
```

```

1064     c.weighty=1;
1065     c.weightx=0.3333;
1066     c.gridy++;
1067     c.gridwidth = 1;
1068     c.gridx=0;
1069     c.anchor=GridBagConstraints.FIRST_LINE_END;
1070     panel.add(previousButton, c);
1071
1072     c.anchor = GridBagConstraints.PAGE_START;
1073     c.gridx = 1;
1074     panel.add(helpButton, c);
1075
1076     c.anchor=GridBagConstraints.FIRST_LINE_START;
1077     c.gridx = 2;
1078     panel.add(nextButton, c);
1079
1080     this.getContentPane().removeAll();
1081     this.add(new JScrollPane(panel));
1082 }
1083
1084 public void actionPerformed(ActionEvent e)
1085 {
1086     if (e.getSource()==saveLocationButton)
1087     {
1088         File[] filesInFolder = new File[1000];
1089
1090         File file = new File("");
1091         JFileChooser fc = new JFileChooser(file);
1092
1093         fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
1094         int dialogResult = fc.showOpenDialog(this);
1095
1096         if (dialogResult==JFileChooser.APPROVE_OPTION)
1097         {
1098             saveFileLocation = fc.getSelectedFile();
1099             String openLocation = saveFileLocation.getName();
1100             saveLocationLabel.setText("Selected folder: "+openLocation);
1101
1102             int numberOfFilesInFolder=0;
1103             if (saveFileLocation.isDirectory())
1104             {
1105                 filesInFolder = saveFileLocation.listFiles();
1106                 numberOfFilesInFolder = filesInFolder.length;
1107             }
1108             if (!saveFileLocation.isDirectory())
1109             {
1110                 statusLabel.setForeground(Color.red);
1111                 statusLabel.setText("Status: Not a folder");
1112                 nextButton.setEnabled(false);
1113             }
1114             if (saveFileLocation.isDirectory() && numberOfFilesInFolder>0)
1115             {
1116                 nextButton.setEnabled(false);
1117                 statusLabel.setForeground(Color.red);
1118                 statusLabel.setText("Status: Folder is not empty");
1119             }

```

```

1120         if (saveFileLocation.isDirectory() && numberOfFilesInFolder==0)
1121         {
1122             nextButton.setEnabled(true);
1123             statusLabel.setText("Status: Ok to proceed");
1124             statusLabel.setForeground(Color.green);
1125         }
1126     }
1127 }
1128 if (e.getSource()==helpButton)
1129 {
1130     String helpText = myMethods.getHelpTextChoseSaveFolder();
1131     helpDialog.newText(helpText);
1132 }
1133
1134 if (e.getSource()==previousButton)
1135 {
1136     this.setVisible(false);
1137     createProjectDialog1.setVisible(true);
1138 }
1139
1140 if (e.getSource()==nextButton)
1141 {
1142     this.setVisible(false);
1143     createProjectDialog3.setVisible(true);
1144 }
1145 }
1146 }
1147
1148 private class CreateProjectDialog3 extends JDialog implements ActionListener
1149 {
1150     //Locates an image used to convert pixels to mm. Makes sure that it is a
1151     //readable image of the same size as the images used to create the project
1152     File conversionFileLocation;
1153     JLabel infoLabel = new JLabel("The conversion image is used to convert pixels to mm.");
1154     JLabel infoLabel2 = new JLabel("It preferably contains an object of known size.");
1155     JLabel locationLabel = new JLabel("Selected file:");
1156     JLabel statusLabel = new JLabel("Status: No file selected");
1157
1158     JButton helpButton;
1159     JButton conversionButton;
1160     JButton previousButton;
1161     JButton nextButton;
1162
1163     JPanel panel = new JPanel();
1164
1165     public CreateProjectDialog3()
1166     {
1167         this.setTitle("Choose conversion image");
1168         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1169         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1170         this.setLocation(screenWidth*10/100,screenHeight*10/100);
1171         this.setSize(screenWidth*35/100,screenHeight*80/100);
1172
1173         this.setModalityType(ModalityType.APPLICATION_MODAL);
1174
1175         helpButton=new JButton("Help");

```

```
1176     helpButton.addActionListener(this);
1177     conversionButton=new JButton("Select Conversion Image");
1178     conversionButton.addActionListener(this);
1179     previousButton=new JButton("Previous");
1180     previousButton.addActionListener(this);
1181     nextButton=new JButton("Next");
1182     nextButton.addActionListener(this);
1183     nextButton.setEnabled(false);
1184     statusLabel.setForeground(Color.red);
1185
1186     panel.setLayout(new GridBagLayout());
1187     GridBagConstraints c = new GridBagConstraints();
1188
1189     c.gridx = 0;
1190     c.gridy = 0;
1191     c.gridwidth = 2;
1192     c.anchor = GridBagConstraints.PAGE_START;
1193     c.insets = new Insets(15,5,5,5);
1194     c.gridwidth = 3;
1195     panel.add(infoLabel, c);
1196
1197     c.insets = new Insets(5,5,5,5);
1198     c.gridy++;
1199     panel.add(infoLabel2, c);
1200
1201     c.insets = new Insets(20,5,5,5);
1202     c.gridy++;
1203     panel.add(conversionButton, c);
1204
1205     c.insets = new Insets(5,5,5,5);
1206     c.gridy++;
1207     panel.add(locationLabel, c);
1208
1209     c.insets = new Insets(20,5,5,5);
1210     c.gridy++;
1211     panel.add(statusLabel, c);
1212
1213     c.gridy++;
1214     c.gridx=0;
1215     c.weightx=0.3333;
1216     c.gridwidth = 1;
1217     c.anchor=GridBagConstraints.FIRST_LINE_END;
1218     panel.add(previousButton, c);
1219
1220     c.gridx=1;
1221     c.anchor = GridBagConstraints.PAGE_START;
1222     panel.add(helpButton, c);
1223
1224     c.anchor=GridBagConstraints.FIRST_LINE_START;
1225     c.weighty=1;
1226     c.gridx = 2;
1227     c.gridwidth = 1;
1228     panel.add(nextButton, c);
1229
1230     this.getContentPane().removeAll();
1231     this.add(new JScrollPane(panel));
```

```

1232     }
1233
1234     public void actionPerformed(ActionEvent e)
1235     {
1236         if (e.getSource()==helpButton)
1237         {
1238             String helpText = myMethods.getHelpTextChoseConversionImage();
1239             helpDialog.newText(helpText);
1240         }
1241
1242         if (e.getSource()==conversionButton)
1243         {
1244             File file = new File("");
1245             JFileChooser fc = new JFileChooser(file);
1246             fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
1247             int dialogResult = fc.showOpenDialog(this);
1248
1249             if (dialogResult==JFileChooser.APPROVE_OPTION)
1250             {
1251                 conversionFileLocation = fc.getSelectedFile();
1252                 String openLocation = conversionFileLocation.getName();
1253                 locationLabel.setText("Selected file: "+openLocation);
1254
1255                 createProjectDialog1.allImagesIsOfSameSize=true;
1256
1257                 if (myMethods.fileIsValidImage(conversionFileLocation))
1258                 {
1259                     if (createProjectDialog1.allImagesIsOfSameSize)
1260                     {
1261                         statusLabel.setText("Ok to proceed");
1262                         nextButton.setEnabled(true);
1263                         statusLabel.setForeground(Color.green);
1264                     }
1265                     else
1266                     {
1267                         statusLabel.setText("Has to be of same size as section images");
1268                         nextButton.setEnabled(false);
1269                         statusLabel.setForeground(Color.red);
1270                     }
1271                 }
1272                 else
1273                 {
1274                     statusLabel.setText("Status: Could not read image format");
1275                     statusLabel.setForeground(Color.red);
1276                     nextButton.setEnabled(false);
1277                 }
1278             }
1279         }
1280
1281         if (e.getSource()==previousButton)
1282         {
1283             this.setVisible(false);
1284             createProjectDialog2.setVisible(true);
1285         }
1286
1287         if (e.getSource()==nextButton)

```

```

1288     {
1289         this.setVisible(false);
1290         createProjectDialog4.setVisible(true);
1291     }
1292 }
1293 }
1294
1295 private class CreateProjectDialog4 extends JDialog implements ActionListener
1296 {
1297     //Determines the levels to be associated with each image
1298     //Makes sure that user entry is numbers and that the levels
1299     //are either continuously increasing or decreasing
1300     String result;
1301
1302     double[] levels;
1303     double firstLevel=0;
1304     double distance=1;
1305
1306     JLabel firstInfo = new JLabel("Use this for equally spaced sections:");
1307     JLabel info = new JLabel("Otherwise enter levels manually here:");
1308
1309     JLabel firstLevelLabel = new JLabel("First bregma level:");
1310     JTextField firstLevelTextField = new JTextField("0", 5);
1311     JLabel firstLevelResultLabel = new JLabel("0");
1312
1313     JLabel distanceLabel = new JLabel("Distance between sections:");
1314     JTextField distanceTextField = new JTextField("1", 5);
1315     JLabel distanceResultLabel = new JLabel("0");
1316
1317     JButton calculateButton;
1318     JButton helpButton;
1319     JButton previousButton;
1320     JButton nextButton;
1321
1322     int numberOfBregmaLevels;
1323
1324     JLabel[] bregmaLevelLabel;
1325     JTextField[] bregmaLevelTextFields ;
1326     JLabel[] bregmaLevelResultLabel;
1327
1328     JLabel statusLabel = new JLabel("Ok to proceed");
1329
1330     JPanel panel = new JPanel();
1331
1332     boolean allOK =true;
1333
1334     public CreateProjectDialog4(int initNumberOfBregmaLevels)
1335     {
1336         this.setTitle("Enter bregma levels");
1337         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1338         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1339         this.setLocation(screenWidth*10/100,screenHeight*10/100);
1340         this.setSize(screenWidth*35/100,screenHeight*80/100);
1341
1342         numberOfBregmaLevels=initNumberOfBregmaLevels;
1343         levels = new double[numberOfBregmaLevels];

```



```

1344
1345 bregmaLevelLabel = new JLabel[numberOfBregmaLevels];
1346 bregmaLevelTextFields = new JTextField[numberOfBregmaLevels];
1347 bregmaLevelResultLabel = new JLabel[numberOfBregmaLevels];
1348
1349 statusLabel.setForeground(Color.green);
1350
1351 KeyListener keyListener = new KeyListener()
1352 {
1353     public void keyPressed(KeyEvent keyEvent)
1354     {
1355         printIt("Pressed", keyEvent);
1356     }
1357     public void keyReleased(KeyEvent keyEvent)
1358     {
1359         printIt("Released", keyEvent);
1360         createProjectDialog4.readTextFields();
1361     }
1362     public void keyTyped(KeyEvent keyEvent)
1363     {
1364         printIt("Typed", keyEvent);
1365     }
1366     private void printIt(String title, KeyEvent keyEvent){}
1367 };
1368
1369 firstLevelTextField.addKeyListener(keyListener);
1370 distanceTextField.addKeyListener(keyListener);
1371
1372 for (int i=0; i<numberOfBregmaLevels; i++)
1373 {
1374     bregmaLevelLabel[i] = new JLabel("Bregma level " +(i+1)+":");
1375     bregmaLevelTextFields[i]=new JTextField(""+i,5);
1376     bregmaLevelTextFields[i].addKeyListener(keyListener);
1377     bregmaLevelResultLabel[i] = new JLabel(""+i);
1378 }
1379
1380 this.setUpWindow();
1381 }
1382
1383 public void setUpWindow()
1384 {
1385     this.getContentPane().removeAll();
1386     panel.removeAll();
1387     this.add(new JScrollPane(panel));
1388     this.setModalityType(ModalityType.APPLICATION_MODAL);
1389
1390     calculateButton=new JButton("Calculate levels");
1391     calculateButton.addActionListener(this);
1392     helpButton=new JButton("Help");
1393     helpButton.addActionListener(this);
1394     previousButton=new JButton("Previous");
1395     previousButton.addActionListener(this);
1396     nextButton=new JButton("Next");
1397     nextButton.addActionListener(this);
1398
1399     panel.setLayout(new GridBagLayout());

```

```

1400 GridBagConstraints c = new GridBagConstraints();
1401
1402 c.gridx = 1;
1403 c.gridy = 0;
1404 c.gridwidth = 1;
1405 c.insets = new Insets(30,5,20,5);
1406 c.anchor = GridBagConstraints.PAGE_START;
1407
1408 panel.add(firstInfo, c);
1409
1410 c.gridwidth = 1;
1411 c.insets = new Insets(5,5,5,5);
1412 c.gridx = 0;
1413 c.gridy=1;
1414 c.anchor = GridBagConstraints.FIRST_LINE_END;
1415 panel.add(firstLevelLabel, c);
1416
1417 c.gridx = 1;
1418 c.anchor = GridBagConstraints.PAGE_START;
1419 panel.add(firstLevelTextField, c);
1420
1421 c.gridx = 2;
1422 c.anchor = GridBagConstraints.FIRST_LINE_START;
1423 panel.add(firstLevelResultLabel, c);
1424
1425 c.insets = new Insets(5,5,5,5);
1426 c.gridx = 0;
1427 c.gridy=2;
1428 c.gridwidth = 1;
1429 c.anchor = GridBagConstraints.FIRST_LINE_END;
1430 panel.add(distanceLabel, c);
1431
1432 c.gridx = 1;
1433 c.anchor = GridBagConstraints.PAGE_START;
1434 panel.add(distanceTextField, c);
1435
1436 c.gridx = 2;
1437 c.anchor = GridBagConstraints.FIRST_LINE_START;
1438 panel.add(distanceResultLabel, c);
1439
1440 c.anchor = GridBagConstraints.PAGE_START;
1441 c.gridx = 1;
1442 c.gridy=3;
1443 c.gridwidth = 1;
1444 panel.add(calculateButton, c);
1445
1446 c.gridx = 1;
1447 c.gridy = 4;
1448 c.gridwidth = 1;
1449 c.insets = new Insets(30,5,5,5);
1450 panel.add(info, c);
1451
1452 c.gridwidth = 1;
1453 c.insets = new Insets(5,5,5,5);
1454 for (int i=0; i<numberOfBregmaLevels; i++)
1455 {

```

```

1456         c.gridy = i+5;
1457
1458         c.anchor = GridBagConstraints.FIRST_LINE_END;
1459         c.gridx= 0;
1460         panel.add(bregmaLevelLabel[i], c);
1461
1462         c.anchor = GridBagConstraints.PAGE_START;
1463         c.gridx= 1;
1464         panel.add(bregmaLevelTextFields[i], c);
1465
1466         c.anchor = GridBagConstraints.FIRST_LINE_START;
1467         c.gridx= 2;
1468         panel.add(bregmaLevelResultLabel[i], c);
1469     }
1470
1471     c.gridwidth=1;
1472     c.gridy = numberOfBregmaLevels+6;
1473     c.gridx= 1;
1474     c.anchor = GridBagConstraints.PAGE_START;
1475     c.insets = new Insets(30,5,5,5);
1476     panel.add(statusLabel, c);
1477
1478     c.weighty=1;
1479     c.anchor = GridBagConstraints.FIRST_LINE_END;
1480     c.gridx = 0;
1481     c.weightx=0.3333;
1482     c.gridy = numberOfBregmaLevels+7;
1483     c.gridwidth = 1;
1484     c.insets = new Insets(30,5,5,5);
1485     panel.add(previousButton, c);
1486
1487     c.anchor = GridBagConstraints.PAGE_START;
1488     c.gridx = 1;
1489     panel.add(helpButton, c);
1490
1491     c.anchor = GridBagConstraints.FIRST_LINE_START;
1492     c.gridx = 2;
1493     panel.add(nextButton, c);
1494
1495     this.readTextFields();
1496
1497     this.repaint();
1498 }
1499
1500 public void readTextFields()
1501 {
1502     calculateButton.setEnabled(true);
1503     result = firstLevelTextField.getText();
1504     try
1505     {
1506         firstLevel= Double.parseDouble(result);
1507         firstLevelResultLabel.setForeground(Color.green);
1508         firstLevelResultLabel.setText(result);
1509     }
1510     catch (NumberFormatException error)
1511     {

```

```

1512     firstLevelResultLabel.setForeground(Color.red);
1513     firstLevelResultLabel.setText("Not a number");
1514     calculateButton.setEnabled(false);
1515 }
1516
1517 result = distanceTextField.getText();
1518 try
1519 {
1520     distance= Double.parseDouble(result);
1521     distanceResultLabel.setForeground(Color.green);
1522     distanceResultLabel.setText(result);
1523     if (distance==0)
1524     {
1525         distanceResultLabel.setForeground(Color.red);
1526         distanceResultLabel.setText("Can't be 0");
1527         calculateButton.setEnabled(false);
1528     }
1529 }
1530 catch (NumberFormatException error)
1531 {
1532     distanceResultLabel.setForeground(Color.red);
1533     distanceResultLabel.setText("Not a number");
1534     calculateButton.setEnabled(false);
1535 }
1536
1537 allOK = true;
1538 for (int i=0; i<numberOfBregmaLevels; i++)
1539 {
1540     result = bregmaLevelTextFields[i].getText();
1541     try
1542     {
1543         levels[i] = Double.parseDouble(result);
1544         bregmaLevelResultLabel[i].setForeground(Color.green);
1545         bregmaLevelResultLabel[i].setText(result);
1546     }
1547     catch (NumberFormatException error)
1548     {
1549         bregmaLevelResultLabel[i].setForeground(Color.red);
1550         bregmaLevelResultLabel[i].setText("Not a number");
1551         allOK = false;
1552     }
1553 }
1554
1555
1556 boolean allIncreasing=true;
1557 boolean allDecreasing=true;
1558 if (allOK)
1559 {
1560     for(int i=0;i<numberOfBregmaLevels-1;i++)
1561     {
1562         if (levels[i]>=levels[i+1]) allIncreasing=false;
1563         if (levels[i]<=levels[i+1]) allDecreasing=false;
1564     }
1565 }
1566
1567 if (!allOK)

```

```

1568     {
1569         statusLabel.setForeground(Color.red);
1570         statusLabel.setText("A non-number entered");
1571         nextButton.setEnabled(false);
1572     }
1573     else if (!allIncreasing && !allDecreasing)
1574     {
1575         statusLabel.setForeground(Color.red);
1576         statusLabel.setText("Values must continuously increase or decrease");
1577         nextButton.setEnabled(false);
1578     }
1579     else
1580     {
1581         statusLabel.setForeground(Color.green);
1582         statusLabel.setText("Ok to proceed");
1583         nextButton.setEnabled(true);
1584     }
1585 }
1586
1587 public void actionPerformed(ActionEvent e)
1588 {
1589     if (e.getSource()==calculateButton)
1590     {
1591         firstLevel=0;
1592         distance=1;
1593         double newLevel=0;
1594         double multiplier=1;
1595         double tempDouble;
1596
1597         try
1598         {
1599             result = firstLevelTextField.getText();
1600             firstLevel = Double.parseDouble(result);
1601             result = distanceTextField.getText();
1602             distance = Double.parseDouble(result);
1603         }
1604         catch (NumberFormatException error) {}
1605
1606         for (int i=0; i<numberOfBregmaLevels; i++)
1607         {
1608             multiplier = i;
1609             newLevel = firstLevel + multiplier*distance;
1610             tempDouble= (double) (Math.round(newLevel*1000))/1000;
1611             bregmaLevelTextFields[i].setText(""+tempDouble);
1612         }
1613         this.readTextFields();
1614     }
1615
1616     if (e.getSource()==helpButton)
1617     {
1618         String helpText = myMethods.getHelpTextBregmaLevels();
1619         helpDialog.newText(helpText);
1620     }
1621
1622     if (e.getSource()==previousButton)
1623     {

```

```

1624         this.setVisible(false);
1625         createProjectDialog3.setVisible(true);
1626     }
1627
1628     if (e.getSource()==nextButton)
1629     {
1630         this.readTextFields();
1631
1632         if (allOK)
1633         {
1634             this.setVisible(false);
1635             createProjectDialog5.setVisible(true);
1636         }
1637     }
1638 }
1639 }
1640
1641 private class CreateProjectDialog5 extends JDialog implements ActionListener
1642 {
1643     //Lets the user select the number of User defined areas, the name of
1644     //each area and wheter it should count tissue, background or both
1645     JLabel firstInfo = new JLabel("Enter Number Of User Defined Areas");
1646
1647     int numberOfUDA=3;
1648     JTextField[] udaTextFields;
1649
1650     String[] dialogUdaNames;
1651     boolean[] dialogIncludeBackground;
1652     boolean[] dialogIncludeTissue;
1653
1654     JTextField numberOfUDATextField = new JTextField("3", 5);
1655
1656     JButton helpButton;
1657     JButton increaseButton;
1658     JButton decreaseButton;
1659
1660     JRadioButton[] tissueRadioButton;
1661     JRadioButton[] backgroundRadioButton;
1662     JRadioButton[] bothRadioButton;
1663     ButtonGroup[] buttonGroup;
1664
1665     JButton createProjectButton;
1666     JButton previousButton;
1667
1668     JPanel panel = new JPanel();
1669
1670     public CreateProjectDialog5()
1671     {
1672         this.setTitle("Enter user defined areas");
1673         int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
1674         int screenHeight= (int) Toolkit.getDefaultToolkit().getScreenSize().getHeight();
1675         this.setLocation(screenWidth*10/100,screenHeight*10/100);
1676         this.setSize(screenWidth*35/100,screenHeight*80/100);
1677
1678         numberOfUDATextField.addActionListener(this);
1679

```

```

1680     udaTextFields = new JTextField[numberOfUDA];
1681     tissueRadioButton = new JRadioButton(numberOfUDA);
1682     backgroundRadioButton = new JRadioButton(numberOfUDA);
1683     bothRadioButton = new JRadioButton(numberOfUDA);
1684     buttonGroup = new ButtonGroup(numberOfUDA);
1685
1686     for (int i=0; i<numberOfUDA; i++)
1687     {
1688         udaTextFields[i]=new JTextField("Area "+(i+1),20);
1689     }
1690
1691     this.setUpWindow();
1692 }
1693
1694 public void setUpWindow()
1695 {
1696     this.getContentPane().removeAll();
1697     panel.removeAll();
1698     this.add(new JScrollPane(panel));
1699     this.setModalityType(ModalityType.APPLICATION_MODAL);
1700
1701     helpButton=new JButton("Help");
1702     helpButton.addActionListener(this);
1703     decreaseButton=new JButton("Decrease");
1704     decreaseButton.addActionListener(this);
1705     increaseButton=new JButton("Increase");
1706     increaseButton.addActionListener(this);
1707     createProjectButton=new JButton("Create project");
1708     createProjectButton.addActionListener(this);
1709     previousButton=new JButton("Previous");
1710     previousButton.addActionListener(this);
1711
1712     panel.setLayout(new GridBagLayout());
1713     GridBagConstraints c = new GridBagConstraints();
1714
1715     c.insets = new Insets(30,5,0,5);
1716     c.anchor = GridBagConstraints.PAGE_START;
1717     c.gridwidth = 1;
1718     c.gridx = 1;
1719     c.gridy = 0;
1720     panel.add(firstInfo, c);
1721
1722     c.insets.top=10;
1723     c.gridx = 0;
1724     c.gridy = 1;
1725     panel.add(decreaseButton, c);
1726
1727     numberOfUDATextField.setText(""+numberOfUDA);
1728     c.gridx = 1;
1729     c.gridy = 1;
1730     c.gridwidth = 1;
1731     panel.add(numberOfUDATextField, c);
1732
1733     c.gridx = 2;
1734     c.gridy = 1;
1735     c.gridwidth = 1;

```

```

1736
1737     panel.add(increaseButton, c);
1738
1739     c.insets.top=40;
1740
1741     for (int i=0; i<numberOfUDA; i++)
1742     {
1743         c.gridx = 0;
1744         c.gridy = i*2+2;
1745         c.gridwidth = 3;
1746         panel.add(udaTextFields[i], c);
1747
1748         tissueRadioButton[i]= new JRadioButton("Tissue");
1749         backgroundRadioButton[i]= new JRadioButton("Background");
1750         bothRadioButton[i]= new JRadioButton("Both",true);
1751
1752         buttonGroup[i] = new ButtonGroup();
1753
1754         buttonGroup[i].add(tissueRadioButton[i]);
1755         buttonGroup[i].add(backgroundRadioButton[i]);
1756         buttonGroup[i].add(bothRadioButton[i]);
1757
1758         c.gridx = 0;
1759         c.gridy = i*2+3;
1760         c.gridwidth = 1;
1761         c.insets.top=10;
1762         panel.add(tissueRadioButton[i], c);
1763         c.gridx = 1;
1764         panel.add(backgroundRadioButton[i], c);
1765         c.gridx = 2;
1766         panel.add(bothRadioButton[i], c);
1767         c.insets.top=40;
1768     }
1769
1770     c.weighty=1;
1771     c.gridx = 2;
1772     c.gridy = numberOfUDA*2+2;
1773     c.gridwidth = 1;
1774     c.insets.top=40;
1775     c.insets.bottom=40;
1776
1777     c.gridx = 0;
1778     panel.add(previousButton, c);
1779
1780     c.gridx=1;
1781     panel.add(helpButton, c);
1782
1783     c.gridx=2;
1784     panel.add(createProjectButton, c);
1785 }
1786
1787 public void actionPerformed(ActionEvent e)
1788 {
1789     if (e.getSource()==helpButton)
1790     {
1791         String helpText = myMethods.getHelpTextUDA();

```



```

1792     helpDialog.newText(helpText);
1793 }
1794
1795 if (e.getSource()==numberOfUDATextField)
1796 {
1797     try
1798     {
1799         numberOfUDA = Integer.parseInt(numberOfUDATextField.getText());
1800         this.setVisible(false);
1801
1802         udaTextFields = new JTextField[numberOfUDA];
1803         tissueRadioButton = new JRadioButton[numberOfUDA];
1804         backgroundRadioButton = new JRadioButton[numberOfUDA];
1805         bothRadioButton = new JRadioButton[numberOfUDA];
1806         buttonGroup = new ButtonGroup[numberOfUDA];
1807
1808         for (int i=0; i<numberOfUDA; i++)
1809         {
1810             udaTextFields[i]=new JTextField("Area "+(i+1),20);
1811         }
1812
1813         this.setUpWindow();
1814         this.setVisible(true);
1815     } catch (Exception error) {}
1816 }
1817
1818 if (e.getSource()==decreaseButton)
1819 {
1820     if (numberOfUDA>0) numberOfUDA--;
1821     this.setVisible(false);
1822     numberOfUDATextField.setText(""+numberOfUDA);
1823
1824     udaTextFields = new JTextField[numberOfUDA];
1825     tissueRadioButton = new JRadioButton[numberOfUDA];
1826     backgroundRadioButton = new JRadioButton[numberOfUDA];
1827     bothRadioButton = new JRadioButton[numberOfUDA];
1828     buttonGroup = new ButtonGroup[numberOfUDA];
1829
1830     for (int i=0; i<numberOfUDA; i++)
1831     {
1832         udaTextFields[i]=new JTextField("Area "+(i+1),20);
1833     }
1834
1835     this.setUpWindow();
1836     this.setVisible(true);
1837 }
1838
1839 if (e.getSource()==increaseButton)
1840 {
1841     numberOfUDA++;
1842
1843     this.setVisible(false);
1844     numberOfUDATextField.setText(""+numberOfUDA);
1845
1846     udaTextFields = new JTextField[numberOfUDA];
1847     tissueRadioButton = new JRadioButton[numberOfUDA];

```

