CSI-5-ADP

Student Number: 3807942

Deadline: 16:00 Friday 30 April 2021

Contents

Comparative Analysis

Demo.java

```
1 package adp.elevation;
2
3 import java.awt.image.BufferedImage;
4 import java.io.File;
5 import java.io.IOException;
6 import javax.imageio.ImageIO;
7 import adp.elevation.jar.BasicSearcher;
8 import adp.elevation.jar.Searcher;
9 import adp.elevation.jar.Searcher.SearchListener;
10
11/**
12 * This class implements {@link Searcher.SearchListener} and emits all
messages
13 * on the command line output.
14 */
15public class Demo implements SearchListener {
16
17  public Demo(final File file) throws IOException {
18      final BufferedImage raster = ImageIO.read(file);
19      final Searcher searcher = new BasicSearcher(raster, Configura-
tion.side, Configuration.deviationThreshold);// ,
20
// image2);
21      searcher.runSearch(this);
22  }
23
24  @Override
25  public void information(final String message) {
26      System.out.println(message);
27  }
28
29  @Override
30  public void possibleMatch(final int position, final long elapsedTime,
final long numberOfPositionsTriedSoFar) {
31      System.out.println("Possible match at: " + position + " at " +
(elapsedTime / 1000.0) + "s ("
32              + numberOfPositionsTriedSoFar + " positions attempted)");
33  }
34
35  @Override
36  public void update(final int position, final long elapsedTime, final
long numberOfPositionsTriedSoFar) {
37      System.out.println("Searching at: " + position + " at " +
(elapsedTime / 1000.0) + "s ("
38              + numberOfPositionsTriedSoFar + " positions attempted)");
39  }
40
41  /**
42   * Set the file names to search rasters of different sizes.
43   *
44   * @param args
45   * @throws IOException
46   */
47  public static void main(final String[] args) throws IOException {
48      final File file = new File("rgbelevation/smallelevation.png");
49      // final File file = new File( "rgbelevation/bigelevation.png");
50      new Demo(file);
51  }
52
53}
```

Figure 1

```
1 package adp.elevation;
2
3 /**
4  * Convenient central location for the configuration of the size of
square
5  * regions to search for and the maximum standard deviation to allow for
a
6  * match. There is no need to change these values for the assignment.
7  */
8 public class Configuration {
9
10  public static final int side = 10;
11  public static final double deviationThreshold = 10;
12
13}
```

Figure 2

Line 47 to 51 in Figure 1 is the main method of *Demo.java* which initializes a file variable containing one of the elevation images in the *rgbelevation* folder. After, the use of *new Demo(file)* is used to initialise a *Demo* object which acts as a *SearchListener* as seen from line 15 where the *Demo* class implements *SearchListener* which is used to observe the progress and results of search objects.

Within the constructor of *Demo*, lines 17 to 22, the file is read and stored as a *BufferedImage* which is passed to a *Searcher* object through a *BasicSearcher* instance. *BasicSeacher* is a subclass of *AbstractSearcher* which takes a *BufferedImage*, an *integer* and a *double* where the image will be searched for square regions with pixel sides equal to the *integer* variable and where the elevation data doesn't exceed the pre-set standard deviation value. Both values are provided by the *Configuration* class, as depicted in Figure 2 lines 10 and 11, with the *side* and *deviationThreshold* value used being 10.

The *runSearch* method called on the *searcher* variable, on line 21 in Figure 1, performs the search on the file provided and reports the progress of the search to the *SearchListener* object passed to the *runSearch* method using the *this* keyword which in this case would be the instance of the *Demo* object which was initialized in the *main* method.

Lines 25 to 27 in Figure 1 depicts the *information* method which is invoked when passed messages such as if the search has begun, was completed or if it has been aborted. Lines 30 to 33 in Figure 1 is the *possibleMatch* method which is invoked when the *Searcher* object finds a position that may match the criteria where the method returns the position, elapsed time and number of positions that have been tried. Lines 36 to 39 in Figure 1is the *update* method which is invoked numerously by the *Searcher* object to indicate the progress of the search by returning the elapsed time and number of positions that have been tried.

SearchUI

```
1  package adp.elevation.ui;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Dimension;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.GridLayout;
9  import java.awt.Rectangle;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.image.BufferedImage;
13 import java.io.File;
14 import java.io.IOException;
15 import java.util.ArrayList;
16 import java.util.List;
17 import javax.imageio.ImageIO;
18 import javax.swing.JButton;
19 import javax.swing.JFileChooser;
20 import javax.swing.JFrame;
21 import javax.swing.JLabel;
22 import javax.swing.JPanel;
23 import javax.swing.SwingUtilities;
24 import adp.elevation.Configuration;
25 import adp.elevation.jar.BasicSearcher;
26 import adp.elevation.jar.Searcher;
27 import adp.elevation.jar.Searcher.SearchListener;
28
29 /**
30  * This class implements a basic GUI interface for Searcher
implementations.
31  * This class implements SearchListener to receive the Searcher's output
32  * information.
33  */
34 public class SearchUI extends JFrame implements SearchListener {
35  private static final long serialVersionUID = 1L;
36  private final JButton openBigButton = new JButton("Open elevation
data");
37  private final JLabel mainFilenameLabel = new JLabel();
38  private final ImagePanel mainImagePanel = new ImagePanel();
39  private final JFileChooser chooser = new JFileChooser();
40  private final JLabel outputLabel = new JLabel("information");
41  private final JButton startButton = new JButton("Start");
42  private Searcher searcher;
43  private BufferedImage raster;
44
45  /**
46   * Construct an SearchUI and set it visible.
47   */
48  public SearchUI() {
49      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // kill the
application on closing the window
50      final JPanel mainFilePanel = new JPanel(new BorderLayout());
51      mainFilePanel.add(this.openBigButton, BorderLayout.WEST);
52      mainFilePanel.add(this.mainFilenameLabel, BorderLayout.CENTER);
53
54      final JPanel topPanel = new JPanel(new GridLayout(0, 1));
55      topPanel.add(mainFilePanel);
56
57      final JPanel imagePanel = new JPanel(new BorderLayout());
58      imagePanel.add(this.mainImagePanel, BorderLayout.CENTER);
```

Figure 3

```
59
60      final JPanel bottomPanel = new JPanel(new BorderLayout());
61      bottomPanel.add(this.outputLabel, BorderLayout.CENTER);
62      bottomPanel.add(this.startButton, BorderLayout.SOUTH);
63
64      final JPanel mainPanel = new JPanel(new BorderLayout());
65      mainPanel.add(topPanel, BorderLayout.NORTH);
66      mainPanel.add(imagePanel, BorderLayout.CENTER);
67      mainPanel.add(bottomPanel, BorderLayout.SOUTH);
68
69      this.openBigButton.addActionListener(new ActionListener() {
70          @