```

1848 backgroundRadioButton = new JRadioButton(numberOfUDA);
1849 bothRadioButton = new JRadioButton(numberOfUDA);
1850 buttonGroup = new ButtonGroup(numberOfUDA);
1851
1852 for (int i=0; i<numberOfUDA; i++)
1853 {
1854     udaTextFields[i]=new JTextField("Area "+(i+1),20);
1855 }
1856
1857 this.setUpWindow();
1858 this.setVisible(true);
1859 }
1860
1861 if (e.getSource()==createProjectButton)
1862 {
1863
1864     dialogUdaNames = new String(numberOfUDA);
1865     dialogIncludeBackground = new boolean(numberOfUDA);
1866     dialogIncludeTissue = new boolean(numberOfUDA);
1867
1868     for (int i=0; i<numberOfUDA; i++)
1869     {
1870         dialogUdaNames[i] = udaTextFields[i].getText();
1871
1872         if (tissueRadioButton[i].isSelected())
1873         {
1874             dialogIncludeTissue[i] = true;
1875             dialogIncludeBackground[i] = false;
1876         }
1877         if (backgroundRadioButton[i].isSelected())
1878         {
1879             dialogIncludeTissue[i] = false;
1880             dialogIncludeBackground[i] = true;
1881         }
1882         if (bothRadioButton[i].isSelected())
1883         {
1884             dialogIncludeTissue[i] = true;
1885             dialogIncludeBackground[i] = true;
1886         }
1887     }
1888
1889     try
1890     {
1891         this.setVisible(false);
1892         myProject.createNewProject() ;
1893     }catch (Exception error) {}
1894
1895     this.setVisible(false);
1896 }
1897
1898 if (e.getSource()==previousButton)
1899 {
1900     this.setVisible(false);
1901     createProjectDialog4.setVisible(true);
1902 }
1903 }

```

```

1904 }
1905
1906 private class Project
1907 {
1908     public void createNewProject()
1909     {
1910         progressMonitor = new ProgressMonitor(null,
1911             "Building Project",
1912             "Sections analyzed: 0", 0, createProjectDialog1.numberOfSections);
1913
1914         progressMonitor.setMillisToDecideToPopup(0);
1915         progressMonitor.setMillisToPopup(0);
1916
1917         BuildProject buildProject = new BuildProject();
1918         buildProject.execute();
1919     }
1920
1921     public class BuildProject extends SwingWorker<Void, Void>
1922     {
1923         //Uses the informaiton from the createProjectDailog frames to build a new project.
1924         @Override
1925         public Void doInBackground()
1926         {
1927             frame.setVisible(false);
1928             int sectionsAnalyzed=0;
1929
1930             myProjectData = new ProjectData(createProjectDialog1.numberOfSubjects,
1931                 createProjectDialog1.highestNumberOfSectionsInSubject,
1932                 createProjectDialog5.numberOfUDA,
1933                 createProjectDialog1.numberOfSections);
1934
1935             treePanel = new TreePanel(createProjectDialog1.numberOfSubjects,
1936                 createProjectDialog1.highestNumberOfSectionsInSubject,
1937                 createProjectDialog5.numberOfUDA);
1938
1939             myProjectData.projectLocation = createProjectDialog2.saveFileLocation.getPath();
1940             myProjectData.imageWidth=createProjectDialog1.detectedImageWidth;
1941             myProjectData.imageHeight=createProjectDialog1.detectedImageHeight;
1942
1943             for (int i=0; i<myProjectData.numberOfUDA; i++)
1944             {
1945                 myProjectData.UDANames[i]=createProjectDialog5.dialogUdaNames[i];
1946                 myProjectData.includeBackground[i]=createProjectDialog5.dialogIncludeBackground[i];
1947                 myProjectData.includeTissue[i]=createProjectDialog5.dialogIncludeTissue[i];
1948             }
1949
1950             //Saves a copy of the conversion image in the save folder
1951             try
1952             {
1953                 String conversionFileName = new String(myProjectData.projectLocation+"/ConversionImage");
1954                 File conversionFile = new File(conversionFileName);
1955
1956                 ImageIO.write(ImageIO.read(createProjectDialog3.conversionFileLocation),"JPG",conversionFile);
1957             }catch (IOException event) { }
1958
1959             //Adds subjects to the tree structure

```

```

1959     for (int i = 0; i < createProjectDialog1.numberOfSubjects; i++)
1960     {
1961         treePanel.addSubjectToTree(i, createProjectDialog1.subjectNames[i]);
1962         myProjectData.subjectNameList[i]=createProjectDialog1.subjectNames[i];
1963     }
1964
1965     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
1966
1967     for (int subjectCounter=0; subjectCounter<createProjectDialog1.numberOfSubjects;
subjectCounter++)
1968     {
1969         for (int sectionCounter=0;
sectionCounter<createProjectDialog1.highestNumberOfSectionsInSubject; sectionCounter++)
1970         {
1971             if (createProjectDialog1.sectionExists[subjectCounter][sectionCounter])
1972             {
1973                 //Saves the image file in the save folder
1974                 String originalImageFileName = myMethods.getOriginalImageFileName(subjectCounter,
sectionCounter);
1975                 File saveImageFile= new File(originalImageFileName);
1976                 try
1977                 {
1978                     originalImage=ImageIO.read(createProjectDialog1.sectionFiles[subjectCounter][sectionCounter]);
1979                     ImageIO.write(originalImage, "JPG", saveImageFile);
1980                 } catch (IOException event) { }
1981
1982                 //Analyses the section and adds it to the tree structure
1983                 currentSectionData= new SectionData(myProjectData.imageWidth,
myProjectData.imageHeight);
1984
1985                 currentSectionData.positionInProject= sectionsAnalyzed;
1986
1987                 currentSectionData.lrDividePixels = new
boolean[originalImage.getWidth()][originalImage.getHeight()];
1988                 for (int i=0;i<originalImage.getWidth();i++)
1989                     for (int j=0;j<originalImage.getHeight();j++)
1990                         currentSectionData.lrDividePixels[i][j]=false;
1991
1992                 currentSectionData.UDALinePixels = new
boolean[myProjectData.numberOfUDA][originalImage.getWidth()][originalImage.getHeight()];
1993                 currentSectionData.UDAcurentPoint = new Point[myProjectData.numberOfUDA];
1994                 currentSectionData.UDApreeviousPoint = new Point[myProjectData.numberOfUDA];
1995                 currentSectionData.UDAhaveStartPoint = new boolean[myProjectData.numberOfUDA];
1996
1997                 for (int UDACounter=0; UDACounter<myProjectData.numberOfUDA;UDACounter++)
1998                 {
1999
2000                     for (int i=0;i<originalImage.getWidth();i++)
2001                         for (int j=0;j<originalImage.getHeight();j++)
2002                             currentSectionData.UDALinePixels[UDACounter][i][j]=false;
2003
2004                     currentSectionData.UDAcurentPoint[UDACounter]=new Point(0,0);
2005                     currentSectionData.UDApreeviousPoint[UDACounter]=new Point(0,0);
2006                     currentSectionData.UDAhaveStartPoint[UDACounter]=false;
2007                 }

```

```

2008
2009         myMethods.saveSectionData(subjectCounter, sectionCounter, currentSectionData);
2010
2011         treePanel.addSectionToTree(subjectCounter, sectionCounter);
2012         myProjectData.sectionExists[subjectCounter][sectionCounter] = true;
2013         myProjectData.sectionName[subjectCounter][sectionCounter] =
createProjectDialog1.sectionNames[subjectCounter][sectionCounter];
2014         myProjectData.bregmaLevels[subjectCounter][sectionCounter] =
createProjectDialog4.levels[sectionCounter];
2015
2016         myMethods.analyzeSection(subjectCounter, sectionCounter);
2017         myMethods.updateSectionResultsObject(subjectCounter, sectionCounter);
2018
2019         sectionsAnalyzed++;
2020         progressMonitor.setNote("Sections analyzed: "+sectionsAnalyzed);
2021         progressMonitor.setProgress(sectionsAnalyzed);
2022     }
2023     if (progressMonitor.isCanceled())
2024     {
2025         break;
2026     }
2027 }
2028 myMethods.analyzeSubject(subjectCounter);
2029
2030 if (progressMonitor.isCanceled())
2031 {
2032     errorDialog.newText(myMethods.getErrorTextCreateProjectInterrupted());
2033     errorDialog.setVisible(true);
2034     break;
2035 }
2036 }
2037
2038 //Sets the zoomFactor to make one image fill the panel
2039 Dimension panelSize = middleScrollPane.getSize();
2040 double widthFactor = 100*panelSize.width/originalImage.getWidth();
2041 double heightFactor = 100*panelSize.height/originalImage.getHeight();
2042 if (widthFactor<heightFactor) myProjectData.zoomFactor = (int) widthFactor;
2043 else myProjectData.zoomFactor = (int) heightFactor;
2044 if (myProjectData.zoomFactor<1) myProjectData.zoomFactor=1;
2045
2046 myMethods.saveProjectData();
2047
2048 leftScrollPane.getViewport().add(treePanel);
2049
2050 treePanel.expandTree();
2051
2052 treePanel.tree.setSelectionRow(4); //Displays the conversion panel
2053
2054 upperScrollPane.requestFocus();
2055
2056 middleScrollPane.repaint();
2057 leftScrollPane.repaint();
2058
2059 frame.setVisible(true);
2060
2061 return null;

```

```

2062     }
2063     @Override
2064     public void done() {}
2065 }
2066
2067 //Triggered when the user opens an existing project
2068 public void refreshWhenOpened()
2069 {
2070     treePanel = new TreePanel(myProjectData.numberOfSubjects,
2071                               myProjectData.numberOfLevels,
2072                               myProjectData.numberOfUDA);
2073
2074     for (int i=0;i<myProjectData.numberOfSubjects;i++)
2075     {
2076         treePanel.addSubjectToTree(i, myProjectData.subjectNameList[i]);
2077     }
2078
2079     for (int i=0;i<myProjectData.numberOfSubjects;i++)
2080         for (int j=0;j<myProjectData.numberOfLevels;j++)
2081         {
2082             if (myProjectData.sectionExists[i][j])
2083             {
2084                 treePanel.addSectionToTree(i,j );
2085             }
2086         }
2087     leftScrollPane.getViewport().add(treePanel);
2088
2089     treePanel.expandTree();
2090
2091     treePanel.tree.setSelectionRow(4);
2092
2093     upperScrollPane.requestFocus();
2094 }
2095 }
2096
2097 private class TreePanel extends JPanel implements TreeSelectionListener
2098 {
2099     //The tree structure is displayed on the left scroll pane.
2100     //Whenever the user makes a new selection in the tree structure
2101     //a whenLeftInTree is executed for the previous selection and then
2102     //a whenClickedInTree is executed for the new selection.
2103     JTree tree;
2104
2105     DefaultMutableTreeNode currentSelection;
2106     DefaultMutableTreeNode projectLeaf = new DefaultMutableTreeNode("Project");
2107
2108     DefaultMutableTreeNode[] subjectLeafs;
2109     DefaultMutableTreeNode[][] sectionLeafs;
2110     DefaultMutableTreeNode[][] adjustImageLeafs;
2111     DefaultMutableTreeNode[][] setLeftRightBorderLeafs;
2112     DefaultMutableTreeNode[][] udaLeafs;
2113
2114     DefaultMutableTreeNode resultLeaf = new DefaultMutableTreeNode("Results");
2115     DefaultMutableTreeNode subjectResultLeaf = new DefaultMutableTreeNode("Subject Results");
2116     DefaultMutableTreeNode conversionLeaf = new DefaultMutableTreeNode("Conversion");
2117     DefaultMutableTreeNode tissueDetectionLeaf = new DefaultMutableTreeNode("TissueDetection");

```

```

2118
2119 public TreePanel(int newNumberOfSubjects, int newNumberOfLevels, int newNumeOfUda)
2120 {
2121     tree = new JTree(projectLeaf);
2122
2123     subjectLeafs = new DefaultMutableTreeNode[newNumberOfSubjects];
2124     sectionLeafs = new DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2125     adjustImageLeafs = new DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2126     setLeftRightBorderLeafs = new
DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2127
2128     udaLeafs = new DefaultMutableTreeNode[newNumberOfSubjects][newNumberOfLevels];
2129
2130     projectLeaf.add(resultLeaf);
2131     resultLeaf.setUserObject(sectionResults_middlePanel);
2132     projectLeaf.add(subjectResultLeaf);
2133     subjectResultLeaf.setUserObject(subjectResults_middlePanel);
2134     projectLeaf.add(tissueDetectionLeaf);
2135     tissueDetectionLeaf.setUserObject(tissueDetection_middlePanel);
2136     projectLeaf.add(conversionLeaf);
2137     conversionLeaf.setUserObject(conversion_middlePanel);
2138
2139     this.add(tree);
2140     tree.addTreeSelectionListener(this);
2141 }
2142
2143 // This is triggered when the user clicks the TreePanel or press the keys A or D
2144 public void valueChanged(TreeSelectionEvent event)
2145 {
2146     //Determines the previous selection and triggers a whenLeftInTree
2147     try{
2148         conversion_middlePanel = (Conversion_MiddlePanel) currentSelection.getUserObject();
2149         conversion_middlePanel.whenLeftInTree();
2150     }catch (Exception e) {}
2151
2152     try{
2153         SectionTreeObject currentSection = (SectionTreeObject) currentSelection.getUserObject();
2154         currentSection.whenLeftInTree();
2155     }catch (Exception e) {}
2156
2157     try
2158     {
2159         AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject)
currentSelection.getUserObject();
2160         currentAdjustImage.whenLeftInTree();
2161     }catch (Exception e) {}
2162
2163     try
2164     {
2165         SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
currentSelection.getUserObject();
2166         currentSetLeftRightBorder.whenLeftInTree();
2167     }catch (Exception e) {}
2168
2169     try{
2170         UdaTreeObject currentUdaObject = (UdaTreeObject) currentSelection.getUserObject();

```

```

2171         currentUdaObject.whenLeftInTree();
2172     }catch (Exception e) {}
2173
2174     try{
2175         subjectResults_middlePanel = (SubjectResults_MiddlePanel) currentSelection.getUserObject();
2176         subjectResults_middlePanel.whenLeftInTree();
2177     }catch (Exception e) {}
2178
2179     try{
2180         sectionResults_middlePanel = (SectionResults_MiddlePanel) currentSelection.getUserObject();
2181         sectionResults_middlePanel.whenLeftInTree();
2182     }catch (Exception e) {}
2183
2184     //Sets the default cursor in case the cursor was changed in the adjustImagePanel
2185     middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
2186
2187     //Determines the new selection and triggers a whenClickedInTree.
2188     currentSelection = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
2189
2190     try{
2191         SubjectTreeObject currentSubject = (SubjectTreeObject) currentSelection.getUserObject();
2192         currentSubject.whenClickedInTree();
2193     }catch (Exception e) {}
2194
2195     try{
2196         SectionTreeObject currentSection = (SectionTreeObject) currentSelection.getUserObject();
2197         currentSection.whenClickedInTree();
2198     }catch (Exception e) {}
2199
2200     try{
2201         AdjustImageTreeObject currentAdjustImage = (AdjustImageTreeObject)
currentSelection.getUserObject();
2202         currentAdjustImage.whenClickedInTree();
2203     }catch (Exception e) {}
2204
2205     try{
2206         SetLeftRightBorderTreeObject currentSetLeftRightBorder = (SetLeftRightBorderTreeObject)
currentSelection.getUserObject();
2207         currentSetLeftRightBorder.whenClickedInTree();
2208     }catch (Exception e) {}
2209
2210     try{
2211         UdaTreeObject currentUdaObject = (UdaTreeObject) currentSelection.getUserObject();
2212         currentUdaObject.whenClickedInTree();
2213     }catch (Exception e) {}
2214
2215     try{
2216         tissueDetection_middlePanel = (TissueDetection_MiddlePanel) currentSelection.getUserObject();
2217         tissueDetection_middlePanel.whenClickedInTree();
2218     }catch (Exception e) {}
2219
2220     try{
2221         conversion_middlePanel = (Conversion_MiddlePanel) currentSelection.getUserObject();
2222         conversion_middlePanel.whenClickedInTree();
2223     }catch (Exception e) {}
2224

```



```

2225     try{
2226         subjectResults_middlePanel = (SubjectResults_MiddlePanel) currentSelection.getUserObject();
2227         subjectResults_middlePanel.whenClickedInTree();
2228     }catch (Exception e) { }
2229
2230     try{
2231         sectionResults_middlePanel = (SectionResults_MiddlePanel) currentSelection.getUserObject();
2232         sectionResults_middlePanel.whenClickedInTree();
2233     }catch (Exception e) { }
2234
2235     //Collapses all rows except the selected one
2236     tree.removeTreeSelectionListener(this);
2237     TreePath selectionPath = tree.getSelectionPath();
2238     int rowCount= tree.getRowCount();
2239     for (int i=rowCount; i>-1; i--)
2240     {
2241         try
2242         {
2243             tree.collapseRow(i);
2244         }catch(Exception error){ }
2245     }
2246     tree.makeVisible(selectionPath);
2247     tree.setSelectionPath(selectionPath);
2248     tree.addTreeSelectionListener(this);
2249
2250     upperScrollPane.requestFocus();
2251 }
2252
2253 public void addSubjectToTree (int i, String subject)
2254 {
2255     subjectLeafs[i] = new DefaultMutableTreeNode();
2256     subjectLeafs[i].setUserObject(new SubjectTreeObject(i));
2257     projectLeaf.add(subjectLeafs[i]);
2258 }
2259
2260 public void addSectionToTree (int subjectNumber, int sectionNumber)
2261 {
2262     sectionLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2263     sectionLeafs[subjectNumber][sectionNumber].setUserObject(new SectionTreeObject(subjectNumber,
sectionNumber));
2264     subjectLeafs[subjectNumber].add(sectionLeafs[subjectNumber][sectionNumber]);
2265
2266     adjustImageLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2267     adjustImageLeafs[subjectNumber][sectionNumber].setUserObject(new
AdjustImageTreeObject(subjectNumber, sectionNumber));
2268     sectionLeafs[subjectNumber][sectionNumber].add(adjustImageLeafs[subjectNumber][sectionNumber]);
2269
2270     setLeftRightBorderLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2271     setLeftRightBorderLeafs[subjectNumber][sectionNumber].setUserObject(new
SetLeftRightBorderTreeObject(subjectNumber, sectionNumber));
2272     sectionLeafs[subjectNumber][sectionNumber].add(setLeftRightBorderLeafs[subjectNumber][sectionNumber]);
2273
2274     for (int i=0; i<myProjectData.numberOfUDA;i++)
2275     {

```

```

2276         udaLeafs[subjectNumber][sectionNumber] = new DefaultMutableTreeNode();
2277         udaLeafs[subjectNumber][sectionNumber].setUserObject(new UdaTreeObject(subjectNumber,
sectionNumber, i));
2278         sectionLeafs[subjectNumber][sectionNumber].add(udaLeafs[subjectNumber][sectionNumber]);
2279     }
2280 }
2281 }
2282 public void expandTree()
2283 {
2284     tree.expandRow(0);
2285 }
2286 }
2287
2288 //The different TreeObjects are attached to the tree structure.
2289 //They rely info about which section was activated in the tree
2290 //to the relevant panel object.
2291 private class SubjectTreeObject
2292 {
2293     int subjectNumber;
2294
2295     public SubjectTreeObject (int subject)
2296     {
2297         subjectNumber=subject;
2298     }
2299
2300     public void whenClickedInTree()
2301     {
2302         subject_middlePanel.whenClickedInTree(subjectNumber);
2303     }
2304
2305     @Override
2306     public String toString()
2307     {
2308         return myProjectData.subjectNameList[subjectNumber];
2309     }
2310 }
2311
2312 private class SectionTreeObject
2313 {
2314     int subjectNumber;
2315     int sectionNumber;
2316
2317     public SectionTreeObject (int subject, int section)
2318     {
2319         subjectNumber=subject;
2320         sectionNumber=section;
2321     }
2322
2323     public void whenClickedInTree()
2324     {
2325         section_middlePanel.whenClickedInTree(subjectNumber, sectionNumber);
2326     }
2327
2328     public void whenLeftInTree()
2329     {
2330         section_middlePanel.whenLeftInTree(subjectNumber, sectionNumber);

```

```

2331     }
2332
2333     @Override
2334     public String toString()
2335     {
2336         return myProjectData.sectionName[subjectNumber][sectionNumber];
2337     }
2338 }
2339
2340 public class AdjustImageTreeObject
2341 {
2342     int subjectNumber;
2343     int sectionNumber;
2344
2345     public AdjustImageTreeObject(int initSubjectNumber, int initSectionNumber)
2346     {
2347         subjectNumber=initSubjectNumber;
2348         sectionNumber=initSectionNumber;
2349     }
2350
2351     public void whenClickedInTree()
2352     {
2353         adjustImage_middlePanel.whenClickedInTree(subjectNumber, sectionNumber);
2354     }
2355     public void whenLeftInTree()
2356     {
2357         adjustImage_middlePanel.whenLeftInTree();
2358     }
2359
2360     @Override
2361     public String toString()
2362     {
2363         String temp;
2364         temp="Adjust image";
2365         return temp;
2366     }
2367 }
2368
2369 public class SetLeftRightBorderTreeObject
2370 {
2371     int subjectNumber;
2372     int sectionNumber;
2373
2374     public SetLeftRightBorderTreeObject (int initSubjectNumber, int initSectionNumber)
2375     {
2376         subjectNumber=initSubjectNumber;
2377         sectionNumber=initSectionNumber;
2378     }
2379
2380     public void whenClickedInTree()
2381     {
2382         lr_divideMiddlePanel.whenClickedInTree(subjectNumber, sectionNumber);
2383     }
2384     public void whenLeftInTree()
2385     {
2386         lr_divideMiddlePanel.whenLeftInTree();

```

```

2387     }
2388
2389     @Override
2390     public String toString()
2391     {
2392         String temp;
2393         temp="Set Left-Right Border";
2394         return temp;
2395     }
2396 }
2397
2398 public class UdaTreeObject
2399 {
2400     int subjectNumber;
2401     int sectionNumber;
2402     int UDANumber;
2403
2404     public UdaTreeObject(int initSubject, int initSection, int initUDANumber)
2405     {
2406         subjectNumber=initSubject;
2407         sectionNumber=initSection;
2408         UDANumber=initUDANumber;
2409     }
2410
2411     @Override
2412     public String toString()
2413     {
2414         String temp;
2415         temp=myProjectData.UDANames[UDANumber];
2416         return temp;
2417     }
2418
2419     public void whenClickedInTree()
2420     {
2421         uda_middlePanel.whenClickedInTree(subjectNumber, sectionNumber, UDANumber);
2422     }
2423
2424     public void whenLeftInTree()
2425     {
2426         uda_middlePanel.whenLeftInTree();
2427     }
2428 }
2429
2430 private class SectionResults_MiddlePanel extends JPanel
2431 {
2432     //Creates the table with the results for all sections
2433     JTable table = new JTable();
2434     String[] columnNames;
2435     String resultsForClipboard;
2436
2437     public SectionResults_MiddlePanel()
2438     {
2439         table = new JTable();
2440     }
2441
2442     public void updateTable()

```

```

2443 {
2444     sectionResults_middlePanel.remove(table);
2445
2446     int numberOfUDA= myProjectData.numberOfUDA;
2447
2448     columnNames= new String[15+2*numberOfUDA];
2449
2450     columnNames[0]= "Subject";
2451     columnNames[1]= "Section";
2452     columnNames[2]= "Bregma";
2453     columnNames[3]= "Tissue Pixels";
2454     columnNames[4]= "Tissue mm2";
2455     columnNames[5]= "Left Tissue Pixels";
2456     columnNames[6]= "Left Tissue mm2";
2457     columnNames[7]= "Right Tissue Pixels";
2458     columnNames[8]= "Right Tissue mm2";
2459     columnNames[9]= "Ventricle pixels";
2460     columnNames[10]= "Ventricle mm2";
2461     columnNames[11]= "Left Ventricle Pixels";
2462     columnNames[12]= "Left Ventricle mm2";
2463     columnNames[13]= "Right Ventricle Pixels";
2464     columnNames[14]= "Right Ventricle mm2";
2465
2466     for (int i=0; i<numberOfUDA;i++)
2467     {
2468
2469         columnNames[15+2*i]= myProjectData.UDAnames[i] + " Pixels";
2470         columnNames[16+2*i]= myProjectData.UDAnames[i] + " mm2";
2471     }
2472     this.removeAll();
2473
2474     table = new JTable(myProjectData.roundedSectionResultsObject, columnNames);
2475     table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
2476
2477     TableColumn column = null;
2478     for (int i=0; i<15+2*numberOfUDA ; i++)
2479     {
2480         column = table.getColumnModel().getColumn(i);
2481         column.setPreferredWidth(150);
2482     }
2483
2484     JScrollPane scrollPane = new JScrollPane(table);
2485
2486     firstSplitPane.setLeftComponent(scrollPane);
2487     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
2488     firstSplitPane.setDividerLocation(screenWidth*14/20);
2489 }
2490
2491 public void createClipboardString()
2492 {
2493     String string = new String("");
2494     String temp = new String("");
2495     int numberOfUDA= myProjectData.numberOfUDA;
2496
2497     for (int i=0;i<15+2*numberOfUDA;i++)
2498     {

```

```

2499     try
2500     {
2501         temp=columnNames[i];
2502         string= string + temp;
2503         if (i!=15+2*numberOfUDA-1)
2504             string = string + '\t';
2505
2506     }catch(Exception e){ }
2507 }
2508
2509 for (int i=0;i<myProjectData.totalNumberOfSections;i++)
2510 {
2511     string=string+'\n';
2512     for(int j=0;j<15+2*numberOfUDA;j++)
2513     {
2514         try
2515         {
2516             temp=myProjectData.sectionResultsObject[i][j].toString();
2517
2518             string= string + temp;
2519             if (j!=15+2*numberOfUDA-1)
2520                 string = string + '\t';
2521         }catch(Exception e){ }
2522     }
2523 }
2524 resultsForClipboard = string;
2525 }
2526
2527 public void copyToClipboard()
2528 {
2529     this.createClipboardString();
2530
2531     StringSelection stringToClipboard = new StringSelection(resultsForClipboard);
2532     Toolkit toolkit = Toolkit.getDefaultToolkit();
2533     Clipboard cp = toolkit.getSystemClipboard();
2534     cp.setContents(stringToClipboard, null);
2535 }
2536
2537 @Override
2538 public String toString()
2539 {
2540     return new String("Section Results");
2541 }
2542 public void whenClickedInTree()
2543 {
2544     myMethods.setWaitCursor();
2545
2546     this.updateTable();
2547     middleScrollPane.getViewPort().add(sectionResults_middlePanel);
2548     rightScrollPane.getViewPort().add(sectionResults_rightSidePanel);
2549
2550     if(myProjectData.conversionIsUpdated)
2551     {
2552         sectionResults_rightSidePanel.conversionWarning.setForeground(Color.green);
2553         sectionResults_rightSidePanel.conversionWarning.setText("Conversion updated");
2554     }

```

```

2555     else
2556     {
2557         sectionResults_rightSidePanel.conversionWarning.setForeground(Color.red);
2558         sectionResults_rightSidePanel.conversionWarning.setText("Conversion not updated");
2559     }
2560
2561     if(myProjectData.allLevelsContinuous)
2562     {
2563         sectionResults_rightSidePanel.continousLevelsWarning.setForeground(Color.green);
2564         sectionResults_rightSidePanel.continousLevelsWarning.setText("All levels continous");
2565     }
2566     else
2567     {
2568         sectionResults_rightSidePanel.continousLevelsWarning.setForeground(Color.red);
2569         sectionResults_rightSidePanel.continousLevelsWarning.setText("Levels have to be continous");
2570     }
2571
2572     myMethods.setCustomCursor();
2573 }
2574
2575 public void whenLeftInTree()
2576 {
2577     firstSplitPane.setLeftComponent(middleScrollPane);
2578     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
2579     firstSplitPane.setDividerLocation(screenWidth*14/20);
2580 }
2581 }
2582
2583 public class SectionResults_RightSidePanel extends JPanel implements ActionListener
2584 {
2585     JButton copyToClipboardButton;
2586
2587     JLabel conversionWarning = new JLabel("Conversion not updated");
2588     JLabel continousLevelsWarning = new JLabel("All levels continous");
2589
2590     public SectionResults_RightSidePanel()
2591     {
2592         copyToClipboardButton=new JButton("Copy to clipboard");
2593         copyToClipboardButton.addActionListener(this);
2594
2595         conversionWarning.setForeground(Color.red);
2596
2597         this.setLayout(new GridBagLayout());
2598         GridBagConstraints c = new GridBagConstraints();
2599
2600         c.gridx = 0;
2601         c.gridy = 0;
2602         c.gridwidth = 1;
2603         c.insets = new Insets(10,5,5,5);
2604         c.anchor = GridBagConstraints.PAGE_START;
2605         this.add(copyToClipboardButton, c);
2606
2607         c.insets = new Insets(30,5,5,5);
2608         c.gridy++;
2609         this.add(conversionWarning, c);
2610

```

```

2611     c.insets = new Insets(30,5,5,5);
2612     c.gridy++;
2613     c.weighty=1;
2614     this.add(continousLevelsWarning, c);
2615 }
2616
2617 public void actionPerformed(ActionEvent e)
2618 {
2619     if (e.getSource() == copyToClipboardButton)
2620     {
2621         sectionResults_middlePanel.copyToClipboard();
2622     }
2623 }
2624 }
2625
2626 private class SubjectResults_MiddlePanel extends JPanel
2627 {
2628     //Holds the table with the results for each subject
2629     JTable table = new JTable();
2630     Object[][] data;
2631     Object[][] tableData;
2632     String[] columnNames;
2633
2634     public SubjectResults_MiddlePanel()
2635     {
2636         table = new JTable();
2637     }
2638
2639     public void updateTable()
2640     {
2641         subjectResults_middlePanel.remove(table);
2642         data = new Object[myProjectData.numberOfSubjects][7+myProjectData.numberOfUDA];
2643         tableData = new Object[myProjectData.numberOfSubjects][7+myProjectData.numberOfUDA];
2644         columnNames= new String[7+myProjectData.numberOfUDA];
2645
2646         columnNames[0] = "Subject";
2647         columnNames[1] = "Tissue Volume";
2648         columnNames[2] = "Tissue Volume Left";
2649         columnNames[3] = "Tissue Volume Right";
2650         columnNames[4] = "Ventricle Volume";
2651         columnNames[5] = "Ventricle Volume Left";
2652         columnNames[6] = "Ventricle Volume Right";
2653
2654         for (int udaCounter=0;udaCounter<myProjectData.numberOfUDA;udaCounter++)
2655         {
2656             columnNames[7+udaCounter] = myProjectData.UDANames[udaCounter];
2657         }
2658
2659         double tempDouble;
2660         int counter=0;
2661         for (int i=0; i<myProjectData.numberOfSubjects; i++)
2662         {
2663             if (myProjectData.sectionExists[i][0])
2664             {
2665                 data[counter][0] = myProjectData.subjectNameList[i];
2666                 tableData[counter][0] = myProjectData.subjectNameList[i];

```



```

2667
2668     tempDouble = myProjectData.tissueVolume[i];
2669     data[counter][1] = tempDouble;
2670     tempDouble= (double) (Math.round(tempDouble*100))/100;
2671     tableData[counter][1] = tempDouble;
2672
2673     tempDouble = myProjectData.tissueVolumeLeft[i];
2674     data[counter][2] = tempDouble;
2675     tempDouble= (double) (Math.round(tempDouble*100))/100;
2676     tableData[counter][2] = tempDouble;
2677
2678     tempDouble = myProjectData.tissueVolumeRight[i];
2679     data[counter][3] = tempDouble;
2680     tempDouble= (double) (Math.round(tempDouble*100))/100;
2681     tableData[counter][3] = tempDouble;
2682
2683     tempDouble = myProjectData.ventricleVolume[i];
2684     data[counter][4] = tempDouble;
2685     tempDouble= (double) (Math.round(tempDouble*100))/100;
2686     tableData[counter][4] = tempDouble;
2687
2688     tempDouble = myProjectData.ventricleVolumeLeft[i];
2689     data[counter][5] = tempDouble;
2690     tempDouble= (double) (Math.round(tempDouble*100))/100;
2691     tableData[counter][5] = tempDouble;
2692
2693     tempDouble = myProjectData.ventricleVolumeRight[i];
2694     data[counter][6] = tempDouble;
2695     tempDouble= (double) (Math.round(tempDouble*100))/100;
2696     tableData[counter][6] = tempDouble;
2697
2698     for (int udaCounter=0;udaCounter<myProjectData.numberOfUDA;udaCounter++)
2699     {
2700         tempDouble = myProjectData.udaVolume[i][udaCounter];
2701         data[counter][7+udaCounter] = tempDouble;
2702         tempDouble= (double) (Math.round(tempDouble*100))/100;
2703         tableData[counter][7+udaCounter] = tempDouble;
2704     }
2705     counter++;
2706 }
2707 }
2708
2709 this.removeAll();
2710 table = new JTable(tableData, columnNames);
2711 table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
2712
2713 TableColumn column = null;
2714 for (int i=0; i<7+myProjectData.numberOfUDA ; i++)
2715 {
2716     column = table.getColumnModel().getColumn(i);
2717     column.setPreferredWidth(150);
2718 }
2719
2720 JScrollPane scrollPane = new JScrollPane(table);
2721
2722 firstSplitPane.setLeftComponent(scrollPane);

```

```

2723     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
2724     firstSplitPane.setDividerLocation(screenWidth*14/20);
2725     middleScrollPane.repaint();
2726 }
2727
2728 public void whenClickedInTree()
2729 {
2730     myMethods.setWaitCursor();
2731
2732     this.updateTable();
2733     middleScrollPane.getViewport().add(this);
2734     rightScrollPane.getViewport().add(subjectResults_rightSidePanel);
2735
2736     if(myProjectData.conversionIsUpdated)
2737     {
2738         subjectResults_rightSidePanel.conversionWarning.setForeground(Color.green);
2739         subjectResults_rightSidePanel.conversionWarning.setText("Conversion updated");
2740     }
2741     else
2742     {
2743         subjectResults_rightSidePanel.conversionWarning.setForeground(Color.red);
2744         subjectResults_rightSidePanel.conversionWarning.setText("Conversion not updated");
2745     }
2746
2747     if(myProjectData.allLevelsContinous)
2748     {
2749         subjectResults_rightSidePanel.continousLevelsWarning.setForeground(Color.green);
2750         subjectResults_rightSidePanel.continousLevelsWarning.setText("All levels continous");
2751     }
2752     else
2753     {
2754         subjectResults_rightSidePanel.continousLevelsWarning.setForeground(Color.red);
2755         subjectResults_rightSidePanel.continousLevelsWarning.setText("Levels have to be continous");
2756     }
2757
2758     myMethods.setCustomCursor();
2759 }
2760
2761 public void whenLeftInTree()
2762 {
2763     firstSplitPane.setLeftComponent(middleScrollPane);
2764     int screenWidth= (int) Toolkit.getDefaultToolkit().getScreenSize().getWidth();
2765     firstSplitPane.setDividerLocation(screenWidth*14/20);
2766 }
2767
2768 public void copyToClipboard()
2769 {
2770     String string = new String("");
2771     String temp = new String("");
2772
2773     for (int i=0;i<7+myProjectData.numberOfUDA;i++)
2774     {
2775         try
2776         {
2777             temp=columnNames[i];
2778             string= string + temp;

```

```

2779         if (i!=7+myProjectData.numberOfUDA-1)
2780             string = string + '\t';
2781     }catch(Exception e){}
2782 }
2783
2784 for (int i=0;i<myProjectData.numberOfSubjects;i++)
2785 {
2786     string=string+"\n";
2787     for(int j=0;j<7+myProjectData.numberOfUDA;j++)
2788     {
2789         try
2790         {
2791             temp=data[i][j].toString();
2792             string= string + temp;
2793             if (j!=7+myProjectData.numberOfUDA-1)
2794                 string = string + '\t';
2795             }catch(Exception e){}
2796         }
2797     }
2798
2799     StringSelection stringToClipboard = new StringSelection(string);
2800     Toolkit toolkit = Toolkit.getDefaultToolkit();
2801     Clipboard cp = toolkit.getSystemClipboard();
2802     cp.setContents(stringToClipboard, null);
2803 }
2804
2805 @Override
2806 public String toString()
2807 {
2808     return new String("Subject Results");
2809 }
2810 }
2811
2812 public class SubjectResults_RightSidePanel extends JPanel implements ActionListener
2813 {
2814     JButton copyToClipboardButton;
2815
2816     JLabel conversionWarning = new JLabel("Conversion not updated");
2817     JLabel continousLevelsWarning = new JLabel("All levels continous");
2818
2819     public SubjectResults_RightSidePanel()
2820     {
2821         copyToClipboardButton=new JButton("Copy to clipboard");
2822         copyToClipboardButton.addActionListener(this);
2823
2824         conversionWarning.setForeground(Color.red);
2825
2826         this.setLayout(new GridBagLayout());
2827         GridBagConstraints c = new GridBagConstraints();
2828
2829         c.gridx = 0;
2830         c.gridy = 0;
2831         c.gridwidth = 1;
2832         c.insets = new Insets(10,5,5,5);
2833         c.anchor = GridBagConstraints.PAGE_START;
2834         this.add(copyToClipboardButton, c);

```

```

2835
2836     c.insets = new Insets(30,5,5,5);
2837     c.gridy++;
2838     this.add(conversionWarning, c);
2839
2840     c.insets = new Insets(30,5,5,5);
2841     c.gridy++;
2842     c.weighty=1;
2843     this.add(continousLevelsWarning, c);
2844 }
2845
2846 public void actionPerformed(ActionEvent e)
2847 {
2848     if (e.getSource() == copyToClipboardButton)
2849     {
2850         subjectResults_middlePanel.copyToClipboard();
2851     }
2852 }
2853 }
2854
2855 public class TissueDetection_MiddlePanel extends JPanel implements MouseMotionListener,
MouseListener
2856 {
2857     //Lets the user find suitable tresholds for tissue detection. Moving the mouse over the image
2858     //displays RGB values and whether the pixels is currently interpreted as tissue or background.
2859     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
2860     BufferedImage displayImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
2861
2862     int displaySubject = 0;
2863     int displaySection = 0;
2864
2865     public TissueDetection_MiddlePanel()
2866     {
2867         addMouseMotionListener(this);
2868         addMouseListener(this);
2869     }
2870
2871     public void whenClickedInTree()
2872     {
2873         myMethods.setWaitCursor();
2874
2875         currentSectionData=new SectionData(myProjectData.imageWidth, myProjectData.imageHeight);
2876         currentSectionData.colorTreshold=myProjectData.colorTresholdGeneral;
2877         currentSectionData.intensityTreshold=myProjectData.intensityTresholdGeneral;
2878
2879         tissueDetection_rightSidePanel.colorTresholdTextField.setText(""+myProjectData.colorTresholdGeneral);
2880
2881         tissueDetection_rightSidePanel.intensityTresholdTextField.setText(""+myProjectData.intensityTresholdGeneral);
2882         tissueDetection_rightSidePanel.colorWarningLabel.setText(""+myProjectData.colorTresholdGeneral);
2883         tissueDetection_rightSidePanel.colorWarningLabel.setForeground(Color.green);
2884
2885         tissueDetection_rightSidePanel.intensityWarningLabel.setText(""+myProjectData.intensityTresholdGeneral);
2886         tissueDetection_rightSidePanel.intensityWarningLabel.setForeground(Color.green);
2887         tissueDetection_rightSidePanel.colorTresholdOk=true;
2888         tissueDetection_rightSidePanel.intensityTresholdOk=true;

```

```

2887     tissueDetection_rightSidePanel.applyToAllButton.setEnabled(true);
2888
2889     tissueDetection_middlePanel.updateDisplayImage();
2890     tissueDetection_middlePanel.setPreferredSize(new Dimension(displayImage.getWidth(),
displayImage.getHeight()));
2891     middleScrollPane.getViewport().add(tissueDetection_middlePanel);
2892     rightScrollPane.getViewport().add(tissueDetection_rightSidePanel);
2893
2894
tissueDetection_rightSidePanel.displayedImageLabel.setText(myProjectData.sectionName[displaySubject][displayS
ection]);
2895
2896     myMethods.setCustomCursor();
2897 }
2898
2899 public void updateDisplayImage()
2900 {
2901     int[] tissueColor = {255,0,122};
2902     try
2903     {
2904         File originalImageFile= new File(myMethods.getOriginalImageFileName(displaySubject,
displaySection));
2905         originalImage = ImageIO.read(originalImageFile);
2906         displayImage = ImageIO.read(originalImageFile);
2907     } catch (IOException event) { }
2908     boolean[][] tissue = myMethods.getTissuePixels(displayImage);
2909     myMethods.changePixelsInImage(displayImage, tissue, tissueColor);
2910
2911
tissueDetection_rightSidePanel.displayedImageLabel.setText(myProjectData.sectionName[displaySubject][displayS
ection]);
2912     }
2913
2914     public void mouseMoved(MouseEvent event)
2915     {
2916         Point currentPoint = event.getPoint();
2917         int[] color = new int[3];
2918
2919         currentPoint.x=100*currentPoint.x/myProjectData.zoomFactor;
2920         currentPoint.y=100*currentPoint.y/myProjectData.zoomFactor;
2921
2922         if (currentPoint.x<originalImage.getWidth()
2923             && currentPoint.y<originalImage.getHeight())
2924         {
2925             //Mouse is over the top image
2926             WritableRaster raster = originalImage.getRaster();
2927
2928             color=raster.getPixel(currentPoint.x,currentPoint.y,color);
2929
2930             tissueDetection_rightSidePanel.redValue.setText("Red: "+color[0]);
2931             tissueDetection_rightSidePanel.greenValue.setText("Green: "+color[1]);
2932             tissueDetection_rightSidePanel.blueValue.setText("Blue: "+color[2]);
2933
2934             int firstInt = color[0]+color[2];
2935             int secondInt = color[0]+color[1]+color[2];
2936             double tempDouble = (double) firstInt/secondInt;

```

```

2937 tempDouble = tempDouble*1000;
2938 tempDouble = Math.round(tempDouble);
2939 tempDouble = tempDouble/10;
2940
2941 tissueDetection_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
2942 tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
2943 if(myMethods.pixellIsTissue(color))
2944 {
2945     tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
2946 }
2947 else
2948 {
2949     tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Background");
2950 }
2951
2952 tissueDetection_rightSidePanel.repaint();
2953 }
2954 else if (currentPoint.x<originalImage.getWidth()
2955     && currentPoint.y<2*originalImage.getHeight()
2956     && currentPoint.y>originalImage.getHeight())
2957 {
2958     //Mouse is over the lower image
2959     currentPoint.y=currentPoint.y-originalImage.getHeight();
2960
2961     WritableRaster raster = originalImage.getRaster();
2962
2963     color=raster.getPixel(currentPoint.x,currentPoint.y,color);
2964
2965     tissueDetection_rightSidePanel.redValue.setText("Red: "+color[0]);
2966     tissueDetection_rightSidePanel.greenValue.setText("Green: "+color[1]);
2967     tissueDetection_rightSidePanel.blueValue.setText("Blue: "+color[2]);
2968
2969     int firstInt = color[0]+color[2];
2970     int secondInt = color[0]+color[1]+color[2];
2971     double tempDouble = (double) firstInt/secondInt;
2972     tempDouble = tempDouble*1000;
2973     tempDouble = Math.round(tempDouble);
2974     tempDouble = tempDouble/10;
2975
2976     tissueDetection_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
2977     tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
2978     if(myMethods.pixellIsTissue(color))
2979     {
2980         tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
2981     }
2982     else
2983     {
2984         tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: Background");
2985     }
2986
2987     tissueDetection_rightSidePanel.repaint();
2988 }
2989
2990 else
2991 {
2992     //Mouse is over middle panel but not over one of the panels

```

```

2993     tissueDetection_rightSidePanel.redValue.setText("Red: ---");
2994     tissueDetection_rightSidePanel.greenValue.setText("Green: ---");
2995     tissueDetection_rightSidePanel.blueValue.setText("Blue: ---");
2996     tissueDetection_rightSidePanel.colorValueLabel.setText("Color: ---");
2997     tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
2998     tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: ---");
2999
3000     tissueDetection_rightSidePanel.repaint();
3001 }
3002 }
3003
3004 public void mouseExited(MouseEvent e)
3005 {
3006     tissueDetection_rightSidePanel.redValue.setText("Red: ---");
3007     tissueDetection_rightSidePanel.greenValue.setText("Green: ---");
3008     tissueDetection_rightSidePanel.blueValue.setText("Blue: ---");
3009     tissueDetection_rightSidePanel.colorValueLabel.setText("Color: ---");
3010     tissueDetection_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
3011     tissueDetection_rightSidePanel.resultLabel.setText("Pixel is: ---");
3012
3013     tissueDetection_rightSidePanel.repaint();
3014 }
3015
3016 public void mouseClicked(MouseEvent e) {}
3017 public void mousePressed(MouseEvent e) {}
3018 public void mouseReleased(MouseEvent event) {}
3019 public void mouseDragged(MouseEvent e) {}
3020 public void mouseEntered(MouseEvent e) {}
3021
3022 @Override
3023 public void paint(Graphics g)
3024 {
3025     super.paint(g);
3026
3027     int width = displayImage.getWidth();
3028     int height = displayImage.getHeight();
3029
3030     int drawWidth = width*myProjectData.zoomFactor/100;
3031     int drawHeight = height*myProjectData.zoomFactor/100;
3032
3033     tissueDetection_middlePanel.setPreferredSize(new Dimension(drawWidth+300, 2*drawHeight));
3034
3035     g.drawString("Original image", drawWidth+10, 15);
3036     g.drawImage(originalImage,0,0, drawWidth, drawHeight,this);
3037
3038     g.drawString("Detected tissue", drawWidth+10, drawHeight+15);
3039     g.drawImage(displayImage, 0,drawHeight, drawWidth, drawHeight, this);
3040
3041     this.revalidate();
3042 }
3043 @Override
3044 public String toString()
3045 {
3046     return new String("Tissue Detection");
3047 }
3048 }

```

```

3049
3050 public class TissueDetection_RightSidePanel extends JPanel implements ActionListener
3051 {
3052     JButton helpButton;
3053
3054     JLabel colorTresholdLabel = new JLabel("Color treshold");
3055     JTextField colorTresholdTextField = new JTextField(""+myProjectData.colorTresholdGeneral, 8);
3056     JLabel colorWarningLabel = new JLabel(""+myProjectData.colorTresholdGeneral);
3057     JLabel intensityTresholdLabel = new JLabel("Intensity treshold");
3058     JTextField intensityTresholdTextField = new JTextField(""+myProjectData.intensityTresholdGeneral, 8);
3059     JLabel intensityWarningLabel = new JLabel(""+myProjectData.intensityTresholdGeneral);
3060
3061     JLabel redValue = new JLabel("Red: ---");
3062     JLabel greenValue = new JLabel("Green: ---");
3063     JLabel blueValue = new JLabel("Blue: ---");
3064
3065     JLabel colorValueLabel = new JLabel("Color: ");
3066     JLabel intensityValueLabel = new JLabel("Intensity: ");
3067     JLabel resultLabel = new JLabel("Pixel is: ");
3068
3069     JButton applyToAllButton;
3070     JButton previousButton;
3071     JLabel displayedImageLabel = new JLabel("---");
3072     JButton nextButton;
3073
3074     boolean colorTresholdOk=true;
3075     boolean intensityTresholdOk=true;
3076
3077     public TissueDetection_RightSidePanel()
3078     {
3079         colorTresholdTextField.addActionListener(this);
3080         intensityTresholdTextField.addActionListener(this);
3081
3082         helpButton=new JButton("Help");
3083         helpButton.addActionListener(this);
3084
3085         applyToAllButton=new JButton("Apply To All");
3086         applyToAllButton.addActionListener(this);
3087
3088         previousButton=new JButton("Previous image");
3089         previousButton.addActionListener(this);
3090
3091         nextButton=new JButton("Next image");
3092         nextButton.addActionListener(this);
3093
3094         this.setLayout(new GridBagLayout());
3095         GridBagConstraints c = new GridBagConstraints();
3096
3097         c.gridx = 0;
3098         c.gridy = 0;
3099         c.gridwidth = 1;
3100         c.insets = new Insets(15,5,25,5);
3101         c.anchor = GridBagConstraints.PAGE_START;
3102         this.add(helpButton, c);
3103
3104         c.gridy++;

```



```
3105     c.insets = new Insets(5,5,5,5);
3106     this.add(colorTresholdLabel, c);
3107
3108     c.gridy++;
3109     this.add(colorTresholdTextField, c);
3110
3111     c.gridy++;
3112     this.add(colorWarningLabel, c);
3113
3114     c.gridy++;
3115     this.add(intensityTresholdLabel, c);
3116
3117     c.insets = new Insets(5,5,5,5);
3118     c.gridy++;
3119     this.add(intensityTresholdTextField, c);
3120
3121     c.gridy++;
3122     c.insets = new Insets(5,5,5,5);
3123     this.add(intensityWarningLabel, c);
3124
3125     c.gridy++;
3126     c.insets = new Insets(25,5,5,5);
3127     this.add(redValue, c);
3128
3129     c.insets = new Insets(5,5,5,5);
3130     c.gridy++;
3131     this.add(greenValue, c);
3132
3133     c.gridy++;
3134     this.add(blueValue, c);
3135
3136     c.gridy++;
3137     c.insets = new Insets(25,5,5,5);
3138     this.add(colorValueLabel,c);
3139
3140     c.gridy++;
3141     c.insets = new Insets(5,5,5,5);
3142     this.add(intensityValueLabel,c);
3143
3144     c.gridy++;
3145     this.add(resultLabel, c);
3146
3147     c.gridy++;
3148     c.insets = new Insets(25,5,30,5);
3149     this.add(applyToAllButton,c);
3150
3151     c.gridy++;
3152     c.insets = new Insets(5,5,5,5);
3153     this.add(previousButton,c);
3154
3155     c.gridy++;
3156     this.add(displayedImageLabel,c);
3157
3158     c.gridy++;
3159     c.weighty = 1;
3160     this.add(nextButton,c);
```

```

3161     }
3162
3163     public void actionPerformed(ActionEvent e)
3164     {
3165         if (e.getSource()==helpButton)
3166         {
3167             String helpText = myMethods.getHelpTextTissueDetection();
3168             helpDialog.newText(helpText);
3169         }
3170
3171         if (e.getSource()==colorTresholdTextField)
3172         {
3173             try
3174             {
3175                 double tempDouble = Double.parseDouble(colorTresholdTextField.getText());
3176                 if (tempDouble>=0 && tempDouble<=100)
3177                 {
3178                     myProjectData.colorTresholdGeneral = tempDouble;
3179                     currentSectionData.colorTreshold = tempDouble;
3180                     colorWarningLabel.setText(""+myProjectData.colorTresholdGeneral);
3181                     colorWarningLabel.setForeground(Color.green);
3182                     colorTresholdOk=true;
3183                 }
3184                 else
3185                 {
3186                     colorWarningLabel.setForeground(Color.red);
3187                     colorWarningLabel.setText("Use range 0-100");
3188                     colorTresholdOk=false;
3189                 }
3190             } catch (Exception error)
3191             {
3192                 colorWarningLabel.setForeground(Color.red);
3193                 colorWarningLabel.setText("Not a number");
3194                 colorTresholdOk=false;
3195             }
3196
3197             if (colorTresholdOk && intensityTresholdOk)
3198             {
3199                 applyToAllButton.setEnabled(true);
3200             }
3201             else
3202             {
3203                 applyToAllButton.setEnabled(false);
3204             }
3205
3206             upperScrollPane.requestFocus();
3207
3208             tissueDetection_middlePanel.updateDisplayImage();
3209             tissueDetection_middlePanel.repaint();
3210         }
3211         if (e.getSource()==intensityTresholdTextField)
3212         {
3213             try
3214             {
3215                 double tempDouble = Double.parseDouble(intensityTresholdTextField.getText());
3216                 if (tempDouble>=0 && tempDouble<=765)

```

```

3217     {
3218         myProjectData.intensityTresholdGeneral = tempDouble;
3219         currentSectionData.intensityTreshold = tempDouble;
3220         intensityWarningLabel.setText(""+myProjectData.intensityTresholdGeneral);
3221         intensityWarningLabel.setForeground(Color.green);
3222         intensityTresholdOk=true;
3223     }
3224     else
3225     {
3226         intensityWarningLabel.setForeground(Color.red);
3227         intensityWarningLabel.setText("Use range 0-765");
3228         intensityTresholdOk=false;
3229     }
3230 }catch(Exception error)
3231 {
3232     intensityWarningLabel.setForeground(Color.red);
3233     intensityWarningLabel.setText("Not a number");
3234     intensityTresholdOk=false;
3235 }
3236
3237 if (colorTresholdOk && intensityTresholdOk)
3238 {
3239     applyToAllButton.setEnabled(true);
3240 }
3241 else
3242 {
3243     applyToAllButton.setEnabled(false);
3244 }
3245
3246 upperScrollPane.requestFocus();
3247
3248 tissueDetection_middlePanel.updateDisplayImage();
3249 tissueDetection_middlePanel.repaint();
3250 }
3251
3252 if (e.getSource()==applyToAllButton)
3253 {
3254     //The settings for color treshold and intensity treshold
3255     //is applied to all sections. All areas and volumes are
3256     //also recalculated to match the new tresholds.
3257
3258     int numberOfSections=myProjectData.totalNumberOfSections;
3259
3260     progressMonitor = new ProgressMonitor(null,
3261         "Analyzing sections",
3262         "Sections analyzed: 0", 0, numberOfSections);
3263
3264     progressMonitor.setMillisToDecideToPopup(0);
3265     progressMonitor.setMillisToPopup(0);
3266     progressMonitor.setProgress(0);
3267     progressMonitor.setNote("Sections analyzed: 0");
3268
3269     AnalyzeAllSections analyzeAllSections = new AnalyzeAllSections();
3270     analyzeAllSections.execute();
3271 }
3272

```

```

3273     if (e.getSource() == previousButton)
3274     {
3275         int subject = tissueDetection_middlePanel.displaySubject;
3276         int section = tissueDetection_middlePanel.displaySection;
3277
3278         if (section > 0)
3279         {
3280             tissueDetection_middlePanel.displaySection--;
3281         }
3282         else
3283         {
3284             if (subject > 0)
3285             {
3286                 tissueDetection_middlePanel.displaySubject--;
3287                 for (int i = 0; i < myProjectData.numberOfLevels; i++)
3288                 {
3289                     if (myProjectData.sectionExists[subject-1][i])
3290                         tissueDetection_middlePanel.displaySection = i;
3291                 }
3292             }
3293             else
3294             {
3295                 tissueDetection_middlePanel.displaySubject = myProjectData.numberOfSubjects-1;
3296                 for (int i = 0; i < myProjectData.numberOfLevels; i++)
3297                 {
3298                     if (myProjectData.sectionExists[myProjectData.numberOfSubjects-1][i])
3299                         tissueDetection_middlePanel.displaySection = i;
3300                 }
3301             }
3302         }
3303
3304         tissueDetection_middlePanel.updateDisplayImage();
3305         tissueDetection_middlePanel.repaint();
3306     }
3307
3308     if (e.getSource() == nextButton)
3309     {
3310         int subject = tissueDetection_middlePanel.displaySubject;
3311         int section = tissueDetection_middlePanel.displaySection;
3312
3313         if (subject == myProjectData.numberOfSubjects-1)
3314         {
3315             if (section == myProjectData.numberOfLevels-1)
3316             {
3317                 tissueDetection_middlePanel.displaySubject = 0;
3318                 tissueDetection_middlePanel.displaySection = 0;
3319             }
3320             else
3321             {
3322                 if (myProjectData.sectionExists[subject][section+1])
3323                 {
3324                     tissueDetection_middlePanel.displaySection++;
3325                 }
3326                 else
3327                 {
3328                     tissueDetection_middlePanel.displaySubject = 0;

```

```

3329         tissueDetection_middlePanel.displaySection=0;
3330     }
3331 }
3332 }
3333 else
3334 {
3335     if (section ==myProjectData.numberOfLevels-1)
3336     {
3337         tissueDetection_middlePanel.displaySubject++;
3338         tissueDetection_middlePanel.displaySection=0;
3339     }
3340     else
3341     {
3342         if (myProjectData.sectionExists[subject][section+1])
3343         {
3344             tissueDetection_middlePanel.displaySection++;
3345         }
3346         else
3347         {
3348             tissueDetection_middlePanel.displaySubject++;
3349             tissueDetection_middlePanel.displaySection=0;
3350         }
3351     }
3352 }
3353 tissueDetection_middlePanel.updateDisplayImage();
3354 tissueDetection_middlePanel.repaint();
3355 }
3356 }
3357
3358 public class AnalyzeAllSections extends SwingWorker<Void, Void>
3359 {
3360     @Override
3361     public Void doInBackground()
3362     {
3363         frame.setVisible(false);
3364         int counter=0;
3365         progressMonitor.setProgress(counter);
3366         for (int subject=0;subject<myProjectData.numberOfSubjects;subject++)
3367         {
3368             for (int section=0; section<myProjectData.numberOfLevels; section++)
3369             {
3370                 if (myProjectData.sectionExists[subject][section])
3371                 {
3372                     currentSectionData=myMethods.readSectionData(subject, section);
3373
3374                     currentSectionData.colorTreshold=myProjectData.colorTresholdGeneral;
3375                     currentSectionData.intensityTreshold=myProjectData.intensityTresholdGeneral;
3376
3377                     myMethods.saveSectionData(subject, section, currentSectionData);
3378
3379                     myMethods.analyzeSection(subject, section);
3380                     myMethods.updateSectionResultsObject(subject, section);
3381                     counter++;
3382                     progressMonitor.setProgress(counter);
3383                     progressMonitor.setNote("Sections analyzed: "+counter);
3384

```

```

3385         if (progressMonitor.isCanceled())
3386         {
3387             break;
3388         }
3389     }
3390 }
3391 myMethods.analyzeSubject(subject);
3392 if (progressMonitor.isCanceled())
3393 {
3394     errorDialog.newText(myMethods.getErrorTextApplyToAllInterrupted());
3395     errorDialog.setVisible(true);
3396     break;
3397 }
3398 }
3399 frame.setVisible(true);
3400 return null;
3401 }
3402
3403 @Override
3404 public void done() {}
3405 }
3406 }
3407
3408 private class Conversion_MiddlePanel extends JPanel implements MouseListener
3409 {
3410     //Lets the user click the conversion image to select a distance in pixels.
3411     //The corresponding length in millimeters can then be entered in the right
3412     //side panel. This is used to convert the area measurments in pixels to
3413     //square millimeters and cubic millimeters.
3414     BufferedImage conversionImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
3415
3416     boolean numberEntered=false;
3417
3418     public Conversion_MiddlePanel()
3419     {
3420         addMouseListener(this);
3421     }
3422
3423     public void whenClickedInTree()
3424     {
3425         middleScrollPane.getViewPort().add(conversion_middlePanel);
3426         rightScrollPane.getViewPort().add(conversion_rightSidePanel);
3427         conversion_middlePanel.generateConversionImage();
3428         conversion_rightSidePanel.distanceLabel.setText(""+myProjectData.distancePixels);
3429         conversion_rightSidePanel.distanceLabel.setForeground(Color.green);
3430         if (myProjectData.distancePixels==0)
3431         {
3432             conversion_rightSidePanel.distanceLabel.setText("Click image");
3433             conversion_rightSidePanel.distanceLabel.setForeground(Color.red);
3434         }
3435
3436         conversion_rightSidePanel.distancemmTextField.setText(""+myProjectData.distanceMM);
3437         conversion_rightSidePanel.numberOfMMLLabel.setForeground(Color.green);
3438         conversion_rightSidePanel.numberOfMMLLabel.setText(""+myProjectData.distanceMM);
3439
3440         if (myProjectData.distanceMM<=0)

```

```

3441     {
3442         conversion_rightSidePanel.numberOfMMLLabel.setForeground(Color.red);
3443         conversion_rightSidePanel.numberOfMMLLabel.setText("Must be more than 0");
3444         conversion_middlePanel.numberEntered = false;
3445     }
3446
3447     if (myProjectData.conversionIsUpdated)
3448     {
3449         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.green);
3450
3451         conversion_rightSidePanel.conversionFactorLabel.setText(""+Math.round(myProjectData.conversionFactor));
3452     }
3453     else
3454     {
3455         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.red);
3456         conversion_rightSidePanel.conversionFactorLabel.setText("Needs updating");
3457     }
3458     public void whenLeftInTree()
3459     {
3460
3461     }
3462
3463     public void generateConversionImage()
3464     {
3465         String conversionFileName = new String(myProjectData.projectLocation+"/ConversionImage");
3466         File conversionFile = new File(conversionFileName);
3467
3468         try
3469         {
3470             conversionImage = (ImageIO.read(conversionFile));
3471         } catch (Exception error) {}
3472
3473         boolean[][] linePixels = new boolean[conversionImage.getWidth()][conversionImage.getHeight()];
3474
3475         for (int i=0;i<conversionImage.getWidth();i++)
3476             for (int j=0;j<conversionImage.getHeight();j++)
3477                 linePixels[i][j]=false;
3478
3479         linePixels = myMethods.drawLineBoolean(linePixels,
3480             myProjectData.firstConversionPoint, myProjectData.secondConversionPoint);
3481
3482         linePixels = myMethods.widenLine(linePixels, myProjectData.lineWidth);
3483         myMethods.changePixelsInImage(conversionImage, linePixels, myProjectData.lineColor);
3484     }
3485
3486     public void calculateDistanceInPixels()
3487     {
3488         double distanceX;
3489         double distanceY;
3490         double distance;
3491
3492         distanceX = Math.abs(myProjectData.firstConversionPoint.x-
3493             myProjectData.secondConversionPoint.x);
3493         distanceY = Math.abs(myProjectData.firstConversionPoint.y-
3494             myProjectData.secondConversionPoint.y);

```

```

3494 distance = Math.sqrt(distanceX*distanceX+distanceY*distanceY);
3495 distance = Math.round(distance);
3496
3497 myProjectData.distancePixels= (int) distance;
3498
3499 conversion_rightSidePanel.distanceLabel.setForeground(Color.green);
3500 conversion_rightSidePanel.distanceLabel.setText(""+myProjectData.distancePixels);
3501
3502 if (myProjectData.distancePixels==0)
3503 {
3504     conversion_rightSidePanel.distanceLabel.setForeground(Color.red);
3505     conversion_rightSidePanel.distanceLabel.setText("Click image");
3506 }
3507 }
3508 }
3509
3510 public void calculateConversionFactor()
3511 {
3512     this.calculateDistanceInPixels();
3513
3514     double distanceMM=0;
3515
3516     String newValue = conversion_rightSidePanel.distancemmTextField.getText();
3517     try
3518     {
3519         distanceMM = Double.parseDouble(newValue);
3520         myProjectData.distanceMM = distanceMM;
3521         conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.green);
3522         conversion_rightSidePanel.numberOfMMLabel.setText(""+distanceMM);
3523         numberEntered=true;
3524
3525         if (distanceMM<=0)
3526         {
3527             conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.red);
3528             conversion_rightSidePanel.numberOfMMLabel.setText("Must be more than 0");
3529             conversion_middlePanel.numberEntered = false;
3530         }
3531     }
3532     catch (NumberFormatException error)
3533     {
3534         conversion_rightSidePanel.numberOfMMLabel.setForeground(Color.red);
3535         conversion_rightSidePanel.numberOfMMLabel.setText("Not a number");
3536         numberEntered = false;
3537     }
3538
3539     if(myProjectData.distancePixels>0 && distanceMM>0)
3540     {
3541         myProjectData.conversionFactor = (myProjectData.distancePixels*myProjectData.distancePixels)
3542             /(distanceMM*distanceMM);
3543
3544         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.green);
3545
3546         conversion_rightSidePanel.conversionFactorLabel.setText(""+Math.round(myProjectData.conversionFactor));
3547         myProjectData.conversionIsUpdated=true;
3548     }
3549     else

```



```

3549     {
3550         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.red);
3551         conversion_rightSidePanel.conversionFactorLabel.setText("Needs updating");
3552         myProjectData.conversionIsUpdated=false;
3553     }
3554 }
3555
3556 @Override
3557 public void paint(Graphics g)
3558 {
3559     super.paint(g);
3560
3561     int width = conversionImage.getWidth();
3562     int height = conversionImage.getHeight();
3563
3564     conversion_middlePanel.setPreferredSize(new Dimension(width*myProjectData.zoomFactor/100,
3565 height*myProjectData.zoomFactor/100));
3566
3567     g.drawImage(conversionImage,
3568         0,0, width*myProjectData.zoomFactor/100, height*myProjectData.zoomFactor/100,
3569         0,0, width, height,
3570         this);
3571     this.revalidate();
3572 }
3573
3574 public void mouseClicked(MouseEvent event)
3575 {
3576     Point currentPoint=event.getPoint();
3577
3578     currentPoint.x=100*currentPoint.x/myProjectData.zoomFactor;
3579     currentPoint.y=100*currentPoint.y/myProjectData.zoomFactor;
3580
3581     if (currentPoint.x<conversionImage.getWidth() && currentPoint.y<conversionImage.getHeight())
3582     {
3583         if (event.getButton()==MouseEvent.BUTTON1)
3584         {
3585             myProjectData.firstConversionPoint = currentPoint;
3586         }
3587         if (event.getButton()==MouseEvent.BUTTON3)
3588         {
3589             myProjectData.secondConversionPoint = currentPoint;
3590         }
3591
3592         this.calculateDistanceInPixels();
3593         conversion_rightSidePanel.conversionFactorLabel.setForeground(Color.red);
3594         conversion_rightSidePanel.conversionFactorLabel.setText("Needs updating");
3595         myProjectData.conversionIsUpdated=false;
3596
3597         this.generateConversionImage();
3598         repaint();
3599     }
3600 }
3601
3602 public void mousePressed(MouseEvent event) {}
3603 public void mouseEntered(MouseEvent event) {}

```

```

3604     public void mouseExited(MouseEvent event) {}
3605     public void mouseReleased(MouseEvent event) {}
3606
3607     @Override
3608     public String toString()
3609     {
3610         return new String("Conversion Panel");
3611     }
3612 }
3613
3614 public class Conversion_RightSidePanel extends JPanel implements ActionListener
3615 {
3616     JButton helpButton;
3617
3618     JLabel descriptionLabel1a = new JLabel("Left click for first point");
3619     JLabel descriptionLabel1b = new JLabel("Right click for second point");
3620     JLabel descriptionLabel2 = new JLabel("Distance in pixels will be calculated");
3621     JLabel descriptionLabel3 = new JLabel("Enter corresponding distance in mm");
3622
3623     JLabel infoLabel1 = new JLabel("Distance (pixels):");
3624     JLabel distanceLabel = new JLabel("0");
3625     JLabel infoLabel2 = new JLabel("Distance (mm):");
3626     JTextField distancemmTextField = new JTextField("0",8);
3627     JLabel numberOfMMLLabel = new JLabel("Cant be zero");
3628     JLabel infoLabel3 = new JLabel("Pixels/mm2:");
3629     JLabel conversionFactorLabel = new JLabel("0");
3630
3631     JButton updateConversionFactorButton;
3632
3633     public Conversion_RightSidePanel()
3634     {
3635         helpButton = new JButton("Help");
3636         helpButton.addActionListener(this);
3637         updateConversionFactorButton = new JButton("Update Conversion");
3638         updateConversionFactorButton.addActionListener(this);
3639
3640         distancemmTextField.addActionListener(this);
3641
3642         this.setLayout(new GridBagLayout());
3643         GridBagConstraints c = new GridBagConstraints();
3644
3645         c.gridx = 0;
3646         c.gridy = 0;
3647         c.gridwidth = 1;
3648         c.anchor = GridBagConstraints.PAGE_START;
3649         c.insets = new Insets(10,5,20,5);
3650
3651         this.add(helpButton, c);
3652
3653         c.gridy ++;
3654         c.insets = new Insets(5,5,5,5);
3655         this.add(descriptionLabel1a, c);
3656
3657         c.gridy ++;
3658         this.add(descriptionLabel1b, c);
3659

```

```

3660     c.gridy ++;
3661     this.add(descriptionLabel2, c);
3662
3663     c.gridy ++;
3664     this.add(descriptionLabel3, c);
3665
3666     c.gridy ++;
3667     c.insets = new Insets(20,5,5,5);
3668     this.add(infoLabel1, c);
3669
3670     c.gridy ++;
3671     c.insets = new Insets(5,5,5,5);
3672     this.add(distanceLabel, c);
3673
3674     c.gridy ++;
3675     c.insets = new Insets(20,5,5,5);
3676     this.add(infoLabel2, c);
3677
3678     c.gridy ++;
3679     c.insets = new Insets(5,5,5,5);
3680     this.add(distancemmTextField, c);
3681
3682     c.gridy ++;
3683     this.add(numberOfMMLLabel, c);
3684
3685     c.gridy ++;
3686     c.insets = new Insets(20,5,5,5);
3687     this.add(infoLabel3, c);
3688
3689     c.gridy ++;
3690     c.insets = new Insets(5,5,5,5);
3691     this.add(conversionFactorLabel, c);
3692
3693     c.gridy ++;
3694     c.insets = new Insets(20,5,5,5);
3695     c.weighty = 1;
3696     this.add(updateConversionFactorButton, c);
3697 }
3698
3699 public void actionPerformed(ActionEvent e)
3700 {
3701     if (e.getSource()==distancemmTextField)
3702     {
3703         upperScrollPane.requestFocus();
3704
3705         double distanceMM=0;
3706         String newValue = conversion_rightSidePanel.distancemmTextField.getText();
3707         try
3708         {
3709             distanceMM = Double.parseDouble(newValue);
3710             if (myProjectData.distanceMM != distanceMM)
3711             {
3712                 myProjectData.distanceMM = distanceMM;
3713                 conversionFactorLabel.setText("Needs updating");
3714                 conversionFactorLabel.setForeground(Color.red);
3715             }

```

```

3716         conversion_rightSidePanel.numberOfMMLLabel.setForeground(Color.green);
3717         conversion_rightSidePanel.numberOfMMLLabel.setText(""+distanceMM);
3718         conversion_middlePanel.numberEntered=true;
3719
3720         if (distanceMM<=0)
3721         {
3722             conversion_rightSidePanel.numberOfMMLLabel.setForeground(Color.red);
3723             conversion_rightSidePanel.numberOfMMLLabel.setText("Must be more than 0");
3724             conversion_middlePanel.numberEntered = false;
3725         }
3726     }
3727     catch (NumberFormatException error)
3728     {
3729         conversion_rightSidePanel.numberOfMMLLabel.setForeground(Color.red);
3730         conversion_rightSidePanel.numberOfMMLLabel.setText("Not a number");
3731         conversion_middlePanel.numberEntered = false;
3732     }
3733 }
3734 if (e.getSource()==helpButton)
3735 {
3736     String helpText = myMethods.getHelpTextChoseConversionImage();
3737     helpDialog.newText(helpText);
3738 }
3739
3740 if (e.getSource()==updateConversionFactorButton)
3741 {
3742     //Reanalyses the data to match the new conversion factor
3743     myMethods.setWaitCursor();
3744
3745     conversion_middlePanel.calculateConversionFactor();
3746
3747     if (conversion_middlePanel.numberEntered && myProjectData.distanceMM>0 &&
myProjectData.distancePixels>0)
3748     {
3749         progressMonitor = new ProgressMonitor(null,
3750             "Recalculating volumes",
3751             "Subjects analyzed: 0", 0, myProjectData.numberOfSubjects);
3752
3753         progressMonitor.setMillisToDecideToPopup(0);
3754         progressMonitor.setMillisToPopup(0);
3755         progressMonitor.setProgress(0);
3756         UpdateConversion updateConversion = new UpdateConversion();
3757         updateConversion.execute();
3758     }
3759     myMethods.setCustomCursor();
3760 }
3761 }
3762
3763 public class UpdateConversion extends SwingWorker<Void, Void>
3764 {
3765     @Override
3766     public Void doInBackground()
3767     {
3768         frame.setVisible(false);
3769         int counter=0;
3770         progressMonitor.setProgress(counter);

```

```

3771     for (int subject=0;subject<myProjectData.numberOfSubjects;subject++)
3772     {
3773         for (int section=0; section<myProjectData.numberOfLevels; section++)
3774         {
3775             if (myProjectData.sectionExists[subject][section])
3776             {
3777                 myMethods.updateSectionResultsObject(subject, section);
3778             }
3779         }
3780         myMethods.analyzeSubject(subject);
3781         counter++;
3782         progressMonitor.setProgress(counter);
3783         progressMonitor.setNote("Subjects analyzed: "+counter);
3784
3785         if (progressMonitor.isCanceled())
3786         {
3787             errorDialog.newText(myMethods.getErrorTextUpdateConversionInterrupted());
3788             errorDialog.setVisible(true);
3789             break;
3790         }
3791     }
3792     frame.setVisible(true);
3793     return null;
3794 }
3795
3796 @Override
3797 public void done() {}
3798 }
3799 }
3800
3801 public class Subject_MiddlePanel extends JPanel
3802 {
3803     //Displays the original images for all sections in this subject
3804     //to get an overview
3805     int subject=0;
3806
3807     BufferedImage[] images = new BufferedImage[100];
3808     int imageHeigth=10;
3809     int imageWidth=10;
3810     int sectionsInSubject;
3811
3812     public void whenClickedInTree(int currentSubject)
3813     {
3814         myMethods.setWaitCursor();
3815
3816         subject=currentSubject;
3817         sectionsInSubject=0;
3818         File file;
3819         String fileName;
3820         for (int i=0;i<myProjectData.numberOfLevels;i++)
3821         {
3822             if (myProjectData.sectionExists[subject][i])
3823             {
3824                 try
3825                 {
3826                     fileName = myMethods.getOriginalImageFileName(subject, i);

```

```

3827         file = new File(fileName);
3828         images[i]=ImageIO.read(file);
3829         imageWidth=images[i].getWidth();
3830         imageHeigth=images[i].getHeight();
3831         sectionsInSubject++;
3832     } catch (IOException event) { }
3833     }
3834 }
3835
3836 middleScrollPane.getViewPort().add(subject_middlePanel);
3837 rightScrollPane.getViewPort().add(subject_rightSidePanel);
3838 middleScrollPane.repaint();
3839
3840 myMethods.setCustomCursor();
3841 }
3842
3843 @Override
3844 public void paint(Graphics g)
3845 {
3846     super.paint(g);
3847
3848     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
3849     int drawHeight = imageHeigth*myProjectData.zoomFactor/100;
3850
3851     FontMetrics fontMetrics = g.getFontMetrics();
3852     int fontHeight = fontMetrics.getHeight();
3853     int subjectNumber=0;
3854
3855     subject_middlePanel.setPreferredSize(new Dimension(drawWidth+300,
sectionsInSubject*drawHeight));
3856
3857     for (int i=0;i<sectionsInSubject;i++)
3858     {
3859         subjectNumber=i+1;
3860         g.drawImage(images[i], 0, drawHeight*i, drawWidth, drawHeight, this);
3861         g.drawString("Section " +subjectNumber, drawWidth+5, drawHeight*i+fontHeight);
3862         g.drawString(""+myProjectData.sectionName[subject][i], drawWidth+5,
drawHeight*i+2*fontHeight);
3863         g.drawString("Level: " +myProjectData.bregmaLevels[subject][i], drawWidth+5,
drawHeight*i+3*fontHeight);
3864     }
3865
3866     revalidate();
3867 }
3868 }
3869
3870 public class Subject_RightSidePanel extends JPanel
3871 {
3872     JLabel infoLabel = new JLabel("Subject overview");
3873
3874     public Subject_RightSidePanel()
3875     {
3876         this.setLayout(new GridBagLayout());
3877         GridBagConstraints c = new GridBagConstraints();
3878
3879         c.gridx = 0;

```

```

3880     c.gridy = 0;
3881     c.gridwidth = 1;
3882     c.weighty=1;
3883     c.insets = new Insets(10,5,5,5);
3884     c.anchor = GridBagConstraints.PAGE_START;
3885     this.add(infoLabel, c);
3886 }
3887 }
3888
3889 private class Section_MiddlePanel extends JPanel
3890 {
3891     //Displays the original image and derived images to get
3892     //an overview of this section.
3893     int subject=0;
3894     int section=0;
3895
3896     BufferedImage originalImage;
3897     BufferedImage adjustedImage;
3898     BufferedImage areasImage;
3899     BufferedImage[] UDImages;
3900
3901     int imageHeight=10;
3902     int imageWidth=10;
3903
3904     public void whenClickedInTree(int currentSubject, int currentSection)
3905     {
3906         subject=currentSubject;
3907         section=currentSection;
3908
3909         UDImages=new BufferedImage[myProjectData.numberOfUDA];
3910
3911         currentSectionData=myMethods.readSectionData(subject, section);
3912
3913         section_rightSidePanel.levelLabel.setText(""+ myProjectData.bregmaLevels[subject][section]);
3914         section_rightSidePanel.bregmaLevelTextField.setText(""+
myProjectData.bregmaLevels[subject][section]);
3915
3916         section_rightSidePanel.updateBregmaLevel();
3917
3918         myMethods.setWaitCursor();
3919         this.generateImages();
3920         myMethods.setCustomCursor();
3921
3922         middleScrollPane.getViewport().add(section_middlePanel);
3923         rightScrollPane.getViewport().add(section_rightSidePanel);
3924
3925         repaint();
3926
3927     }
3928     public void whenLeftInTree(int currentSubject, int currentSection)
3929     {
3930
3931     }
3932
3933     public void generateImages()
3934     {

```

```

3935     String fileName;
3936     File file;
3937
3938     try
3939     {
3940         fileName = myMethods.getOriginalImageFileName(subject, section);
3941         file = new File(fileName);
3942         originalImage=ImageIO.read(file);
3943
3944         fileName = myMethods.getOriginalImageFileName(subject, section);
3945         file = new File(fileName);
3946         adjustedImage=ImageIO.read(file);
3947
3948         myMethods.changePixelsInImage(adjustedImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
3949         myMethods.changePixelsInImage(adjustedImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
3950
3951
3952         areasImage=ImageIO.read(file);
3953
3954         imageWidth=originalImage.getWidth();
3955         imageHeigth=originalImage.getHeight();
3956
3957     } catch (IOException event) { }
3958
3959     for (int i=0;i<myProjectData.numberOfUDA;i++)
3960     {
3961         UDAIMages[i]= new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
3962         try
3963         {
3964             fileName = myMethods.getOriginalImageFileName(subject, section);
3965             file = new File(fileName);
3966             UDAIMages[i]=ImageIO.read(file);
3967
3968         } catch (IOException event) { }
3969     }
3970
3971     boolean[][] tissuePixels = myMethods.getTissuePixels(areasImage);
3972     boolean[][] largestObjectPixels = myMethods.newGetLargestTissueObject(tissuePixels);
3973     boolean[][] outlinePixels = myMethods.findOutline(largestObjectPixels, imageWidth, imageHeigth);
3974     Point startPosition = myMethods.findStartPositionFromOutline(outlinePixels);
3975     boolean[][] insidePixels = myMethods.newFillInside(outlinePixels, startPosition);
3976     boolean[][] ventriclePixels = myMethods.getVentriclePixels(tissuePixels, insidePixels);
3977     boolean[][] isLeft = myMethods.isLeftFromArray(currentSectionData.lRDividePixels);
3978
3979     WritableRaster destinationRaster=areasImage.getRaster();
3980
3981     int[] black = {0,0,0};
3982     int[] red = {255,0,0};
3983     int[] blue = {0,0,255};
3984     int[] green = {0,255,0};
3985
3986     for (int i=0; i<imageWidth;i++)
3987         for (int j=0; j<imageHeigth;j++)
3988     {

```



```

3989
3990         if (insidePixels[i][j] && tissuePixels[i][j] && isLeft[i][j])
3991         {
3992             destinationRaster.setPixel(i, j, red);
3993         }
3994         if (insidePixels[i][j] && tissuePixels[i][j] && !isLeft[i][j])
3995         {
3996             destinationRaster.setPixel(i, j, black);
3997         }
3998         if (ventriclePixels[i][j] && isLeft[i][j])
3999         {
4000             destinationRaster.setPixel(i, j, blue);
4001         }
4002         if (ventriclePixels[i][j] && !isLeft[i][j])
4003         {
4004             destinationRaster.setPixel(i, j, green);
4005         }
4006     }
4007
4008     for (int UDAnumber=0; UDAnumber<myProjectData.numberOfUDA; UDAnumber++)
4009     {
4010         myMethods.changePixelsInImage(UDAIMages[UDAnumber],
currentSectionData.markedAsBackground, myProjectData.addBackgroundColor);
4011         myMethods.changePixelsInImage(UDAIMages[UDAnumber],
currentSectionData.markedAsTissue, myProjectData.addTissueColor);
4012
4013
4014         boolean[][] UDAoutlinePixels =
myMethods.widenLine(currentSectionData.UDALinePixels[UDAnumber], myProjectData.lineWidth);
4015         myMethods.changePixelsInImage(UDAIMages[UDAnumber], UDAoutlinePixels, black );
4016
4017         if (currentSectionData.haveUdaOrigin[UDAnumber])
4018         {
4019             Point UDASTartPosition = currentSectionData.udaOrigin[UDAnumber];
4020
4021             for (int i=0; i<imageWidth;i++)
4022             {
4023                 currentSectionData.UDALinePixels[UDAnumber][i][0]=true;
4024                 currentSectionData.UDALinePixels[UDAnumber][i][imageHeight-1]=true;
4025             }
4026             for (int i=0; i<imageHeight;i++)
4027             {
4028                 currentSectionData.UDALinePixels[UDAnumber][0][i]=true;
4029                 currentSectionData.UDALinePixels[UDAnumber][imageWidth-1][i]=true;
4030             }
4031
4032             boolean[][] udaPixels =
myMethods.newFillInside(currentSectionData.UDALinePixels[UDAnumber], UDASTartPosition);
4033
4034             boolean includeTissue = myProjectData.includeTissue[UDAnumber];
4035             boolean includeBackground = myProjectData.includeBackground[UDAnumber];
4036
4037             for (int i=0; i<imageWidth;i++)
4038                 for (int j=0; j<imageHeight;j++)
4039                 {
4040                     if (tissuePixels[i][j] && !includeTissue) udaPixels[i][j]=false;

```

```

4041         if (!tissuePixels[i][j] && !includeBackground) udaPixels[i][j]=false;
4042     }
4043
4044     myMethods.changePixelsInImage(UDAIMages[UDANumber], udaPixels, green );
4045 }
4046 }
4047 }
4048
4049 @Override
4050 public void paint(Graphics g)
4051 {
4052     super.paint(g);
4053
4054     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4055     int drawHeigth = imageHeigth*myProjectData.zoomFactor/100;
4056     FontMetrics fontMetrics = g.getFontMetrics();
4057     int fontHeight = fontMetrics.getHeight();
4058
4059     section_middlePanel.setPreferredSize(new Dimension(drawWidth+300,
(3+myProjectData.numberOfUDA)*drawHeigth));
4060
4061     g.drawImage(originalImage, 0, 0, drawWidth, drawHeigth, this);
4062     g.drawString("Original image", drawWidth+5, fontHeight);
4063     g.drawImage(adjustedImage, 0, drawHeigth, drawWidth, drawHeigth, this);
4064     g.drawString("Adjusted image", drawWidth+5, drawHeigth+fontHeight);
4065     g.drawImage(areasImage, 0, 2*drawHeigth, drawWidth, drawHeigth, this);
4066     g.drawString("Detected areas", drawWidth+5, 2*drawHeigth+fontHeight);
4067
4068     for (int UDANumber=0; UDANumber<myProjectData.numberOfUDA; UDANumber++)
4069     {
4070         g.drawImage(UDAIMages[UDANumber], 0, (3+UDANumber)*drawHeigth, drawWidth,
drawHeigth, this);
4071         g.drawString(""+myProjectData.UDANames[UDANumber], drawWidth+5,
(3+UDANumber)*drawHeigth+fontHeight);
4072     }
4073
4074     revalidate();
4075 }
4076 }
4077
4078 private class Section_RightSidePanel extends JPanel implements ActionListener
4079 {
4080     //The level used for this section can be changed here. If the levels
4081     //in a subject are not continous a warning will be generated here and
4082     //when the subject results and the section results are displayed.
4083
4084     JLabel infoLabel = new JLabel("Current level");
4085     JLabel levelLabel = new JLabel("");
4086     JButton updateBregmaLevelButton;
4087     JTextField bregmaLevelTextField = new JTextField("",8);
4088     JLabel warningLabel= new JLabel("");
4089
4090     public Section_RightSidePanel()
4091     {
4092         updateBregmaLevelButton=new JButton("Update Bregma Level");
4093         updateBregmaLevelButton.addActionListener(this);

```

```

4094     bregmaLevelTextField.addActionListener(this);
4095
4096     this.setLayout(new GridBagLayout());
4097     GridBagConstraints c = new GridBagConstraints();
4098
4099     c.gridx = 0;
4100     c.gridy = 0;
4101     c.gridwidth = 1;
4102     c.insets = new Insets(10,5,5,5);
4103     c.anchor = GridBagConstraints.PAGE_START;
4104     this.add(infoLabel, c);
4105
4106     c.gridy++;
4107     c.insets = new Insets(5,5,5,5);
4108     this.add(levelLabel, c);
4109
4110     c.gridy++;
4111     c.insets = new Insets(5,5,5,5);
4112     this.add(updateBregmaLevelButton, c);
4113
4114     c.gridy++;
4115     c.insets = new Insets(5,5,5,5);
4116     this.add(bregmaLevelTextField, c);
4117
4118     c.gridy++;
4119     c.weighty=1;
4120     this.add(warningLabel, c);
4121 }
4122
4123 public void updateBregmaLevel()
4124 {
4125     boolean isNumber=false;
4126
4127     String bregmaLevel = bregmaLevelTextField.getText();
4128     try
4129     {
4130         myProjectData.bregmaLevels[section_middlePanel.subject][section_middlePanel.section] =
4131         Double.parseDouble(bregmaLevel);
4132         isNumber=true;
4133         warningLabel.setText("");
4134         levelLabel.setText(""+
4135 myProjectData.bregmaLevels[section_middlePanel.subject][section_middlePanel.section]);
4136     }catch(Exception error){ }
4137
4138     if(!isNumber)
4139     {
4140         warningLabel.setForeground(Color.red);
4141         warningLabel.setText("Not a number");
4142     }
4143     else
4144     {
4145         boolean allIncreasing=true;
4146         boolean allDecreasing=true;
4147
4148         int subject= section_middlePanel.subject;
4149         int section= section_middlePanel.section;

```

```

4148
4149         for(int i=0;i<myProjectData.numberOfLevels-1;i++)
4150         {
4151             if (myProjectData.sectionExists[subject][i+1])
4152             {
4153                 if (myProjectData.bregmaLevels[subject][i]>=myProjectData.bregmaLevels[subject][i+1])
allIncreasing=false;
4154                 if (myProjectData.bregmaLevels[subject][i]<=myProjectData.bregmaLevels[subject][i+1])
allDecreasing=false;
4155             }
4156         }
4157
4158         if (!(allIncreasing || allDecreasing))
4159         {
4160             warningLabel.setForeground(Color.red);
4161             warningLabel.setText("Values must continuously increase or decrease");
4162             myProjectData.continousLevels[subject]=false;
4163         }
4164         else
4165         {
4166             warningLabel.setForeground(Color.red);
4167             warningLabel.setText("");
4168             myProjectData.continousLevels[subject]=true;
4169         }
4170     }
4171
4172     myProjectData.allLevelsContinous=true;
4173     for (int i=0; i<myProjectData.numberOfSubjects;i++)
4174     {
4175         if (!myProjectData.continousLevels[i]) myProjectData.allLevelsContinous=false;
4176     }
4177
4178     myMethods.analyzeSubject(section_middlePanel.subject);
4179 }
4180
4181 public void actionPerformed(ActionEvent e)
4182 {
4183     if (e.getSource()==updateBregmaLevelButton)
4184     {
4185         this.updateBregmaLevel();
4186     }
4187     if (e.getSource()==bregmaLevelTextField)
4188     {
4189         upperScrollPane.requestFocus();
4190         this.updateBregmaLevel();
4191     }
4192 }
4193 }
4194
4195 private class AdjustImage_MiddlePanel extends JPanel implements MouseListener, MouseMotionListener
4196 {
4197     //Here the user can remove unwanted tissue and add tissue to fill for example tears.
4198     //It is also possible to change the color treshold and intensity treshold
4199     //for this section only.
4200     int subject=0;
4201     int section=0;

```

```

4202
4203     boolean addBackground=true;
4204
4205     int size = 10;
4206     int circleDrawSize=10;
4207     int[] drawColor = {255,255,255};
4208     Cursor transparentCursor = new Cursor(Cursor.HAND_CURSOR);
4209     boolean cursorOverPanel = false;
4210
4211     Point currentPoint = new Point(100,100);
4212
4213     int circlePosX=100;
4214     int circlePosY=100;
4215     int circleSize=20;
4216
4217     int cursorPosX=100;
4218     int cursorPosY=100;
4219
4220     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
4221     BufferedImage imageToAdjust = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
4222     BufferedImage modifiedImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
4223
4224     public AdjustImage_MiddlePanel()
4225     {
4226         addMouseListener(this);
4227         addMouseMotionListener(this);
4228         this.setTransparentCursor();
4229     }
4230
4231     public void mouseClicked(MouseEvent event)
4232     {
4233         if (event.getButton() == MouseEvent.BUTTON1)
4234         {
4235             //Adds or remove tissue in a circle with radius size.
4236
4237             int imageWidth = originalImage.getWidth();
4238             int imageHeight = originalImage.getHeight();
4239             int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4240             int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4241
4242
4243
4244
4245
4246
4247             currentPoint=event.getPoint();
4248             currentPoint.y = currentPoint.y-drawHeight;
4249             currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4250             currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4251
4252             if (currentPoint.x<imageWidth && currentPoint.x>-1
4253                 && currentPoint.y<imageHeight && currentPoint.y>-1)
4254                 this.newDrawPoint(currentPoint);
4255
4256             currentPoint=event.getPoint();
4257             currentPoint.y = currentPoint.y-2*drawHeight;

```

```

4258     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4259     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4260
4261     if (currentPoint.x<imageWidth && currentPoint.x>-1
4262         && currentPoint.y<imageHeight && currentPoint.y>-1)
4263         this.newDrawPoint(currentPoint);
4264
4265
4266
4267
4268
4269
4270
4271
4272     repaint();
4273
4274     currentPoint=event.getPoint();
4275     cursorPosX=currentPoint.x;
4276     cursorPosY=currentPoint.y;
4277
4278     circleDrawSize=size*myProjectData.zoomFactor/100;
4279
4280     circlePosX=cursorPosX-circleDrawSize;
4281     circlePosY=cursorPosY-circleDrawSize;
4282     circleSize=circleDrawSize*2;
4283 }
4284 if (event.getButton() == MouseEvent.BUTTON3)
4285 {
4286     //Recalculates the result image.
4287
4288     adjustImage_middlePanel.generateAdjustedImage();
4289     adjustImage_middlePanel.generateResultImage();
4290     adjustImage_middlePanel.repaint();
4291 }
4292 }
4293 public void mouseDragged(MouseEvent event)
4294 {
4295     //Modifies image in the same way as if a mouseClicked was triggered.
4296
4297     currentPoint=event.getPoint();
4298
4299     int imageWidth = originalImage.getWidth();
4300     int imageHeight = originalImage.getHeight();
4301     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4302     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4303
4304     currentPoint.y = currentPoint.y-drawHeight;
4305     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4306
4307     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4308
4309
4310     if (currentPoint.x<imageWidth && currentPoint.x>-1
4311         && currentPoint.y<imageHeight && currentPoint.y>-1)
4312         this.newDrawPoint(currentPoint);
4313

```

```

4314
4315
4316
4317     currentPoint=event.getPoint();
4318     currentPoint.y = currentPoint.y-2*drawHeight;
4319     currentPoint.y = currentPoint.y*imageHeight/drawHeight;
4320     currentPoint.x = currentPoint.x*imageWidth/drawWidth;
4321
4322
4323     if (currentPoint.x<imageWidth && currentPoint.x>-1
4324         && currentPoint.y<imageHeight && currentPoint.y>-1)
4325         this.newDrawPoint(currentPoint);
4326
4327
4328     repaint();
4329
4330     currentPoint=event.getPoint();
4331     cursorPosX=currentPoint.x;
4332     cursorPosY=currentPoint.y;
4333
4334     circleDrawSize=size*myProjectData.zoomFactor/100;
4335
4336     circlePosX=cursorPosX-circleDrawSize;
4337     circlePosY=cursorPosY-circleDrawSize;
4338     circleSize=circleDrawSize*2;
4339 }
4340
4341 public void mouseMoved(MouseEvent event)
4342 {
4343     //Displays RGB values when mouse is over an image. Also displays
4344     //the values for color and intensity and whether the pixel
4345     //is considered to be tissue or background.
4346     int[] color = new int[3];
4347
4348     int imageWidth = originalImage.getWidth();
4349     int imageHeight = originalImage.getHeight();
4350
4351     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4352     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4353
4354     Point cursorPosition = event.getPoint();
4355     Point imagePosition = new Point();
4356
4357     middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
4358     cursorOverPanel=false;
4359     adjustImage_rightSidePanel.redValue.setText("Red: ---");
4360     adjustImage_rightSidePanel.greenValue.setText("Green: ---");
4361     adjustImage_rightSidePanel.blueValue.setText("Blue: ---");
4362     adjustImage_rightSidePanel.colorValueLabel.setText("Color: ---");
4363     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
4364     adjustImage_rightSidePanel.resultLabel.setText("Pixel is: ---");
4365
4366     if (
4367         cursorPosition.y<drawHeight &&
4368         cursorPosition.x<drawWidth )
4369     {

```

```

4370
4371     imagePosition.x=cursorPosition.x*100/myProjectData.zoomFactor;
4372     imagePosition.y=cursorPosition.y;
4373     imagePosition.y=imagePosition.y*100/myProjectData.zoomFactor;
4374
4375     WritableRaster raster = originalImage.getRaster();
4376     color=raster.getPixel(imagePosition.x,imagePosition.y,color);
4377     adjustImage_rightSidePanel.redValue.setText("Red: "+color[0]);
4378     adjustImage_rightSidePanel.greenValue.setText("Green: "+color[1]);
4379     adjustImage_rightSidePanel.blueValue.setText("Blue: "+color[2]);
4380
4381     int firstInt = color[0]+color[2];
4382     int secondInt = color[0]+color[1]+color[2];
4383     double tempDouble = (double) firstInt/secondInt;
4384     tempDouble = tempDouble*1000;
4385     tempDouble = Math.round(tempDouble);
4386     tempDouble = tempDouble/10;
4387
4388     adjustImage_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
4389     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
4390     if(myMethods.pixelIsTissue(color))
4391     {
4392         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
4393     }
4394     else
4395     {
4396         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Background");
4397     }
4398
4399     adjustImage_rightSidePanel.repaint();
4400 }
4401
4402 if (cursorPosition.y>drawHeight &&
4403     cursorPosition.y<2*drawHeight &&
4404     cursorPosition.x< drawWidth )
4405 {
4406     middleScrollPane.setCursor(transparentCursor);
4407     cursorOverPanel=true;
4408
4409     imagePosition.x=cursorPosition.x*100/myProjectData.zoomFactor;
4410     imagePosition.y=cursorPosition.y-drawHeight;
4411     imagePosition.y=imagePosition.y*100/myProjectData.zoomFactor;
4412
4413     WritableRaster raster = originalImage.getRaster();
4414     color=raster.getPixel(imagePosition.x,imagePosition.y,color);
4415
4416     adjustImage_rightSidePanel.redValue.setText("Red: "+color[0]);
4417     adjustImage_rightSidePanel.greenValue.setText("Green: "+color[1]);
4418     adjustImage_rightSidePanel.blueValue.setText("Blue: "+color[2]);
4419
4420     int firstInt = color[0]+color[2];
4421     int secondInt = color[0]+color[1]+color[2];
4422     double tempDouble = (double) firstInt/secondInt;
4423     tempDouble = tempDouble*1000;
4424     tempDouble = Math.round(tempDouble);
4425     tempDouble = tempDouble/10;

```



```

4426
4427     adjustImage_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
4428     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
4429     if(myMethods.pixellIsTissue(color))
4430     {
4431         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
4432     }
4433     else
4434     {
4435         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Background");
4436     }
4437
4438     adjustImage_rightSidePanel.repaint();
4439 }
4440
4441 if (cursorPosition.y>2*drawHeight &&
4442     cursorPosition.y<3*drawHeight &&
4443     cursorPosition.x< drawWidth )
4444 {
4445     middleScrollPane.setCursor(transparentCursor);
4446     cursorOverPanel=true;
4447
4448     imagePosition.x=cursorPosition.x*100/myProjectData.zoomFactor;
4449     imagePosition.y=cursorPosition.y-2*drawHeight;
4450     imagePosition.y=imagePosition.y*100/myProjectData.zoomFactor;
4451
4452     WritableRaster raster = originalImage.getRaster();
4453     color=raster.getPixel(imagePosition.x,imagePosition.y,color);
4454
4455     adjustImage_rightSidePanel.redValue.setText("Red: "+color[0]);
4456     adjustImage_rightSidePanel.greenValue.setText("Green: "+color[1]);
4457     adjustImage_rightSidePanel.blueValue.setText("Blue: "+color[2]);
4458
4459     int firstInt = color[0]+color[2];
4460     int secondInt = color[0]+color[1]+color[2];
4461     double tempDouble = (double) firstInt/secondInt;
4462     tempDouble = tempDouble*1000;
4463     tempDouble = Math.round(tempDouble);
4464     tempDouble = tempDouble/10;
4465
4466     adjustImage_rightSidePanel.colorValueLabel.setText("Color: "+tempDouble);
4467     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: "+secondInt);
4468     if(myMethods.pixellIsTissue(color))
4469     {
4470         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Tissue");
4471     }
4472     else
4473     {
4474         adjustImage_rightSidePanel.resultLabel.setText("Pixel is: Background");
4475     }
4476
4477     adjustImage_rightSidePanel.repaint();
4478 }
4479
4480 currentPoint=event.getPoint();
4481 cursorPosX=currentPoint.x;

```

```

4482     cursorPosY=currentPoint.y;
4483
4484     circleDrawSize=size*myProjectData.zoomFactor/100;
4485
4486     circlePosX=cursorPosX-circleDrawSize;
4487     circlePosY=cursorPosY-circleDrawSize;
4488     circleSize=circleDrawSize*2;
4489
4490     adjustImage_rightSidePanel.repaint();
4491 }
4492
4493 public void mouseEntered(MouseEvent event)
4494 {
4495     cursorOverPanel=true;
4496 }
4497 public void mouseExited(MouseEvent event)
4498 {
4499     cursorOverPanel=false;
4500
4501     adjustImage_rightSidePanel.redValue.setText("Red: ---");
4502     adjustImage_rightSidePanel.greenValue.setText("Green: ---");
4503     adjustImage_rightSidePanel.blueValue.setText("Blue: ---");
4504     adjustImage_rightSidePanel.colorValueLabel.setText("Color: ---");
4505     adjustImage_rightSidePanel.intensityValueLabel.setText("Intensity: ---");
4506     adjustImage_rightSidePanel.resultLabel.setText("Pixel is: ---");
4507 }
4508 public void mousePressed(MouseEvent event) {}
4509 public void mouseReleased(MouseEvent event) {}
4510
4511 public void whenClickedInTree(int currentSubject, int currentSection)
4512 {
4513     myMethods.setWaitCursor();
4514
4515     subject=currentSubject;
4516     section=currentSection;
4517
4518     currentSectionData=myMethods.readSectionData(subject, section);
4519
4520     adjustImage_rightSidePanel.updateTextFields();
4521     adjustImage_rightSidePanel.colorWarningLabel.setText("");
4522     adjustImage_rightSidePanel.intensityWarningLabel.setText("");
4523
4524     this.generateOriginalImage();
4525     this.generateAdjustedImage();
4526     this.generateResultImage();
4527
4528     middleScrollPane.getViewPort().add(adjustImage_middlePanel);
4529
4530     rightScrollPane.getViewPort().add(adjustImage_rightSidePanel);
4531     adjustImage_middlePanel.setPreferredSize(new Dimension(originalImage.getWidth(),
4532 3*originalImage.getHeight()+300));
4533
4534     myMethods.setCustomCursor();
4535 }
4536 public void whenLeftInTree()
4537 {

```



```

4593         if(addBackground)
4594         {
4595             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y-j]=true;
4596             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y-j]=false;
4597             adjustRaster.setPixel(newPoint.x+i,newPoint.y-j, myProjectData.addBackgroundColor);
4598             resultRaster.setPixel(newPoint.x+i,newPoint.y-j, myProjectData.addBackgroundColor);
4599         }
4600         else
4601         {
4602             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y-j]=false;
4603             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y-j]=true;
4604             adjustRaster.setPixel(newPoint.x+i,newPoint.y-j, myProjectData.addTissueColor);
4605             resultRaster.setPixel(newPoint.x+i,newPoint.y-j, myProjectData.addTissueColor);
4606         }
4607     } catch(Exception error){ }
4608
4609     try
4610     {
4611         if(addBackground)
4612         {
4613             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y+j]=true;
4614             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y+j]=false;
4615             adjustRaster.setPixel(newPoint.x+i,newPoint.y+j, myProjectData.addBackgroundColor);
4616             resultRaster.setPixel(newPoint.x+i,newPoint.y+j, myProjectData.addBackgroundColor);
4617         }
4618         else
4619         {
4620             currentSectionData.markedAsBackground[newPoint.x+i][newPoint.y+j]=false;
4621             currentSectionData.markedAsTissue[newPoint.x+i][newPoint.y+j]=true;
4622             adjustRaster.setPixel(newPoint.x+i,newPoint.y+j, myProjectData.addTissueColor);
4623             resultRaster.setPixel(newPoint.x+i,newPoint.y+j, myProjectData.addTissueColor);
4624         }
4625     } catch(Exception error){ }
4626     }
4627 }
4628 }
4629
4630 public void generateOriginalImage()
4631 {
4632     try
4633     {
4634         File originalImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
4635         originalImage = ImageIO.read(originalImageFile);
4636     } catch (IOException event) { }
4637 }
4638
4639 public void generateAdjustedImage()
4640 {
4641     try
4642     {
4643         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
4644         imageToAdjust = ImageIO.read(adjustedImageFile);
4645     } catch (IOException event) { }
4646
4647     myMethods.changePixelsInImage(imageToAdjust, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);

```

```

4648     myMethods.changePixelsInImage(imageToAdjust, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
4649 }
4650
4651 public void generateResultImage()
4652 {
4653     try
4654     {
4655         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
4656         modifiedImage = ImageIO.read(adjustedImageFile);
4657     } catch (IOException event) { }
4658
4659     boolean[][] tissuePixels = myMethods.getTissuePixels(modifiedImage);
4660
4661     WritableRaster destinationRaster=modifiedImage.getRaster();
4662
4663     int[] red = {255,0,0};
4664
4665     for (int i=0; i<modifiedImage.getWidth();i++)
4666         for (int j=0; j<modifiedImage.getHeight();j++)
4667         {
4668             if (tissuePixels[i][j])
4669             {
4670                 destinationRaster.setPixel(i, j, red);
4671             }
4672         }
4673 }
4674
4675 @Override
4676 public void paint(Graphics g)
4677 {
4678     super.paint(g);
4679
4680     int imageWidth = originalImage.getWidth();
4681     int imageHeight = originalImage.getHeight();
4682
4683     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
4684     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
4685
4686     adjustImage_middlePanel.setPreferredSize(new Dimension(drawWidth+300, 3*drawHeight));
4687
4688     g.drawString("Original image", drawWidth+10, 15);
4689     g.drawImage(originalImage, 0, 0, drawWidth, drawHeight, this);
4690
4691     g.drawString("Adjust this image", drawWidth+10, 15+drawHeight);
4692     g.drawImage(imageToAdjust, 0, drawHeight, drawWidth, drawHeight, this);
4693
4694     g.drawString("Detected tissue pixels", drawWidth+10, 15+2*drawHeight);
4695     g.drawImage(modifiedImage, 0, 2*drawHeight, drawWidth, drawHeight, this);
4696
4697     if (cursorOverPanel)
4698     {
4699         Graphics2D g2 = (Graphics2D) g;
4700         Color tempDrawColor = new Color (drawColor[0],drawColor[1],drawColor[2]);
4701         g2.setColor(tempDrawColor);
4702         g2.draw(new Ellipse2D.Double(circlePosX, circlePosY,

```

```

4703         circleSize, circleSize));
4704     g2.setColor(Color.BLACK);
4705     g2.draw(new Ellipse2D.Double(circlePosX-1, circlePosY-1,
4706         circleSize+2, circleSize+2));
4707 }
4708
4709 repaint();
4710 revalidate();
4711 }
4712
4713 public void setTransparentCursor()
4714 {
4715     //Windows can't use a cursor larger than 32x32 pixels.
4716     //To get around this problem this method creates a transparent cursor.
4717     //Whenever the cursor is over an image the transparent cursor is used
4718     //and a circle is drawn at the cursor position.
4719     Toolkit toolkit = Toolkit.getDefaultToolkit();
4720     BufferedImage bufferedImage = new BufferedImage(32,32,BufferedImage.TYPE_INT_ARGB);
4721     WritableRaster raster = bufferedImage.getRaster();
4722     int[] transparent = {0,0,0,0};
4723
4724     for (int i=0;i<32;i++)
4725         for (int j=0;j<32;j++)
4726             raster.setPixel(i,j,transparent);
4727
4728     Image tempImage = bufferedImage;
4729     Point hotSpot = new Point();
4730     hotSpot.x=0;
4731     hotSpot.y=0;
4732     transparentCursor = toolkit.createCustomCursor(tempImage , hotSpot, "Circel");
4733 }
4734
4735 }
4736
4737 public class AdjustImage_RightSidePanel extends JPanel implements ActionListener
4738 {
4739     JButton helpButton;
4740     JButton revertToOriginalButton;
4741     JButton updateImageButton;
4742     JButton increaseButton;
4743     JButton decreaseButton;
4744     JButton addTissueButton;
4745     JButton removeTissueButton;
4746
4747     JLabel colorTresholdLabel = new JLabel("Color treshold");
4748     JTextField colorTresholdTextField = new JTextField(""+myProjectData.colorTresholdGeneral, 8);
4749     JLabel colorWarningLabel = new JLabel("");
4750     JLabel intensityTresholdLabel = new JLabel("Intensity treshold");
4751     JTextField intensityTresholdTextField = new JTextField(""+myProjectData.intensityTresholdGeneral, 8);
4752     JLabel intensityWarningLabel = new JLabel("");
4753
4754     int size=10;
4755     int[] white = {255,255,255};
4756     int[] tissueColor = {255,0,255};
4757
4758     JLabel redValue = new JLabel("Red: ---");

```

```

4759 JLabel greenValue = new JLabel("Green: ---");
4760 JLabel blueValue = new JLabel("Blue: ---");
4761
4762 JLabel colorValueLabel = new JLabel("Color: ");
4763 JLabel intensityValueLabel = new JLabel("Intensity: ");
4764 JLabel resultLabel = new JLabel("Pixel is: ");
4765
4766 JLabel testLabel = new JLabel("");
4767
4768 public AdjustImage_RightSidePanel()
4769 {
4770     helpButton = new JButton ("Help");
4771     helpButton.addActionListener(this);
4772
4773     revertToOriginalButton = new JButton ("Revert To Original");
4774     revertToOriginalButton.addActionListener(this);
4775
4776     updateImageButton = new JButton("Update Image");
4777     updateImageButton.addActionListener(this);
4778
4779     decreaseButton = new JButton("- Draw Size");
4780     decreaseButton.addActionListener(this);
4781
4782     increaseButton = new JButton("+ Draw Size");
4783     increaseButton.addActionListener(this);
4784
4785     addTissueButton = new JButton("Add tissue");
4786     addTissueButton.addActionListener(this);
4787
4788     removeTissueButton = new JButton("Remove tissue");
4789     removeTissueButton.addActionListener(this);
4790
4791     colorTresholdTextField.addActionListener(this);
4792     intensityTresholdTextField.addActionListener(this);
4793
4794     this.setLayout(new GridBagLayout());
4795     GridBagConstraints c = new GridBagConstraints();
4796
4797     c.gridx = 0;
4798     c.gridy = 0;
4799     c.gridwidth = 1;
4800     c.insets = new Insets(10,5,20,5);
4801     c.anchor = GridBagConstraints.PAGE_START;
4802     this.add(helpButton , c);
4803
4804     c.insets = new Insets(5,5,5,5);
4805     c.gridy ++;
4806     this.add(revertToOriginalButton , c);
4807
4808     c.gridy ++;
4809     this.add(updateImageButton, c);
4810
4811     c.insets = new Insets(20,5,5,5);
4812     c.gridy ++;
4813     this.add(decreaseButton, c);
4814

```

```
4815     c.insets = new Insets(5,5,5,5);
4816     c.gridy ++;
4817     this.add(increaseButton, c);
4818
4819     c.gridy ++;
4820     this.add(addTissueButton, c);
4821
4822     c.gridy ++;
4823     this.add(removeTissueButton, c);
4824
4825     c.gridy++;
4826     c.insets = new Insets(25,5,5,5);
4827     this.add(colorTresholdLabel, c);
4828
4829     c.gridy++;
4830     c.insets = new Insets(5,5,5,5);
4831     this.add(colorTresholdTextField, c);
4832
4833     c.gridy++;
4834     this.add(colorWarningLabel, c);
4835
4836     c.gridy++;
4837     this.add(intensityTresholdLabel, c);
4838
4839     c.insets = new Insets(5,5,25,5);
4840     c.gridy++;
4841     this.add(intensityTresholdTextField, c);
4842
4843     c.gridy++;
4844     c.insets = new Insets(5,5,5,5);
4845     this.add(intensityWarningLabel, c);
4846
4847     c.gridy++;
4848     c.insets = new Insets(0,5,15,5);
4849     this.add(redValue, c);
4850
4851     c.gridy++;
4852     this.add(greenValue, c);
4853
4854     c.gridy++;
4855     this.add(blueValue, c);
4856
4857     c.gridy++;
4858     c.insets = new Insets(25,5,5,5);
4859     this.add(colorValueLabel,c);
4860
4861     c.gridy++;
4862     c.insets = new Insets(5,5,5,5);
4863     this.add(intensityValueLabel,c);
4864
4865     c.gridy++;
4866     c.weighty = 1;
4867     this.add(resultLabel, c);
4868 }
4869
4870 public void actionPerformed(ActionEvent e)
```



```

4871 {
4872     if (e.getSource() == helpButton)
4873     {
4874         String helpText = myMethods.getHelpTextAdjustImage();
4875         helpDialog.newText(helpText);
4876     }
4877
4878     if (e.getSource() == revertToOriginalButton)
4879     {
4880         for (int i=0; i<myProjectData.imageWidth;i++)
4881             for (int j=0; j<myProjectData.imageHeight; j++)
4882             {
4883                 currentSectionData.markedAsBackground[i][j]=false;
4884                 currentSectionData.markedAsTissue[i][j]=false;
4885             }
4886
4887         adjustImage_middlePanel.generateOriginalImage();
4888         adjustImage_middlePanel.generateAdjustedImage();
4889         adjustImage_middlePanel.generateResultImage();
4890         adjustImage_middlePanel.repaint();
4891     }
4892
4893     if (e.getSource() == updateImageButton)
4894     {
4895         adjustImage_middlePanel.generateAdjustedImage();
4896         adjustImage_middlePanel.generateResultImage();
4897         adjustImage_middlePanel.repaint();
4898     }
4899
4900     if (e.getSource() == decreaseButton)
4901     {
4902         size=adjustImage_middlePanel.size;
4903
4904         if (size<5)
4905         {
4906             size -=1;
4907         }
4908         else if (size<12)
4909         {
4910             size -= 3;
4911         }
4912         else
4913         {
4914             size -= 6;
4915         }
4916
4917         if (size<1) size=1;
4918         adjustImage_middlePanel.size =size;
4919     }
4920
4921     if (e.getSource() == increaseButton)
4922     {
4923         size=adjustImage_middlePanel.size;
4924
4925         if (size<5)
4926         {

```

```

4927         size +=1;
4928     }
4929     else if (size<12)
4930     {
4931         size += 3;
4932     }
4933     else
4934     {
4935         size += 6;
4936     }
4937
4938     adjustImage_middlePanel.size =size;
4939 }
4940
4941 if (e.getSource() == addTissueButton)
4942 {
4943     adjustImage_middlePanel.drawColor =tissueColor;
4944     adjustImage_middlePanel.addBackground=false;
4945 }
4946 if (e.getSource() == removeTissueButton)
4947 {
4948     adjustImage_middlePanel.drawColor = white;
4949     adjustImage_middlePanel.addBackground=true;
4950 }
4951
4952 if (e.getSource()==colorTresholdTextField)
4953 {
4954     upperScrollPane.requestFocus();
4955
4956     try
4957     {
4958         double tempDouble = Double.parseDouble(colorTresholdTextField.getText());
4959         if (tempDouble>=0 && tempDouble<=100)
4960         {
4961             currentSectionData.colorTreshold = tempDouble;
4962             colorWarningLabel.setText("");
4963         }
4964         else
4965         {
4966             colorWarningLabel.setForeground(Color.red);
4967             colorWarningLabel.setText("Use range 0-100");
4968         }
4969     }catch(Exception error)
4970     {
4971         colorWarningLabel.setForeground(Color.red);
4972         colorWarningLabel.setText("Not a number");
4973     }
4974
4975     adjustImage_middlePanel.generateResultImage();
4976     adjustImage_middlePanel.repaint();
4977 }
4978 if (e.getSource()==intensityTresholdTextField)
4979 {
4980     upperScrollPane.requestFocus();
4981
4982     try

```

```

4983     {
4984         double tempDouble = Double.parseDouble(intensityTresholdTextField.getText());
4985         if (tempDouble>=0 && tempDouble<=765)
4986         {
4987             currentSectionData.intensityTreshold = tempDouble;
4988             intensityWarningLabel.setText("");
4989         }
4990         else
4991         {
4992             intensityWarningLabel.setForeground(Color.red);
4993             intensityWarningLabel.setText("Use range 0-765");
4994         }
4995     } catch (Exception error)
4996     {
4997         intensityWarningLabel.setForeground(Color.red);
4998         intensityWarningLabel.setText("Not a number");
4999     }
5000
5001     adjustImage_middlePanel.generateResultImage();
5002     adjustImage_middlePanel.repaint();
5003 }
5004 }
5005
5006 public void updateTextFields()
5007 {
5008     colorTresholdTextField.setText(""+currentSectionData.colorTreshold);
5009     intensityTresholdTextField.setText(""+currentSectionData.intensityTreshold);
5010 }
5011 }
5012
5013 private class LR_DivideMiddlePanel extends JPanel implements MouseListener, MouseMotionListener
5014 {
5015     //Lets the user draw a line which separates the left and right hemispheres
5016     int subject=0;
5017     int section=0;
5018
5019     BufferedImage LR_LinesImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5020     BufferedImage LR_ResultsImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5021
5022     public LR_DivideMiddlePanel()
5023     {
5024         addMouseListener(this);
5025         addMouseMotionListener(this);
5026     }
5027
5028     public void mouseClicked(MouseEvent event)
5029     {
5030         if (event.getButton() == MouseEvent.BUTTON1)
5031         {
5032             this.newDrawPoint(event.getPoint());
5033         }
5034         if (event.getButton() == MouseEvent.BUTTON3)
5035         {
5036             lr_divideMiddlePanel.generateLR_LinesImage();
5037             lr_divideMiddlePanel.generateLR_AreasImage();
5038             lr_divideMiddlePanel.repaint();

```

```

5039     }
5040 }
5041
5042 public void mouseDragged(MouseEvent event)
5043 {
5044     this.newDrawPoint(event.getPoint());
5045 }
5046
5047 public void mouseMoved(MouseEvent event) {}
5048 public void mousePressed(MouseEvent event) {}
5049 public void mouseReleased(MouseEvent event) {}
5050 public void mouseEntered(MouseEvent event) {}
5051 public void mouseExited(MouseEvent event) {}
5052
5053 public void whenClickedInTree(int currentSubject, int currentSection)
5054 {
5055     myMethods.setWaitCursor();
5056
5057     subject=currentSubject;
5058     section=currentSection;
5059
5060     currentSectionData=myMethods.readSectionData(subject, section);
5061     this.generateLR_LinesImage();
5062     this.generateLR_AreasImage();
5063     rightScrollPane.getViewPort().add(lr_DivideRightSidePanel);
5064     middleScrollPane.getViewPort().add(lr_divideMiddlePanel);
5065     lr_divideMiddlePanel.setPreferredSize(new Dimension(LR_LinesImage.getWidth(),
5066 2*LR_LinesImage.getHeight()+100));
5067     myMethods.setCustomCursor();
5068 }
5069 public void whenLeftInTree()
5070 {
5071     //Calculates areas and saves the data before leaving
5072     myMethods.setWaitCursor();
5073     this.generateLR_AreasImage();
5074     myMethods.saveSectionData(subject, section, currentSectionData);
5075     myMethods.updateSectionResultsObject(subject, section);
5076     myMethods.analyzeSubject(subject);
5077     myMethods.setCustomCursor();
5078 }
5079
5080 public void newDrawPoint(Point newPoint)
5081 {
5082     newPoint.x=100*newPoint.x/myProjectData.zoomFactor;
5083     newPoint.y=100*newPoint.y/myProjectData.zoomFactor;
5084
5085     if (newPoint.x<LR_LinesImage.getWidth() && newPoint.y<LR_LinesImage.getHeight()
5086         && newPoint.x>-1 && newPoint.y>-1
5087         && currentSectionData.lrHaveStartPoint)
5088     {
5089         currentSectionData.lrPreviousPoint=currentSectionData.lrCurrentPoint;
5090         currentSectionData.lrCurrentPoint= newPoint;
5091         currentSectionData.lrDividePixels = myMethods.drawLineBoolean
5092             (currentSectionData.lrDividePixels, currentSectionData.lrPreviousPoint,
currentSectionData.lrCurrentPoint);

```

```

5093
5094     boolean[][] linePixels = myMethods.widenLine(currentSectionData.lrDividePixels,
myProjectData.lineWidth);
5095     myMethods.changePixelsInImage(LR_LinesImage, linePixels, myProjectData.lineColor);
5096
5097     repaint();
5098     revalidate();
5099
5100 }
5101 else if (newPoint.x<LR_LinesImage.getWidth() && newPoint.y<LR_LinesImage.getHeight()
5102         && newPoint.x>-1 && newPoint.y>-1 )
5103 {
5104     currentSectionData.lrPreviousPoint=newPoint;
5105     currentSectionData.lrCurrentPoint=newPoint;
5106     currentSectionData.lrHaveStartPoint=true;
5107 }
5108 }
5109
5110 public void clearLines()
5111 {
5112     currentSectionData.lrHaveStartPoint=false;
5113     for (int i=0;i<LR_LinesImage.getWidth();i++)
5114         for (int j=0;j<LR_LinesImage.getHeight();j++)
5115             currentSectionData.lrDividePixels[i][j]=false;
5116
5117     lr_divideMiddlePanel.generateLR_LinesImage();
5118     lr_divideMiddlePanel.generateLR_AreasImage();
5119     repaint();
5120 }
5121
5122 public void generateLR_LinesImage()
5123 {
5124     try
5125     {
5126         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
5127         LR_LinesImage = ImageIO.read(adjustedImageFile);
5128     } catch (IOException event) { }
5129
5130     myMethods.changePixelsInImage(LR_LinesImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5131     myMethods.changePixelsInImage(LR_LinesImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5132
5133     boolean[][] linePixels = myMethods.widenLine(currentSectionData.lrDividePixels,
myProjectData.lineWidth);
5134     myMethods.changePixelsInImage(LR_LinesImage, linePixels, myProjectData.lineColor);
5135 }
5136
5137 public void generateLR_AreasImage()
5138 {
5139     try
5140     {
5141         File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
5142
5143         LR_ResultsImage = ImageIO.read(adjustedImageFile);
5144     } catch (IOException event) { }

```

```

5145
5146     boolean[][] tissuePixels = myMethods.getTissuePixels(LR_ResultsImage);
5147     boolean[][] largestObjectPixels = myMethods.newGetLargestTissueObject(tissuePixels);
5148     boolean[][] outlinePixels = myMethods.findOutline(largestObjectPixels, LR_ResultsImage.getWidth(),
LR_ResultsImage.getHeight());
5149     Point startPosition = myMethods.findStartPositionFromOutline(outlinePixels);
5150     boolean[][] insidePixels = myMethods.newFillInside(outlinePixels, startPosition);
5151     boolean[][] ventriclePixels = myMethods.getVentriclePixels(tissuePixels, insidePixels);
5152     boolean[][] isLeft = myMethods.isLeftFromArray(currentSectionData.lrDividePixels);
5153
5154     myMethods.changePixelsInImage(LR_ResultsImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5155     myMethods.changePixelsInImage(LR_ResultsImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5156
5157     WritableRaster destinationRaster=LR_ResultsImage.getRaster();
5158
5159     int tissuePixelsLeft=0;
5160     int tissuePixelsRight=0;
5161     int ventriclePixelsLeft=0;
5162     int ventriclePixelsRight=0;
5163
5164     int[] black = {255,255,0};
5165     int[] red = {255,0,0};
5166     int[] blue = {0,0,255};
5167     int[] green = {0,255,0};
5168
5169     for (int i=0; i<LR_ResultsImage.getWidth();i++)
5170         for (int j=0; j<LR_ResultsImage.getHeight();j++)
5171             {
5172                 if (tissuePixels[i][j] && isLeft[i][j])
5173                     {
5174                         destinationRaster.setPixel(i, j, red);
5175                         tissuePixelsLeft++;
5176                     }
5177                 if (tissuePixels[i][j] && !isLeft[i][j])
5178                     {
5179                         destinationRaster.setPixel(i, j, black);
5180                         tissuePixelsRight++;
5181                     }
5182                 if (ventriclePixels[i][j] && isLeft[i][j])
5183                     {
5184                         destinationRaster.setPixel(i, j, blue);
5185                         ventriclePixelsLeft++;
5186                     }
5187                 if (ventriclePixels[i][j] && !isLeft[i][j])
5188                     {
5189                         destinationRaster.setPixel(i, j, green);
5190                         ventriclePixelsRight++;
5191                     }
5192             }
5193
5194     myProjectData.numberOfTissuePixelsLeft[subject][section] = tissuePixelsLeft;
5195     myProjectData.numberOfTissuePixelsRight[subject][section] = tissuePixelsRight;
5196
5197     myProjectData.numberOfVentriclePixelsLeft[subject][section] = ventriclePixelsLeft;

```

```

5198     myProjectData.numberOfVentriclePixelsRight[subject][section] = ventriclePixelsRight;
5199 }
5200
5201 @Override
5202 public void paint(Graphics g)
5203 {
5204     super.paint(g);
5205
5206     int imageWidth = LR_LinesImage.getWidth();
5207     int imageHeight = LR_LinesImage.getHeight();
5208
5209     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
5210     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5211
5212     lr_divideMiddlePanel.setPreferredSize(new Dimension(drawWidth+300, 2*drawHeight));
5213
5214     g.drawString("Draw line here", drawWidth+10, 15);
5215     g.drawImage(LR_LinesImage, 0,0, drawWidth, drawHeight, this);
5216
5217     g.drawString("Detected areas", drawWidth+10, drawHeight+15);
5218     g.drawImage(LR_ResultsImage,0, drawHeight, drawWidth, drawHeight, this);
5219
5220     this.revalidate();
5221 }
5222 }
5223
5224 public class LR_DivideRightSidePanel extends JPanel implements ActionListener
5225 {
5226     JLabel info1Label = new JLabel("Left click or drag to draw outline");
5227     JLabel info1Label2 = new JLabel("Right click to update detected areas");
5228
5229     JButton clearLinesButton;
5230     JButton updateSectionButton;
5231
5232     public LR_DivideRightSidePanel()
5233     {
5234         clearLinesButton=new JButton("Clear Lines");
5235         clearLinesButton.addActionListener(this);
5236         updateSectionButton=new JButton("Update Section");
5237         updateSectionButton.addActionListener(this);
5238
5239         this.setLayout(new GridBagLayout());
5240         GridBagConstraints c = new GridBagConstraints();
5241
5242         c.gridx = 0;
5243         c.gridy = 0;
5244         c.gridwidth = 1;
5245         c.insets = new Insets(10,5,5,5);
5246         c.anchor = GridBagConstraints.PAGE_START;
5247         this.add(info1Label, c);
5248
5249         c.gridy++;
5250         c.insets = new Insets(5,5,5,5);
5251         this.add(info1Label2, c);
5252
5253         c.gridy++;

```

```

5254     c.insets = new Insets(20,5,5,5);
5255     this.add(clearLinesButton, c);
5256
5257     c.insets = new Insets(5,5,5,5);
5258     c.gridy++;
5259     c.weighty=1;
5260     this.add(updateSectionButton, c);
5261 }
5262
5263 public void actionPerformed(ActionEvent e)
5264 {
5265     if (e.getSource() == clearLinesButton)
5266     {
5267         lr_divideMiddlePanel.clearLines();
5268     }
5269
5270     if (e.getSource() == updateSectionButton)
5271     {
5272         lr_divideMiddlePanel.generateLR_LinesImage();
5273         lr_divideMiddlePanel.generateLR_AreasImage();
5274         lr_divideMiddlePanel.repaint();
5275     }
5276 }
5277 }
5278
5279 public class UDA_MiddlePanel extends JPanel implements MouseListener, MouseMotionListener
5280 {
5281     //Lets the user draw a line around an area. When the user right click the
5282     //size of the area is calculated.
5283     int subject=0;
5284     int section=0;
5285     int UDANumber=0;
5286
5287     BufferedImage originalImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5288     BufferedImage udaPanelImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5289
5290     public UDA_MiddlePanel()
5291     {
5292         addMouseListener(this);
5293         addMouseMotionListener(this);
5294     }
5295
5296     public void mouseClicked(MouseEvent event)
5297     {
5298         if (event.getButton() == MouseEvent.BUTTON1)
5299         {
5300             this.newDrawPoint(event.getPoint());
5301         }
5302
5303         if (event.getButton() == MouseEvent.BUTTON3)
5304         {
5305             int imageWidth = originalImage.getWidth();
5306             int imageHeight = originalImage.getHeight();
5307             int drawWidth = imageWidth*myProjectData.zoomFactor/100;
5308             int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5309

```



```

5310
5311     Point currentPoint=event.getPoint();
5312
5313     currentPoint.y=currentPoint.y-drawHeight;
5314     currentPoint.x=100*currentPoint.x/myProjectData.zoomFactor;
5315     currentPoint.y=100*currentPoint.y/myProjectData.zoomFactor;
5316
5317     if (currentPoint.x<udaPanelImage.getWidth() && currentPoint.y<udaPanelImage.getHeight()
5318         && currentPoint.x>-1 && currentPoint.y>-1
5319     )
5320     {
5321         currentSectionData.udaOrigin[UDANumber] = currentPoint;
5322         currentSectionData.haveUdaOrigin[UDANumber] = true;
5323         this.generateUdaPanelImage();
5324     }
5325 }
5326
5327
5328 public void mouseDragged(MouseEvent event)
5329 {
5330     this.newDrawPoint(event.getPoint());
5331 }
5332
5333 public void mouseMoved(MouseEvent event) {}
5334 public void mousePressed(MouseEvent event) {}
5335 public void mouseEntered(MouseEvent event) {}
5336 public void mouseExited(MouseEvent event) {}
5337 public void mouseReleased(MouseEvent event) {}
5338
5339 public void whenClickedInTree(int currentSubject, int currentSection, int currentUDANumber)
5340 {
5341     myMethods.setWaitCursor();
5342
5343     subject=currentSubject;
5344     section=currentSection;
5345     UDANumber=currentUDANumber;
5346
5347     currentSectionData=myMethods.readSectionData(subject, section);
5348
5349     this.generateUdaPanelImage();
5350     uda_middlePanel.setPreferredSize(new Dimension(udaPanelImage.getWidth(),
udaPanelImage.getHeight()));
5351     middleScrollPane.getViewport().add(uda_middlePanel);
5352     this.repaint();
5353
5354     rightScrollPane.getViewport().add(uda_rightSidePanel);
5355     myMethods.setCustomCursor();
5356 }
5357
5358 public void whenLeftInTree()
5359 {
5360     myMethods.setWaitCursor();
5361     myMethods.saveSectionData(subject, section, currentSectionData);
5362     myMethods.analyzeSubject(subject);
5363     myMethods.updateSectionResultsObject(subject, section);
5364     myMethods.setCustomCursor();

```

```

5365     }
5366
5367     public void newDrawPoint(Point newPoint)
5368     {
5369
5370         int imageWidth = originalImage.getWidth();
5371         int imageHeight = originalImage.getHeight();
5372         int drawWidth = imageWidth*myProjectData.zoomFactor/100;
5373         int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5374
5375         newPoint.y=newPoint.y-drawHeight;
5376         newPoint.x=100*newPoint.x/myProjectData.zoomFactor;
5377         newPoint.y=100*newPoint.y/myProjectData.zoomFactor;
5378
5379         if (newPoint.x<udaPanelImage.getWidth() && newPoint.y<udaPanelImage.getHeight()
5380             && newPoint.x>-1 && newPoint.y>-1
5381             && currentSectionData.UDAhaveStartPoint[UDAnumber])
5382         {
5383
5384             currentSectionData.UDApriorPoint[UDAnumber]=currentSectionData.UDAcurentPoint[UDAnumber];
5385             currentSectionData.UDAcurentPoint[UDAnumber]= newPoint;
5386             currentSectionData.UDALinePixels[UDAnumber] = myMethods.drawLineBoolean
5387                 (currentSectionData.UDALinePixels[UDAnumber],
5388                 currentSectionData.UDApriorPoint[UDAnumber],
5389                 currentSectionData.UDAcurentPoint[UDAnumber]);
5390
5391             boolean[][] outlinePixels =
myMethods.widenLine(currentSectionData.UDALinePixels[UDAnumber], myProjectData.lineWidth);
5392             myMethods.changePixelsInImage(udaPanelImage, outlinePixels, myProjectData.lineColor );
5393
5394             repaint();
5395
5396         }
5397         else if (newPoint.x<udaPanelImage.getWidth() && newPoint.y<udaPanelImage.getHeight()
5398             && newPoint.x>-1 && newPoint.y>-1 )
5399         {
5400             currentSectionData.UDApriorPoint[UDAnumber]=newPoint;
5401             currentSectionData.UDAcurentPoint[UDAnumber]=newPoint;
5402             currentSectionData.UDAhaveStartPoint[UDAnumber]=true;
5403         }
5404     }
5405 }
5406
5407
5408     public void clearLines()
5409     {
5410         for (int i=0;i<udaPanelImage.getWidth();i++)
5411             for (int j=0;j<udaPanelImage.getHeight();j++)
5412                 currentSectionData.UDALinePixels[UDAnumber][i][j]=false;
5413
5414         currentSectionData.UDAhaveStartPoint[UDAnumber]=false;
5415         currentSectionData.haveUdaOrigin[UDAnumber] = false;
5416         this.generateUdaPanelImage();
5417         currentSectionData.numberOfUdaPixels[UDAnumber] = 0;
5418     }

```

```

5419
5420     public void generateUdaPanelImage()
5421     {
5422         try
5423         {
5424             File adjustedImageFile= new File(myMethods.getOriginalImageFileName(subject, section));
5425             originalImage = ImageIO.read(adjustedImageFile);
5426             udaPanelImage = ImageIO.read(adjustedImageFile);
5427         } catch (IOException event) { }
5428
5429         myMethods.changePixelsInImage(originalImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5430         myMethods.changePixelsInImage(originalImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5431
5432         boolean[][] tissuePixels = myMethods.getTissuePixels(udaPanelImage);
5433
5434         myMethods.changePixelsInImage(udaPanelImage, currentSectionData.markedAsBackground,
myProjectData.addBackgroundColor);
5435         myMethods.changePixelsInImage(udaPanelImage, currentSectionData.markedAsTissue,
myProjectData.addTissueColor);
5436
5437         if (currentSectionData.haveUdaOrigin[UDAnumber])
5438         {
5439             Point startPosition = currentSectionData.udaOrigin[UDAnumber];
5440             boolean[][] udaPixels =
myMethods.newFillInside(currentSectionData.UDALinePixels[UDAnumber], startPosition);
5441
5442             boolean includeTissue = myProjectData.includeTissue[UDAnumber];
5443             boolean includeBackground = myProjectData.includeBackground[UDAnumber];
5444
5445             for (int i=0; i<udaPanelImage.getWidth();i++)
5446                 for (int j=0; j<udaPanelImage.getHeight();j++)
5447                 {
5448                     if (tissuePixels[i][j] && !includeTissue) udaPixels[i][j]=false;
5449                     if (!tissuePixels[i][j] && !includeBackground) udaPixels[i][j]=false;
5450                 }
5451
5452             int[] green = {0,255,0};
5453             myMethods.changePixelsInImage(udaPanelImage, udaPixels, green );
5454
5455             int numberOfUdaPixels=myMethods.countPixels(udaPixels);
5456             currentSectionData.numberOfUdaPixels[UDAnumber] = numberOfUdaPixels;
5457         }
5458
5459         boolean[][] outlinePixels = myMethods.widenLine(currentSectionData.UDALinePixels[UDAnumber],
myProjectData.lineWidth);
5460         myMethods.changePixelsInImage(udaPanelImage, outlinePixels, myProjectData.lineColor );
5461     }
5462
5463     @Override
5464     public void paint(Graphics g)
5465     {
5466         super.paint(g);
5467
5468         int imageWidth = udaPanelImage.getWidth();

```

```

5469     int imageHeight = udaPanelImage.getHeight();
5470
5471     int drawWidth = imageWidth*myProjectData.zoomFactor/100;
5472     int drawHeight = imageHeight*myProjectData.zoomFactor/100;
5473
5474     uda_middlePanel.setPreferredSize(new Dimension(drawWidth+300, 2*drawHeight));
5475
5476     g.drawString("Original image", drawWidth+10, 15);
5477     g.drawImage(originalImage,0,0, drawWidth, drawHeight, this);
5478
5479     g.drawString("Draw here", drawWidth+10, drawHeight+15);
5480     g.drawImage(udaPanelImage,0,drawHeight, drawWidth, drawHeight, this);
5481
5482     repaint();
5483     revalidate();
5484 }
5485 }
5486
5487 public class UDA_RightSidePanel extends JPanel implements ActionListener
5488 {
5489     JButton helpButton;
5490
5491     JLabel info1Label = new JLabel("Left click to draw outline");
5492     JLabel info2Label = new JLabel("Right click inside area");
5493     JLabel info3Label = new JLabel("when outline is complete");
5494
5495     JButton clearLinesButton;
5496
5497     public UDA_RightSidePanel()
5498     {
5499         helpButton=new JButton("Help");
5500         helpButton.addActionListener(this);
5501         clearLinesButton=new JButton("Clear Lines");
5502         clearLinesButton.addActionListener(this);
5503
5504         this.setLayout(new GridBagLayout());
5505         GridBagConstraints c = new GridBagConstraints();
5506
5507         c.gridx = 0;
5508         c.gridy = 0;
5509         c.gridwidth = 1;
5510         c.insets = new Insets(10,5,20,5);
5511         c.anchor = GridBagConstraints.PAGE_START;
5512         this.add(helpButton, c);
5513
5514         c.gridy++;
5515         c.insets = new Insets(10,5,5,5);
5516         this.add(info1Label, c);
5517
5518         c.gridy++;
5519         c.insets = new Insets(20,5,5,5);
5520         this.add(info2Label, c);
5521
5522         c.gridy++;
5523         c.insets = new Insets(5,5,5,5);
5524         this.add(info3Label, c);

```

```

5525
5526     c.insets = new Insets(20,5,5,5);
5527     c.gridy++;
5528     c.weighty=1;
5529     this.add(clearLinesButton, c);
5530 }
5531
5532 public void actionPerformed(ActionEvent e)
5533 {
5534     if (e.getSource() == helpButton)
5535     {
5536         String helpText = myMethods.getHelpTextUDA();
5537         helpDialog.newText(helpText);
5538     }
5539
5540     if (e.getSource() == clearLinesButton)
5541     {
5542         uda_middlePanel.clearLines();
5543     }
5544 }
5545 }
5546
5547 static public class ProjectData implements Serializable
5548 {
5549     //One object of this class is created and saved to disk.
5550     String projectLocation;
5551
5552     double conversionFactor=1;
5553     double conversionNumber=1;
5554     boolean conversionIsUpdated=false;
5555
5556     //Keeps track of whether the levels for a
5557     //subject are continuously increasing or decreasing
5558     boolean[] continuousLevels;
5559     boolean allLevelsContinuous=true;
5560
5561     int numberOfSubjects = 0;
5562     int totalNumberOfSections=0;
5563     int numberOfLevels=0;
5564     String[] subjectNameList;
5565
5566     boolean[][] sectionExists;
5567     String[][] sectionName;
5568
5569     double[][] bregmaLevels;
5570
5571     //Needed in the conversion panel
5572     Point firstConversionPoint = new Point(0,0);
5573     Point secondConversionPoint = new Point(0,0);
5574     int distancePixels=0;
5575     double distanceMM = 0;
5576
5577     int numberOfUDA;
5578     String[] UDAnames;
5579     boolean[] includeTissue;
5580     boolean[] includeBackground;

```

```

5581
5582     int[][] numberOfTissuePixels;
5583     int[][] numberOfTissuePixelsLeft;
5584     int[][] numberOfTissuePixelsRight;
5585
5586     int[][] numberOfVentriclePixels;
5587     int[][] numberOfVentriclePixelsLeft;
5588     int[][] numberOfVentriclePixelsRight;
5589
5590     double[] tissueVolume;
5591     double[] tissueVolumeLeft;
5592     double[] tissueVolumeRight;
5593
5594     double[] ventricleVolume;
5595     double[] ventricleVolumeLeft;
5596     double[] ventricleVolumeRight;
5597
5598     double[][] udaVolume;
5599
5600     int zoomFactor=100;
5601     int lineWidth=2;
5602     int[] lineColor = {0,0,0};
5603
5604     double colorTresholdGeneral=70;
5605     double intensityTresholdGeneral=50;
5606
5607     Object[][] sectionResultsObject;
5608     Object[][] roundedSectionResultsObject;
5609
5610     int imageWidth=1;
5611     int imageHeight=1;
5612
5613     int[] addTissueColor = {255,0,255};
5614     int[] addBackgroundColor = {255,255,255};
5615
5616     ProjectData(int newNumberOfSubject, int newNumberOfLevels, int newNumberOfUda, int
newTotalNumberOfSections)
5617     {
5618         numberOfSubjects = newNumberOfSubject;
5619         numberOfLevels = newNumberOfLevels;
5620         numberOfUDA = newNumberOfUda;
5621         totalNumberOfSections = newTotalNumberOfSections;
5622
5623         subjectNameList = new String[newNumberOfSubject];
5624
5625         continousLevels = new boolean[newNumberOfSubject];
5626
5627         sectionExists = new boolean[newNumberOfSubject][newNumberOfLevels];
5628         for (int i =0; i<newNumberOfSubject;i++)
5629         {
5630             continousLevels[i]=true;
5631             for (int j=0; j<newNumberOfLevels; j++)
5632             {
5633                 sectionExists[i][j]=false;
5634             }
5635         }

```

```

5636
5637     sectionName = new String[newNumberOfSubject][newNumberOfLevels];
5638
5639     bregmaLevels = new double[newNumberOfSubject][newNumberOfLevels];
5640
5641     UDAnames = new String[newNumberOfUda];
5642     includeTissue = new boolean[newNumberOfUda];
5643     includeBackground = new boolean[newNumberOfUda];
5644
5645     numberOfTissuePixels = new int[newNumberOfSubject][newNumberOfLevels];
5646     numberOfTissuePixelsLeft = new int[newNumberOfSubject][newNumberOfLevels];
5647     numberOfTissuePixelsRight = new int[newNumberOfSubject][newNumberOfLevels];
5648
5649     numberOfVentriclePixels = new int[newNumberOfSubject][newNumberOfLevels];
5650     numberOfVentriclePixelsLeft = new int[newNumberOfSubject][newNumberOfLevels];
5651     numberOfVentriclePixelsRight = new int[newNumberOfSubject][newNumberOfLevels];
5652
5653     tissueVolume = new double[newNumberOfSubject];
5654     tissueVolumeLeft = new double[newNumberOfSubject];
5655     tissueVolumeRight = new double[newNumberOfSubject];
5656
5657     ventricleVolume = new double[newNumberOfSubject];
5658     ventricleVolumeLeft = new double[newNumberOfSubject];
5659     ventricleVolumeRight = new double[newNumberOfSubject];
5660
5661     udaVolume = new double[newNumberOfSubject][newNumberOfUda];
5662
5663     sectionResultsObject = new Object[totalNumberOfSections][15+2*numberOfUDA];
5664     roundedSectionResultsObject = new Object[totalNumberOfSections][15+2*numberOfUDA];
5665 }
5666 }
5667
5668 static public class SectionData implements Serializable
5669 {
5670     //One object for each section in the project is created and saved to disk
5671     int highestNumberOfUDA=100;
5672
5673     Point[] udaOrigin = new Point[highestNumberOfUDA];
5674     boolean[] haveUdaOrigin = new boolean[highestNumberOfUDA];
5675
5676     int[] numberOfUdaPixels = new int[highestNumberOfUDA];
5677
5678     double colorTreshold=75;
5679     double intensityTreshold=50;
5680
5681     boolean[][] lrDividePixels;
5682     Point lrCurrentPoint = new Point(0,0);
5683     Point lrPreviousPoint = new Point(0,0);
5684     boolean lrHaveStartPoint = false;
5685
5686     boolean[][][] UDAlinePixels;
5687     Point[] UDACurrentPoint;
5688     Point[] UDAPreviousPoint;
5689     boolean[] UDAHaveStartPoint;
5690
5691     int positionInProject=0;

```

```

5692
5693     boolean[][] markedAsBackground;
5694     boolean[][] markedAsTissue;
5695
5696     public SectionData(int imageWidht, int imageHeight)
5697     {
5698         for (int i=0; i<haveUdaOrigin.length; i++) haveUdaOrigin[i]=false;
5699
5700         markedAsBackground= new boolean[imageWidht][imageHeight];
5701         markedAsTissue= new boolean[imageWidht][imageHeight];
5702
5703         for (int i=0; i<imageWidht; i++)
5704             for (int j=0; j<imageHeight; j++)
5705             {
5706                 markedAsBackground[i][j]=false;
5707                 markedAsTissue[i][j]=false;
5708             }
5709     }
5710 }
5711
5712
5713 public class MyMethods
5714 {
5715     //Sets the normal cursor
5716     public void setCustomCursor()
5717     {
5718         buttonPanel.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
5719         leftScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
5720         middleScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
5721         rightScrollPane.setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
5722     }
5723
5724     //The hourglass cursor is mainly used when swithing between panels
5725     public void setWaitCursor()
5726     {
5727         buttonPanel.setCursor(new Cursor(Cursor.WAIT_CURSOR));
5728         leftScrollPane.setCursor(new Cursor(Cursor.WAIT_CURSOR));
5729         middleScrollPane.setCursor(new Cursor(Cursor.WAIT_CURSOR));
5730         rightScrollPane.setCursor(new Cursor(Cursor.WAIT_CURSOR));
5731     }
5732
5733     //They keys A, S, D and W can be used to zoom in and out and move up
5734     //and down in the tree. This is blocked when a text field has the focus.
5735     public boolean textFieldIsFocusOwner()
5736     {
5737         boolean textFieldIsFocusOwner=false;
5738
5739         if (tissueDetection_rightSidePanel.colorTreshholdTextField.isFocusOwner())
5740             textFieldIsFocusOwner=true;
5741         if (tissueDetection_rightSidePanel.intensityTreshholdTextField.isFocusOwner())
5742             textFieldIsFocusOwner=true;
5743         if (conversion_rightSidePanel.distancemmTextField.isFocusOwner()) textFieldIsFocusOwner=true;
5744         if (section_rightSidePanel.bregmaLevelTextField.isFocusOwner()) textFieldIsFocusOwner=true;
5745         if (adjustImage_rightSidePanel.colorTreshholdTextField.isFocusOwner()) textFieldIsFocusOwner=true;
5746         if (adjustImage_rightSidePanel.intensityTreshholdTextField.isFocusOwner())
5747             textFieldIsFocusOwner=true;

```



```

5745
5746     return textFieldIsFocusOwner;
5747 }
5748
5749 //Makes a complete analysis of the section and saves the data
5750 public void analyzeSection(int subject, int section)
5751 {
5752     String fileName = myMethods.getOriginalImageFileName(subject, section);
5753     File file = new File(fileName);
5754     BufferedImage adjustedImage = new BufferedImage(2000,2000,BufferedImage.TYPE_INT_RGB);
5755
5756     try
5757     {
5758         adjustedImage = ImageIO.read(file);
5759     }catch (Exception error){ }
5760
5761     boolean[][] tissuePixels = myMethods.getTissuePixels(adjustedImage);
5762     int numberOfTissuePixels=myMethods.countPixels(tissuePixels);
5763     boolean[][] largestObjectPixels = myMethods.newGetLargestTissueObject(tissuePixels);
5764     boolean[][] outlinePixels = myMethods.findOutline(largestObjectPixels, adjustedImage.getWidth(),
adjustedImage.getHeight());
5765     Point startPosition = myMethods.findStartPositionFromOutline(outlinePixels);
5766     boolean[][] insidePixels = myMethods.newFillInside(outlinePixels, startPosition);
5767     boolean[][] ventriclePixels = myMethods.getVentriclePixels(tissuePixels, insidePixels);
5768     boolean[][] isLeft = myMethods.isLeftFromArray(currentSectionData.lRDividePixels);
5769
5770     numberOfTissuePixels=0;
5771     int numberOfTissuePixelsLeft=0;
5772     int numberOfTissuePixelsRight=0;
5773
5774     int numberOfVentriclePixels=0;
5775     int numberOfVentriclePixelsLeft=0;
5776     int numberOfVentriclePixelsRight=0;
5777
5778     for (int i=0; i<adjustedImage.getWidth(); i++)
5779         for (int j=0; j<adjustedImage.getHeight(); j++)
5780         {
5781             if (tissuePixels[i][j]) numberOfTissuePixels++;
5782             if (tissuePixels[i][j] && isLeft[i][j]) numberOfTissuePixelsLeft++;
5783             if (tissuePixels[i][j] && !isLeft[i][j]) numberOfTissuePixelsRight++;
5784
5785
5786             if (ventriclePixels[i][j]) numberOfVentriclePixels++;
5787             if (ventriclePixels[i][j] && isLeft[i][j]) numberOfVentriclePixelsLeft++;
5788             if (ventriclePixels[i][j] && !isLeft[i][j]) numberOfVentriclePixelsRight++;
5789         }
5790
5791     myProjectData.numberOfTissuePixels[subject][section] = numberOfTissuePixels;
5792     myProjectData.numberOfTissuePixelsLeft[subject][section] = numberOfTissuePixelsLeft;
5793     myProjectData.numberOfTissuePixelsRight[subject][section] = numberOfTissuePixelsRight;
5794
5795     myProjectData.numberOfVentriclePixels[subject][section] = numberOfVentriclePixels;
5796     myProjectData.numberOfVentriclePixelsLeft[subject][section] = numberOfVentriclePixelsLeft;
5797     myProjectData.numberOfVentriclePixelsRight[subject][section] = numberOfVentriclePixelsRight;
5798
5799

```

```

5800     currentSectionData=myMethods.readSectionData(subject, section);
5801     int[] numberOfUDAPixels = new int[myProjectData.numberOfUDA];
5802
5803     for (int UDAcounter=0; UDAcounter<myProjectData.numberOfUDA; UDAcounter++)
5804     {
5805         numberOfUDAPixels[UDAcounter] = myMethods.pixelsInUDA(subject, section, UDAcounter,
tissuePixels);
5806         currentSectionData.numberOfUdaPixels[UDAcounter] = numberOfUDAPixels[UDAcounter];
5807     }
5808     myMethods.saveSectionData(subject, section, currentSectionData);
5809 }
5810
5811 //Counts the pixels in a user defined area (UDA)
5812 //The largest continous tissue area (largestObject) has to be provided
5813 public int pixelsInUDA(int subject, int section, int UDANumber, boolean[][] tissue)
5814 {
5815     int numberOfPixels=0;
5816
5817     if (currentSectionData.haveUdaOrigin[UDANumber])
5818     {
5819         boolean[][] linePixels = currentSectionData.UDALinePixels[UDANumber];
5820         Point startPosition = currentSectionData.udaOrigin[UDANumber];
5821
5822         boolean[][] UDAPixels = myMethods.newFillInside(linePixels, startPosition);
5823
5824         boolean includeTissue = myProjectData.includeTissue[UDANumber];
5825         boolean includeBackground = myProjectData.includeBackground[UDANumber];
5826
5827         for (int i=0; i<tissue.length;i++)
5828             for (int j=0; j<tissue[0].length;j++)
5829             {
5830                 if (tissue[i][j] && !includeTissue) UDAPixels[i][j]=false;
5831                 if (!tissue[i][j] && !includeBackground) UDAPixels[i][j]=false;
5832             }
5833         numberOfPixels=myMethods.countPixels(UDAPixels);
5834     }
5835     return numberOfPixels;
5836 }
5837
5838 //Updates the data used in the section results table
5839 public void updateSectionResultsObject(int subject, int section)
5840 {
5841     double tempDouble;
5842     int counter=0;
5843
5844     if (myProjectData.sectionExists[subject][section])
5845     {
5846         currentSectionData=myMethods.readSectionData(subject, section);
5847
5848         counter=currentSectionData.positionInProject;
5849
5850         myProjectData.sectionResultsObject[counter][0] = myProjectData.subjectNameList[subject];
5851         myProjectData.roundedSectionResultsObject[counter][0] =
myProjectData.subjectNameList[subject];
5852

```

```

5853         myProjectData.sectionResultsObject[counter][1] =
myProjectData.sectionName[subject][section];
5854         myProjectData.roundedSectionResultsObject[counter][1] =
myProjectData.sectionName[subject][section];
5855
5856         myProjectData.sectionResultsObject[counter][2] =
myProjectData.bregmaLevels[subject][section];
5857         myProjectData.roundedSectionResultsObject[counter][2] =
myProjectData.bregmaLevels[subject][section];
5858
5859         myProjectData.sectionResultsObject[counter][3] =
myProjectData.numberOfTissuePixels[subject][section];
5860         myProjectData.roundedSectionResultsObject[counter][3] =
myProjectData.numberOfTissuePixels[subject][section];
5861
5862         tempDouble=( double)
myProjectData.numberOfTissuePixels[subject][section]/myProjectData.conversionFactor;
5863         myProjectData.sectionResultsObject[counter][4] = tempDouble;
5864         tempDouble= (double) (Math.round(tempDouble*100))/100;
5865         myProjectData.roundedSectionResultsObject[counter][4] = tempDouble;
5866
5867         myProjectData.sectionResultsObject[counter][5] =
myProjectData.numberOfTissuePixelsLeft[subject][section];
5868         myProjectData.roundedSectionResultsObject[counter][5] =
myProjectData.numberOfTissuePixelsLeft[subject][section];
5869
5870         tempDouble = (double)
myProjectData.numberOfTissuePixelsLeft[subject][section]/myProjectData.conversionFactor;
5871         myProjectData.sectionResultsObject[counter][6] = tempDouble;
5872         tempDouble= (double) (Math.round(tempDouble*100))/100;
5873         myProjectData.roundedSectionResultsObject[counter][6] = tempDouble;
5874
5875         myProjectData.sectionResultsObject[counter][7] =
myProjectData.numberOfTissuePixelsRight[subject][section];
5876         myProjectData.roundedSectionResultsObject[counter][7] =
myProjectData.numberOfTissuePixelsRight[subject][section];
5877
5878         tempDouble = (double)
myProjectData.numberOfTissuePixelsRight[subject][section]/myProjectData.conversionFactor;
5879         myProjectData.sectionResultsObject[counter][8] = tempDouble;
5880         tempDouble= (double) (Math.round(tempDouble*100))/100;
5881         myProjectData.roundedSectionResultsObject[counter][8] = tempDouble;
5882
5883         myProjectData.sectionResultsObject[counter][9] =
myProjectData.numberOfVentriclePixels[subject][section];
5884         myProjectData.roundedSectionResultsObject[counter][9] =
myProjectData.numberOfVentriclePixels[subject][section];
5885
5886         tempDouble = (double)
myProjectData.numberOfVentriclePixels[subject][section]/myProjectData.conversionFactor;
5887         myProjectData.sectionResultsObject[counter][10] = tempDouble;
5888         tempDouble= (double) (Math.round(tempDouble*100))/100;
5889         myProjectData.roundedSectionResultsObject[counter][10] = tempDouble;
5890
5891         myProjectData.sectionResultsObject[counter][11] =
myProjectData.numberOfVentriclePixelsLeft[subject][section];

```

```

5892         myProjectData.roundedSectionResultsObject[counter][11] =
myProjectData.numberOfVentriclePixelsLeft[subject][section];
5893
5894         tempDouble = (double)
myProjectData.numberOfVentriclePixelsLeft[subject][section]/myProjectData.conversionFactor;
5895         myProjectData.sectionResultsObject[counter][12] = tempDouble;
5896         tempDouble= (double) (Math.round(tempDouble*100))/100;
5897         myProjectData.roundedSectionResultsObject[counter][12] = tempDouble;
5898
5899         myProjectData.sectionResultsObject[counter][13] =
myProjectData.numberOfVentriclePixelsRight[subject][section];
5900         myProjectData.roundedSectionResultsObject[counter][13] =
myProjectData.numberOfVentriclePixelsRight[subject][section];
5901
5902         tempDouble = (double)
myProjectData.numberOfVentriclePixelsRight[subject][section]/myProjectData.conversionFactor;
5903         myProjectData.sectionResultsObject[counter][14] = tempDouble;
5904         tempDouble= (double) (Math.round(tempDouble*100))/100;
5905         myProjectData.roundedSectionResultsObject[counter][14] = tempDouble;
5906
5907         for (int k=0; k<myProjectData.numberOfUDA;k++)
5908         {
5909             myProjectData.sectionResultsObject[counter][15+2*k] =
currentSectionData.numberOfUdaPixels[k];
5910             myProjectData.roundedSectionResultsObject[counter][15+2*k] =
currentSectionData.numberOfUdaPixels[k];
5911
5912             tempDouble = (double)
currentSectionData.numberOfUdaPixels[k]/myProjectData.conversionFactor;
5913             myProjectData.sectionResultsObject[counter][16+2*k] = tempDouble;
5914             tempDouble= (double) (Math.round(tempDouble*100))/100;
5915             myProjectData.roundedSectionResultsObject[counter][16+2*k] = tempDouble;
5916         }
5917     }
5918 }
5919
5920 //Calculates the volumes for a subject. It assumes that the areas in the sections are up to date.
5921 public void analyzeSubject(int subject)
5922 {
5923     double area1=0;
5924     double area2=0;
5925     double distance=0;
5926     double sliceVolume=0;
5927
5928     double tissueVolume=0;
5929     double tissueVolumeLeft=0;
5930     double tissueVolumeRight=0;
5931
5932     double ventricleVolume=0 ;
5933     double ventricleVolumeLeft=0;
5934     double ventricleVolumeRight=0;
5935
5936     double[] udaVolume = new double[myProjectData.numberOfUDA];
5937
5938     for (int udaCounter=0; udaCounter<myProjectData.numberOfUDA; udaCounter++)
5939         udaVolume[udaCounter]=0;

```

```

5940
5941     for (int j=0;j<myProjectData.numberOfLevels-1;j++)
5942     {
5943         if (myProjectData.sectionExists[subject][j] && myProjectData.sectionExists[subject][j+1])
5944         {
5945             distance= myProjectData.bregmaLevels[subject][j+1]-myProjectData.bregmaLevels[subject][j];
5946             distance = Math.abs(distance);
5947
5948             SectionData firstSectionData = myMethods.readSectionData(subject, j);
5949             SectionData secondSectionData = myMethods.readSectionData(subject, j+1);
5950
5951             area1 = (double)
myProjectData.numberOfTissuePixels[subject][j]/myProjectData.conversionFactor;
5952             area2 = (double)
myProjectData.numberOfTissuePixels[subject][j+1]/myProjectData.conversionFactor;
5953
5954             sliceVolume=distance*(area1+area2)/2;
5955             tissueVolume=tissueVolume+sliceVolume;
5956
5957             area1 = (double)
myProjectData.numberOfTissuePixelsLeft[subject][j]/myProjectData.conversionFactor;
5958             area2 = (double)
myProjectData.numberOfTissuePixelsLeft[subject][j+1]/myProjectData.conversionFactor;
5959
5960             sliceVolume=distance*(area1+area2)/2;
5961             tissueVolumeLeft=tissueVolumeLeft+sliceVolume;
5962
5963             area1 = (double)
myProjectData.numberOfTissuePixelsRight[subject][j]/myProjectData.conversionFactor;
5964             area2 = (double)
myProjectData.numberOfTissuePixelsRight[subject][j+1]/myProjectData.conversionFactor;
5965
5966             sliceVolume=distance*(area1+area2)/2;
5967             tissueVolumeRight=tissueVolumeRight+sliceVolume;
5968
5969             area1 = (double)
myProjectData.numberOfVentriclePixels[subject][j]/myProjectData.conversionFactor;
5970             area2 = (double)
myProjectData.numberOfVentriclePixels[subject][j+1]/myProjectData.conversionFactor;
5971
5972             sliceVolume=distance*(area1+area2)/2;
5973             ventricleVolume=ventricleVolume+sliceVolume;
5974
5975             area1 = (double)
myProjectData.numberOfVentriclePixelsLeft[subject][j]/myProjectData.conversionFactor;
5976             area2 = (double)
myProjectData.numberOfVentriclePixelsLeft[subject][j+1]/myProjectData.conversionFactor;
5977
5978             sliceVolume=distance*(area1+area2)/2;
5979             ventricleVolumeLeft=ventricleVolumeLeft+sliceVolume;
5980
5981             area1 = (double)
myProjectData.numberOfVentriclePixelsRight[subject][j]/myProjectData.conversionFactor;
5982             area2 = (double)
myProjectData.numberOfVentriclePixelsRight[subject][j+1]/myProjectData.conversionFactor;
5983

```

```

5984         sliceVolume=distance*(area1+area2)/2;
5985         ventricleVolumeRight=ventricleVolumeRight+sliceVolume;
5986
5987         for (int udaCounter=0; udaCounter<myProjectData.numberOfUDA; udaCounter++)
5988         {
5989
5990             area1 = (double)
firstSectionData.numberOfUdaPixels[udaCounter]/myProjectData.conversionFactor;
5991             area2 = (double)
secondSectionData.numberOfUdaPixels[udaCounter]/myProjectData.conversionFactor;
5992
5993             sliceVolume=distance*(area1+area2)/2;
5994             udaVolume[udaCounter]=udaVolume[udaCounter]+sliceVolume;
5995
5996         }
5997     }
5998 }
5999
6000 myProjectData.tissueVolume[subject] = tissueVolume;
6001 myProjectData.tissueVolumeLeft[subject] = tissueVolumeLeft;
6002 myProjectData.tissueVolumeRight[subject] = tissueVolumeRight;
6003
6004 myProjectData.ventricleVolume[subject] = ventricleVolume;
6005 myProjectData.ventricleVolumeLeft[subject] = ventricleVolumeLeft;
6006 myProjectData.ventricleVolumeRight[subject] = ventricleVolumeRight;
6007
6008 for (int udaCounter=0; udaCounter<myProjectData.numberOfUDA; udaCounter++)
6009     myProjectData.udaVolume[subject][udaCounter] = udaVolume[udaCounter];
6010 }
6011
6012 //Determines if a pixel is tissue or background based on the color and treshold values.
6013 public boolean pixelIsTissue (int[] color)
6014 {
6015     boolean isTissue=true;
6016
6017     if (color[0]+color[1]+color[2]==0)
6018     {
6019         isTissue=false;
6020     }
6021     else if (100*(color[0]+color[2])/(color[0]+color[1]+color[2])<currentSectionData.colorTreshold)
6022         isTissue=false;
6023
6024     if (color[0]+color[1]+color[2]<currentSectionData.intensityTreshold)
6025         isTissue=false;
6026
6027
6028     return isTissue;
6029 }
6030
6031 //Returns an array of boolean with the same size as the input image.
6032 //The value is true if the pixel is tissue and false if it is background
6033 public boolean[][] getTissuePixels(BufferedImage inputImage)
6034 {
6035     boolean[][] tissuePixels = new boolean[inputImage.getWidth()][inputImage.getHeight()];
6036     Raster myRaster=inputImage.getRaster();
6037     int[] color = new int[3];

```

```

6038
6039     for (int i=0;i<inputImage.getWidth();i++)
6040         for (int j=0;j<inputImage.getHeight();j++)
6041             {
6042                 color=myRaster.getPixel(i,j,color);
6043
6044                 tissuePixels[i][j]=this.pixelIsTissue(color);
6045
6046                 if (currentSectionData.markedAsBackground[i][j]) tissuePixels[i][j]=false;
6047                 if (currentSectionData.markedAsTissue[i][j]) tissuePixels[i][j]=true;
6048
6049                 if (i<3 || i>(inputImage.getWidth()-4) || j<3 || j>(inputImage.getHeight()-4))
6050                     {
6051                         //Pixels close to the border of the image are set as false to avoid
6052                         //trying to access pixels outside the image when working with objects
6053                         //close to the border
6054                         tissuePixels[i][j]=false;
6055                     }
6056             }
6057
6058     //If theres no tissue pixels one pixel is set as tissue to avoid errors
6059     if (myMethods.countPixels(tissuePixels) == 0) tissuePixels[3][3]=true;
6060
6061     return tissuePixels;
6062 }
6063
6064
6065
6066
6067 //Returns the largest continous tissue area
6068 //To make it run faster it start the scan on the middle horisontal line where
6069 // the largest object is likely to be. It then scans to the bottom of the image
6070 //and then continous from the top to the middle. If a detected object makes up
6071 //more than half of the tissue pixels a larger object cant possibly be found
6072 //and the search is stopped.
6073
6074
6075 public boolean[][] newGetLargestTissueObject(boolean[][] tissuePixels)
6076 {
6077     int width = tissuePixels.length;
6078     int height = tissuePixels[0].length;
6079     int numberOfTissuePixels = this.countObjectSize(tissuePixels, width, height);
6080
6081     int sizeOfLargestObject=0;
6082     int currentObjectSize=0;
6083
6084     boolean[][] largestObject = new boolean [width][height];
6085     boolean[][] currentObject = new boolean [width][height];
6086     boolean[][] partOfAnObject = new boolean [width][height];
6087
6088     for (int i=0;i<width;i++)
6089         for (int j=0;j<height;j++)
6090             {
6091                 largestObject[i][j]=false;
6092                 currentObject[i][j]=false;
6093                 partOfAnObject[i][j]=false;

```

```

6094     }
6095
6096
6097     for (int x=width/2; x<width; x++)
6098         for (int y=0; y<height; y++)
6099         {
6100             if (!partOfAnObject[x][y] && tissuePixels[x][y])
6101             {
6102                 currentObject= this.newFillObject(tissuePixels, currentObject, new Point(x,y));
6103                 currentObjectSize=this.countObjectSize(currentObject, width, height);
6104
6105                 for (int i=0;i<width;i++)
6106                     for (int j=0;j<height;j++)
6107                     {
6108                         if (currentObject[i][j]) partOfAnObject[i][j]=true;
6109                     }
6110
6111                 if (currentObjectSize>sizeOfLargestObject)
6112                 {
6113                     sizeOfLargestObject=currentObjectSize;
6114                     for (int i=0;i<width;i++)
6115                         for (int j=0;j<height;j++)
6116                         {
6117                             largestObject[i][j]=currentObject[i][j];
6118                         }
6119                     if (sizeOfLargestObject>5+numberOfTissuePixels/2) return largestObject;
6120                 }
6121             }
6122         }
6123     }
6124
6125     for (int x=0; x<width/2; x++)
6126         for (int y=0; y<height; y++)
6127         {
6128             if (!partOfAnObject[x][y] && tissuePixels[x][y])
6129             {
6130                 currentObject= this.newFillObject(tissuePixels, currentObject, new Point(x,y));
6131                 currentObjectSize=this.countObjectSize(currentObject, width, height);
6132
6133                 for (int i=0;i<width;i++)
6134                     for (int j=0;j<height;j++)
6135                     {
6136                         if (currentObject[i][j]) partOfAnObject[i][j]=true;
6137                     }
6138
6139                 if (currentObjectSize>sizeOfLargestObject)
6140                 {
6141                     sizeOfLargestObject=currentObjectSize;
6142                     for (int i=0;i<width;i++)
6143                         for (int j=0;j<height;j++)
6144                         {
6145                             largestObject[i][j]=currentObject[i][j];
6146                         }
6147                     if (sizeOfLargestObject>5+numberOfTissuePixels/2) return largestObject;
6148                 }
6149             }

```



```

6150     }
6151     return largestObject;
6152 }
6153
6154 public boolean[][] newFillObject(boolean [][] tissuePixels, boolean[][] currentObject, Point startPosition)
6155 {
6156     int width = tissuePixels.length;
6157     int height = tissuePixels[0].length;
6158
6159     boolean[][] isChecked = new boolean[width][height];
6160
6161     for (int x=0;x<width;x++)
6162         for (int y=0;y<height;y++)
6163         {
6164             currentObject[x][y]=false;
6165             isChecked[x][y]=false;
6166         }
6167
6168     currentObject[startPosition.x][startPosition.y]=true;
6169
6170     boolean foundMore=true;
6171
6172     while (foundMore)
6173     {
6174         foundMore=false;
6175
6176         for (int x=0;x<width;x++)
6177             for (int y=0;y<height;y++)
6178                 if (currentObject[x][y] && !isChecked[x][y])
6179                 {
6180                     try{
6181                         if(!currentObject[x][y-1] && tissuePixels[x][y-1])
6182                         {
6183                             currentObject[x][y-1]=true;
6184                             foundMore=true;
6185                         }
6186                     }catch(Exception error) {}
6187
6188                     try{
6189                         if(!currentObject[x][y+1] && tissuePixels[x][y+1])
6190                         {
6191                             currentObject[x][y+1]=true;
6192                             foundMore=true;
6193                         }
6194                     }catch(Exception error) {}
6195
6196                     try{
6197                         if(!currentObject[x-1][y] && tissuePixels[x-1][y])
6198                         {
6199                             currentObject[x-1][y]=true;
6200                             foundMore=true;
6201                         }
6202                     }catch(Exception error) {}
6203
6204                     try{
6205                         if(!currentObject[x+1][y] && tissuePixels[x+1][y])

```

```

6206         {
6207             currentObject[x+1][y]=true;
6208             foundMore=true;
6209         }
6210     }catch(Exception error) {}
6211
6212     isChecked[x][y]=true;
6213
6214 }
6215 if (!foundMore) return currentObject;
6216
6217 foundMore=false;
6218
6219 for (int x=width-1;x>=0;x--)
6220 for (int y=height-1;y>=0;y--)
6221 if (currentObject[x][y] && !isChecked[x][y])
6222 {
6223     try{
6224         if(!currentObject[x][y-1] && tissuePixels[x][y-1])
6225         {
6226             currentObject[x][y-1]=true;
6227             foundMore=true;
6228         }
6229     }catch(Exception error) {}
6230
6231     try{
6232         if(!currentObject[x][y+1] && tissuePixels[x][y+1])
6233         {
6234             currentObject[x][y+1]=true;
6235             foundMore=true;
6236         }
6237     }catch(Exception error) {}
6238
6239     try{
6240         if(!currentObject[x-1][y] && tissuePixels[x-1][y])
6241         {
6242             currentObject[x-1][y]=true;
6243             foundMore=true;
6244         }
6245     }catch(Exception error) {}
6246
6247     try{
6248         if(!currentObject[x+1][y] && tissuePixels[x+1][y])
6249         {
6250             currentObject[x+1][y]=true;
6251             foundMore=true;
6252         }
6253     }catch(Exception error) {}
6254
6255     isChecked[x][y]=true;
6256 }
6257 }
6258 return currentObject;
6259 }
6260
6261 public int countObjectSize(boolean[][] currentObject, int width, int height)

```

```

6262 {
6263     int objectSize=0;
6264
6265     for (int i=0;i<width;i++)
6266         for (int j=0;j<height;j++)
6267             {
6268                 if (currentObject[i][j]) objectSize++;
6269             }
6270
6271     return objectSize;
6272 }
6273
6274
6275 //Start by scanning in straight lines from the sides until an object pixel is found.
6276 //This will quickly find most of the outline but parts might be missed. To make
6277 //sure that the entire outline the entire image is scanned. If a background pixel is
6278 //found next to an object pixel and an outline pixel it is considered a new outline pixel.
6279 //This is continued until a complete scan is made without finding more outline pixels.
6280 public boolean[][] findOutline (boolean[][] objectPixels, int width, int height)
6281 {
6282     boolean[][] outlinePixels = new boolean [width][height];
6283     boolean[][] pixelIsChecked = new boolean [width][height];
6284
6285     for (int i=0;i<width;i++)
6286         for (int j=0;j<height;j++)
6287             {
6288                 outlinePixels[i][j]=false;
6289                 pixelIsChecked[i][j]=false;
6290             }
6291
6292     boolean foundEdge;
6293
6294     // Scan from above
6295     for (int i=0;i<width;i++)
6296     {
6297         foundEdge=false;
6298         for (int j=0;j<height;j++)
6299             {
6300                 if (!foundEdge && objectPixels[i][j])
6301                 {
6302                     outlinePixels[i][j-1]=true;
6303                     foundEdge=true;
6304                 }
6305             }
6306     }
6307
6308     // Scan from below
6309     for (int i=0;i<width;i++)
6310     {
6311         foundEdge=false;
6312         for (int j=height-1;j>-1;j--)
6313             {
6314                 if (!foundEdge && objectPixels[i][j])
6315                 {
6316                     outlinePixels[i][j+1]=true;
6317                     foundEdge=true;

```

```

6318     }
6319 }
6320 }
6321
6322 // Scan from the right
6323 for (int j=0;j<height;j++)
6324 {
6325     foundEdge=false;
6326     for (int i=0;i<width;i++)
6327     {
6328         if (!foundEdge && objectPixels[i][j])
6329         {
6330             outlinePixels[i-1][j]=true;
6331             foundEdge=true;
6332         }
6333     }
6334 }
6335
6336 // Scan from the left
6337 for (int j=0;j<height;j++)
6338 {
6339     foundEdge=false;
6340     for (int i=width-1;i>-1;i--)
6341     {
6342         if (!foundEdge && objectPixels[i][j])
6343         {
6344             outlinePixels[i+1][j]=true;
6345             foundEdge=true;
6346         }
6347     }
6348 }
6349
6350 //Scan for remaining outline pixels
6351
6352 boolean foundMore=true;
6353 boolean nextToOutline;
6354 boolean nextToTissue;
6355
6356 while (foundMore)
6357 {
6358     foundMore=false;
6359     for (int i=1;i<width-1;i++)
6360         for (int j=1;j<height-1;j++)
6361         {
6362             if (outlinePixels[i][j] && !pixelIsChecked[i][j])
6363             {
6364                 if (this.pixelIsNewOutline(outlinePixels, objectPixels, i+1, j))
6365                 {
6366                     outlinePixels[i+1][j]=true;
6367                     foundMore=true;
6368                 }
6369                 if (this.pixelIsNewOutline(outlinePixels, objectPixels, i-1, j))
6370                 {
6371                     outlinePixels[i-1][j]=true;
6372                     foundMore=true;
6373                 }

```

```

6374         if (this.pixelIsNewOutline(outlinePixels, objectPixels, i, j+1))
6375         {
6376             outlinePixels[i][j+1]=true;
6377             foundMore=true;
6378         }
6379         if (this.pixelIsNewOutline(outlinePixels, objectPixels, i, j-1))
6380         {
6381             outlinePixels[i][j-1]=true;
6382             foundMore=true;
6383         }
6384         pixelIsChecked[i][j]=true;
6385     }
6386 }
6387 }
6388 }
6389
6390     return outlinePixels;
6391 }
6392
6393 //Used by the findOutline method. It checks whether a pixel meets the criteria for an outline pixel.
6394 public boolean pixelIsNewOutline(boolean[][] outlinePixels, boolean[][] objectPixels, int i, int j)
6395 {
6396     boolean pixelIsOutline=false;
6397
6398     if (!outlinePixels[i][j] && !objectPixels[i][j])
6399     {
6400         boolean nextToOutline=false;
6401         if (outlinePixels[i][j+1]) nextToOutline=true;
6402         if (outlinePixels[i][j-1]) nextToOutline=true;
6403         if (outlinePixels[i+1][j]) nextToOutline=true;
6404         if (outlinePixels[i-1][j]) nextToOutline=true;
6405
6406         if (nextToOutline)
6407         {
6408             boolean nextToTissue=false;
6409             if (objectPixels[i][j+1]) nextToTissue=true;
6410             if (objectPixels[i][j-1]) nextToTissue=true;
6411             if (objectPixels[i+1][j]) nextToTissue=true;
6412             if (objectPixels[i-1][j]) nextToTissue=true;
6413             if (objectPixels[i+1][j+1]) nextToTissue=true;
6414             if (objectPixels[i-1][j-1]) nextToTissue=true;
6415             if (objectPixels[i+1][j-1]) nextToTissue=true;
6416             if (objectPixels[i-1][j+1]) nextToTissue=true;
6417
6418             if (nextToTissue) pixelIsOutline=true;
6419         }
6420     }
6421     return pixelIsOutline;
6422 }
6423
6424 public boolean[][] newFillInside(boolean[][] outlinePixels, Point startPoint)
6425 {
6426     int width = outlinePixels.length;
6427     int height = outlinePixels[0].length;
6428
6429     boolean[][] insidePixels = new boolean [width][height];

```

```

6430
6431 for (int i=0;i<width;i++)
6432     for (int j=0;j<height;j++)
6433     {
6434         insidePixels[i][j]=false;
6435     }
6436
6437 insidePixels[startPoint.x][startPoint.y]=true;
6438
6439 boolean foundMore=true;
6440
6441 while (foundMore)
6442 {
6443     foundMore=false;
6444
6445     for (int x=0;x<width;x++)
6446     for (int y=0;y<height;y++)
6447     if (insidePixels[x][y])
6448     {
6449         try{
6450             if(!insidePixels[x][y-1] && !outlinePixels[x][y-1])
6451             {
6452                 insidePixels[x][y-1]=true;
6453                 foundMore=true;
6454             }
6455         }catch(Exception error) {}
6456
6457         try{
6458             if(!insidePixels[x][y+1] && !outlinePixels[x][y+1])
6459             {
6460                 insidePixels[x][y+1]=true;
6461                 foundMore=true;
6462             }
6463         }catch(Exception error) {}
6464
6465         try{
6466             if(!insidePixels[x-1][y] && !outlinePixels[x-1][y])
6467             {
6468                 insidePixels[x-1][y]=true;
6469                 foundMore=true;
6470             }
6471         }catch(Exception error) {}
6472
6473         try{
6474             if(!insidePixels[x+1][y] && !outlinePixels[x+1][y])
6475             {
6476                 insidePixels[x+1][y]=true;
6477                 foundMore=true;
6478             }
6479         }catch(Exception error) {}
6480
6481     }
6482     if (!foundMore) return insidePixels;
6483
6484     foundMore=false;
6485

```

```

6486     for (int x=width-1;x>=0;x--)
6487     for (int y=height-1;y>=0;y--)
6488         if (insidePixels[x][y])
6489             {
6490                 try{
6491                     if(!insidePixels[x][y-1] && !outlinePixels[x][y-1])
6492                         {
6493                             insidePixels[x][y-1]=true;
6494                             foundMore=true;
6495                         }
6496                 }catch(Exception error) {}
6497
6498                 try{
6499                     if(!insidePixels[x][y+1] && !outlinePixels[x][y+1])
6500                         {
6501                             insidePixels[x][y+1]=true;
6502                             foundMore=true;
6503                         }
6504                 }catch(Exception error) {}
6505
6506                 try{
6507                     if(!insidePixels[x-1][y] && !outlinePixels[x-1][y])
6508                         {
6509                             insidePixels[x-1][y]=true;
6510                             foundMore=true;
6511                         }
6512                 }catch(Exception error) {}
6513
6514                 try{
6515                     if(!insidePixels[x+1][y] && !outlinePixels[x+1][y])
6516                         {
6517                             insidePixels[x+1][y]=true;
6518                             foundMore=true;
6519                         }
6520                 }catch(Exception error) {}
6521             }
6522     }
6523     return insidePixels;
6524 }
6525
6526 //Finds a pixel which is inside the outline. This is then used as the
6527 //start point when finding all pixels inside the outline.
6528 public Point findStartPositionFromOutline(boolean[][] outlinePixels)
6529 {
6530     boolean foundStartPoint=false;
6531     boolean foundEdge;
6532     Point startPoint = new Point();
6533     for (int i=0;i<outlinePixels.length;i++)
6534     {
6535         foundEdge=false;
6536         for (int j=0;j<outlinePixels[0].length-1;j++)
6537         {
6538             if (!foundStartPoint && !foundEdge && outlinePixels[i][j] && !outlinePixels[i][j+1])
6539             {
6540                 startPoint.x=i;
6541                 startPoint.y=j+1;

```

```

6542         foundStartPoint=true;
6543     }
6544     if (outlinePixels[i][j]) foundEdge=true;
6545 }
6546 }
6547 return startPoint;
6548 }
6549
6550 //All background pixels inside the outline are considered ventricle pixels.
6551 public boolean[][] getVentriclePixels (boolean[][] tissuePixels, boolean[][] insidePixels)
6552 {
6553     boolean[][] ventriclePixels= new boolean[tissuePixels.length][tissuePixels[0].length];
6554
6555     for (int i=0;i<tissuePixels.length;i++)
6556         for (int j=0;j<tissuePixels[0].length;j++)
6557         {
6558             if (!tissuePixels[i][j] && insidePixels[i][j]) ventriclePixels[i][j]=true;
6559             else ventriclePixels[i][j]=false;
6560         }
6561     return ventriclePixels;
6562 }
6563
6564 //Counts the number of true in an array of boolean.
6565 public int countPixels(boolean[][] inputArray)
6566 {
6567     int numberOfPixels=0;
6568
6569     for (int i=0;i<inputArray.length;i++)
6570         for (int j=0;j<inputArray[0].length;j++)
6571         {
6572             if (inputArray[i][j]) numberOfPixels++;
6573         }
6574     return numberOfPixels;
6575 }
6576
6577 //If the boolean for the pixel is true it is changed to the provided color.
6578 public void changePixelsInImage(BufferedImage image, boolean[][] outlinePixels, int[] color)
6579 {
6580     WritableRaster raster = image.getRaster();
6581
6582     for (int i=0;i<image.getWidth();i++)
6583         for (int j=0;j<image.getHeight();j++)
6584         {
6585             if (outlinePixels[i][j]) raster.setPixel(i, j, color);
6586         }
6587 }
6588
6589 //Scans all horizontal lines from left to right. All pixels in the line
6590 //are considered to be left until a pixel in the lineArray is encountered.
6591 public boolean[][] isLeftFromArray(boolean[][] lineArray)
6592 {
6593     boolean[][] isLeft = new boolean[lineArray.length][lineArray[0].length];
6594
6595     for (int i=0;i<lineArray.length;i++)
6596         for (int j=0;j<lineArray[0].length;j++)
6597             isLeft[i][j] =false;

```



```

6598
6599     boolean foundDivider;
6600
6601     for (int j=0;j<lineArray[0].length;j++)
6602     {
6603         foundDivider=false;
6604         for (int i=0;i<lineArray.length;i++)
6605         {
6606             if (lineArray[i][j]) foundDivider=true;
6607             {
6608                 if (!foundDivider) isLeft[i][j]=true;
6609                 else isLeft[i][j]=false;
6610             }
6611         }
6612     }
6613     return isLeft;
6614 }
6615
6616 public String getOriginalImageFileName(int subjectNumber, int sectionNumber)
6617 {
6618     String projectLocation = myProjectData.projectLocation;
6619
6620     Integer subjectInteger= new Integer(subjectNumber);
6621     String subjectNumberString = subjectInteger.toString();
6622
6623     Integer sectionInteger= new Integer(sectionNumber);
6624     String sectionNumberString = sectionInteger.toString();
6625
6626     String imageFileName = new String(subjectNumberString+"."
6627         +sectionNumberString+".OriginalImage");
6628     imageFileName = (projectLocation+"/"+imageFileName);
6629
6630     return imageFileName;
6631 }
6632
6633 public String getSectionDataFileName(int subjectNumber, int sectionNumber)
6634 {
6635     String projectLocation = myProjectData.projectLocation;
6636
6637     Integer subjectInteger= new Integer(subjectNumber);
6638     String subjectNumberString = subjectInteger.toString();
6639
6640     Integer sectionInteger= new Integer(sectionNumber);
6641     String sectionNumberString = sectionInteger.toString();
6642
6643     String fileName = new String(subjectNumberString+"."
6644         +sectionNumberString+".SectionData");
6645     fileName = (projectLocation+"/"+fileName);
6646
6647     return fileName;
6648 }
6649
6650 public void saveSectionData(int subjectNumber, int sectionNumber, SectionData dataToSave)
6651 {
6652     String saveFileName= myMethods.getSectionDataFileName(subjectNumber, sectionNumber);
6653

```

```

6654     File saveObjectFile = new File(saveFileName);
6655
6656     FileOutputStream fos = null;
6657     ObjectOutputStream out = null;
6658     try
6659     {
6660         fos = new FileOutputStream(saveObjectFile);
6661         out = new ObjectOutputStream(fos);
6662
6663         out.writeObject(dataToSave);
6664
6665         out.close();
6666     }
6667     catch (java.io.FileNotFoundException er)
6668     {
6669         System.out.println(er.toString());
6670     }
6671     catch (java.io.IOException ex)
6672     {
6673         System.out.println(ex.toString());
6674     }
6675 }
6676
6677 public SectionData readSectionData(int subjectNumber, int sectionNumber)
6678 {
6679     SectionData newSectionData = new SectionData(myProjectData.imageWidth,
myProjectData.imageHeight);
6680
6681     String fileName= myMethods.getSectionDataFileName(subjectNumber, sectionNumber);
6682
6683     File file = new File(fileName);
6684
6685     FileInputStream fis = null;
6686     ObjectInputStream in = null;
6687     try
6688     {
6689         fis = new FileInputStream(file);
6690         in = new ObjectInputStream(fis);
6691         newSectionData = (SectionData)in.readObject();
6692         in.close();
6693     }
6694     catch (IOException ex)
6695     {
6696         ex.printStackTrace();
6697     }
6698     catch (ClassNotFoundException ex)
6699     {}
6700
6701
6702     return newSectionData;
6703 }
6704
6705 public void saveProjectData()
6706 {
6707     //Triggers a WhenLeftInTree for the current panel to make sure
6708     //that any last changes are also being saved

```

```

6709     TreePath selectionPath = treePanel.tree.getSelectionPath();
6710     treePanel.tree.setSelectionRow(0);
6711     treePanel.tree.setSelectionPath(selectionPath);
6712
6713
6714     String saveFileName = myProjectData.projectLocation;
6715     saveFileName=saveFileName+"/ProjectData.ser";
6716     File saveObjectFile = new File(saveFileName);
6717     FileOutputStream fos = null;
6718     ObjectOutputStream out = null;
6719     try
6720     {
6721         fos = new FileOutputStream(saveObjectFile);
6722         out = new ObjectOutputStream(fos);
6723         out.writeObject(myProjectData);
6724         out.close();
6725     }
6726     catch (java.io.FileNotFoundException er)
6727     {
6728
6729     }
6730     catch (java.io.IOException ex)
6731     {
6732
6733     }
6734 }
6735
6736 //Adds pixels to a line based on the setting for lineWidth to make
6737 //the line easier to see.
6738 public boolean[][] widenLine (boolean[][] linePixels, int lineWidth)
6739 {
6740     boolean[][] newLine = new boolean[linePixels.length][linePixels[0].length];
6741
6742     for (int i=0 ; i<linePixels.length ; i++)
6743         for (int j =0; j<linePixels[0].length;j++)
6744         {
6745             if (linePixels[i][j])
6746             {
6747                 for (int x=-lineWidth;x<lineWidth+1;x++)
6748                     for (int y = -lineWidth; y<lineWidth+1;y++)
6749                     {
6750                         try
6751                         {
6752                             newLine[i+x][j+y]=true;
6753                         }catch (Exception error){ }
6754                     }
6755             }
6756         }
6757     return newLine;
6758 }
6759
6760 public String getHelpTextChoseProjectFolder()
6761 {
6762     String helpText = new String(
6763         "LOCATE SOURCE IMAGES"+ "\n"+ "\n"+
6764         "When you click the "Select project folder" button you will be" + "\n"+

```

```

6765         " asked to locate the folder containing your source images. " + '\n'+
6766         "The image files from each subject have to be located in a " + '\n'+
6767         "folder (subject folder). All the subject folders have to be " + '\n'+
6768         "placed in the same folder (project folder). When you locate " + '\n'+
6769         "the project folder the program will scan for usable image " + '\n'+
6770         "files and present the number of subject and sections. Any files " + '\n'+
6771         "the program can't open as images are ignored. All images have " + '\n'+
6772         "to be of the same size and the program makes sure they are. " + '\n'+
6773         "The folder names will be used as subject names and the image " + '\n'+
6774         "file names as section names in the project." + '\n'+ '\n'+
6775         "IMPORTANT: The images for subject have to be named in strict " + '\n'+
6776         "alphabetical order, otherwise they will be placed in the wrong " + '\n'+
6777         "position. Especially note that Section12 will appear before " + '\n'+
6778         "Section2, to avoid this problem change the name to Section02.");
6779     return helpText;
6780 }
6781
6782 public String getHelpTextChoseSaveFolder()
6783 {
6784     String helpText = new String(
6785         "CHOOSE WHERE TO SAVE PROJECT"+ '\n'+ '\n'+
6786         "You have to locate a folder where the project data should be " + '\n'+
6787         "saved (save folder). Three files will be created for each " + '\n'+
6788         "section, one that holds the original image, one that holds any " + '\n'+
6789         "modifications done to the image and one file which stores data " + '\n'+
6790         "about the section. The program checks that the folder is empty " + '\n'+
6791         "before proceeding to make sure that no files are overwritten " + '\n'+
6792         "and to avoid mixing up project files with other files. The save " + '\n'+
6793         "folder will require approximately twice the size of the original " + '\n'+
6794         "images and 1MB for each user defined area in each section.");
6795     return helpText;
6796 }
6797
6798 public String getHelpTextChoseConversionImage()
6799 {
6800     String helpText = new String(
6801         "CHOOSE CONVERSION IMAGE"+ '\n'+ '\n'+
6802         "The conversion image is used to convert pixels to mm. This image " + '\n'+
6803         "has to be acquired using the same microscope settings as the " + '\n'+
6804         "section images and contain something of a known size, preferably " + '\n'+
6805         "a scale bar. If the microscope can't include scale bars an image " + '\n'+
6806         "of a grid or some other object of known size will work as well. " + '\n'+
6807         "The program will check to make sure that it is a readable image " + '\n'+
6808         "file of the same size as the section images." + '\n'+ '\n'+
6809         "ENTER CONVERSION VALUES"+ '\n'+
6810         "Once a project has been created the conversion panel is used " + '\n'+
6811         "to determine the number of pixels per square millimeter. The " + '\n'+
6812         "number of pixels is entered by drawing a line in the image, " + '\n'+
6813         "left click for the start point and right click for the end point. " + '\n'+
6814         "The program will calculate the distance between the points in " + '\n'+
6815         "pixels. Enter this distance in millimeters in the text box on " + '\n'+
6816         "the right side panel. Press the update conversion button when " + '\n'+
6817         "the distance is both pixels and millimeters has been entered. " + '\n'+
6818         "The program will now recalculate all areas and volumes using " + '\n'+
6819         "these settings and the number of pixels per square millimeter " + '\n'+
6820         "is reported.");

```

```

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

```

```

6877     }
6878
6879     public String getHelpTextAdjustImage()
6880     {
6881         String helpText = new String(
6882             "ADJUST IMAGE" + '\n' + '\n' +
6883             "Normally at least some sections in a project will contain folds " + '\n' +
6884             "or tears or other errors from the sectioning and staining. This " + '\n' +
6885             "can be corrected for by adding or removing tissue. Use the buttons " + '\n' +
6886             "on the right side panel to choose whether to add or remove tissue " + '\n' +
6887             "and the size used when drawing. It is also possible to change the " + '\n' +
6888             "threshold for tissue detection. This is done in the same way as in " + '\n' +
6889             "the Tissue detection panel but any changes made here will only " + '\n' +
6890             "apply to this section.");
6891         return helpText;
6892     }
6893
6894     public String getAboutText()
6895     {
6896         String aboutText = new String(
6897             "SectionToVolume version 1.0" + '\n' + '\n' +
6898             "Created by Anders Hånell in 2011." + '\n' + '\n' +
6899
6900             "SectionToVolume is free to use. Please cite the article by " + '\n' +
6901             "Hånell et al if you use SectionToVolume in your research." + '\n' + '\n' +
6902             "The functions in the program has been thoroughly tested " + '\n' +
6903             "but no guarantees are given for correct results." + '\n' + '\n' +
6904             "Send questions and suggestions to SectionToVolume@gmail.com" + '\n' + '\n' + '\n' +
6905
6906
6907             "This software is not subject to copyright protection and " + '\n' +
6908             "is in the public domain. SectionToVolume is an experimental " + '\n' +
6909             "system and the author assumes no responsibility whatsoever " + '\n' +
6910             "for its use by other parties, and makes no guarantees, " + '\n' +
6911             "expressed or implied, about its quality, reliability, " + '\n' +
6912             "or any other characteristic."
6913
6914
6915
6916             );
6917
6918         return aboutText;
6919     }
6920
6921
6922     public String getErrorTextApplyToAllInterrupted()
6923     {
6924         String errorText = new String(
6925             "AN ERROR OCCURED" + '\n' + '\n' +
6926             "Apply to all function aborted by user" + '\n' +
6927             "" + '\n' +
6928             "WARNING" + '\n' +
6929             "" + '\n' +
6930             "This can cause incorrect results." + '\n' +
6931             "Start the apply to all function again and let it finish" + '\n' +
6932             "to assure correct calculation of the results." + '\n' + '\n' + '\n'

```

```

6933         );
6934
6935         return errorText;
6936     }
6937
6938     public String getErrorTextUpdateConversionInterrupted()
6939     {
6940         String errorText = new String(
6941             "AN ERROR OCCURED" + '\n' + '\n' +
6942             "Update conversion function aborted by user" + '\n' +
6943             "" + '\n' +
6944             "WARNING" + '\n' +
6945             "" + '\n' +
6946             "This can cause incorrect results." + '\n' +
6947             "Start the update conversion function again and let it finish" + '\n' +
6948             "to assure correct calculation of the results." + '\n' + '\n' + '\n'
6949         );
6950
6951         return errorText;
6952     }
6953
6954     public String getErrorTextCreateProjectInterrupted()
6955     {
6956         String errorText = new String(
6957             "AN ERROR OCCURED" + '\n' + '\n' +
6958             "Create project function aborted by user" + '\n' +
6959             "" + '\n' +
6960             "WARNING" + '\n' +
6961             "" + '\n' +
6962             "The project will most likely not function." + '\n' +
6963             "Delete any created files in the save folder and" +
6964             "start the create project function again and let it finish" + '\n' +
6965             "to create the project." + '\n' + '\n' + '\n'
6966         );
6967
6968         return errorText;
6969     }
6970
6971     public int countNumberOfFiles(File location)
6972     {
6973         int numberOfFiles=0;
6974
6975         File[] subjectFileList = new File[1000];
6976         File[][] sectionFileList = new File[1000][1000];
6977
6978         subjectFileList = location.listFiles();
6979
6980         numberOfFiles+=subjectFileList.length;
6981
6982         for (int subjectCounter = 0; subjectCounter < subjectFileList.length; subjectCounter++)
6983         {
6984             try
6985             {
6986                 sectionFileList[subjectCounter] = subjectFileList[subjectCounter].listFiles();
6987                 numberOfFiles+=sectionFileList[subjectCounter].length;
6988             } catch (Exception error){ }

```

```

6989     }
6990 }
6991
6992     return numberOfFiles;
6993 }
6994
6995 //Tries to read the file and count the number of tissue pixels. If it fails the file is not used.
6996 public boolean fileIsValidImage(File testFile)
6997 {
6998     boolean fileIsImage = false;
6999     BufferedImage tempBufferedImage;
7000
7001     try
7002     {
7003         tempBufferedImage=ImageIO.read(testFile);
7004         WritableRaster raster = tempBufferedImage.getRaster();
7005
7006         int imageWidth=tempBufferedImage.getWidth();
7007         int imageHeight=tempBufferedImage.getHeight();
7008
7009         if (createProjectDialog1.detectedImageWidth==-1)
7010         {
7011             createProjectDialog1.detectedImageWidth=imageWidth;
7012         }
7013         else
7014         {
7015             if (createProjectDialog1.detectedImageWidth!=imageWidth)
7016                 createProjectDialog1.allImagesIsOfSameSize=false;
7017         }
7018
7019         if (createProjectDialog1.detectedImageHeight==-1)
7020         {
7021             createProjectDialog1.detectedImageHeight=imageHeight;
7022         }
7023         else
7024         {
7025             if (createProjectDialog1.detectedImageHeight!=imageHeight)
7026                 createProjectDialog1.allImagesIsOfSameSize=false;
7027         }
7028
7029         fileIsImage=true;
7030     }
7031     catch (Exception event)
7032     {
7033         fileIsImage=false;
7034     }
7035
7036     return fileIsImage;
7037 }
7038
7039 //The positions between the two points are set to true.
7040 public boolean[][] drawLineBoolean (boolean[][] isLine, Point previousPoint, Point currentPoint)
7041 {
7042     int distanceX;
7043     int distanceY;
7044

```



```

7045     int tempX;
7046     int tempY;
7047
7048     distanceX=Math.abs(previousPoint.x-currentPoint.x);
7049     distanceY=Math.abs(previousPoint.y-currentPoint.y);
7050
7051     if (distanceX>distanceY)
7052     {
7053         if (previousPoint.x<currentPoint.x)
7054         {
7055             if (previousPoint.y<=currentPoint.y)
7056             {
7057                 for (int i=1;i<=distanceX;i++)
7058                 {
7059                     tempX=previousPoint.x+i;
7060                     tempY=previousPoint.y+distanceY*i/distanceX;
7061                     isLine[tempX][tempY] = true;
7062                 }
7063             }
7064             if (previousPoint.y>currentPoint.y)
7065             {
7066                 for (int i=1;i<=distanceX;i++)
7067                 {
7068                     tempX=previousPoint.x+i;
7069                     tempY=previousPoint.y-distanceY*i/distanceX;
7070                     isLine[tempX][tempY] = true;
7071                 }
7072             }
7073         }
7074         if (previousPoint.x>currentPoint.x)
7075         {
7076             if (previousPoint.y<=currentPoint.y)
7077             {
7078                 for (int i=1;i<=distanceX;i++)
7079                 {
7080                     tempX=previousPoint.x-i;
7081                     tempY=previousPoint.y+distanceY*i/distanceX;
7082                     isLine[tempX][tempY] = true;
7083                 }
7084             }
7085             if (previousPoint.y>currentPoint.y)
7086             {
7087                 for (int i=1;i<=distanceX;i++)
7088                 {
7089                     tempX=previousPoint.x-i;
7090                     tempY=previousPoint.y-distanceY*i/distanceX;
7091                     isLine[tempX][tempY] = true;
7092                 }
7093             }
7094         }
7095     }
7096     else
7097     {
7098         if (previousPoint.y<currentPoint.y)
7099         {
7100             if (previousPoint.x<=currentPoint.x)

```

```

7101     {
7102         for (int i=1;i<=distanceY;i++)
7103         {
7104             tempY=previousPoint.y+i;
7105             tempX=previousPoint.x+distanceX*i/distanceY;
7106             isLine[tempX][tempY] = true;
7107         }
7108     }
7109     if (previousPoint.x>currentPoint.x)
7110     {
7111         for (int i=1;i<=distanceY;i++)
7112         {
7113             tempY=previousPoint.y+i;
7114             tempX=previousPoint.x-distanceX*i/distanceY;
7115             isLine[tempX][tempY] = true;
7116         }
7117     }
7118 }
7119 if (previousPoint.y>currentPoint.y)
7120 {
7121     if (previousPoint.x<=currentPoint.x)
7122     {
7123         for (int i=1;i<=distanceY;i++)
7124         {
7125             tempY=previousPoint.y-i;
7126             tempX=previousPoint.x+distanceX*i/distanceY;
7127             isLine[tempX][tempY] = true;
7128         }
7129     }
7130     if (previousPoint.x>currentPoint.x)
7131     {
7132         for (int i=1;i<=distanceY;i++)
7133         {
7134             tempY=previousPoint.y-i;
7135             tempX=previousPoint.x-distanceX*i/distanceY;
7136             isLine[tempX][tempY] = true;
7137         }
7138     }
7139 }
7140 }
7141 return isLine;
7142 }
7143 }
7144 }
7145
7146

```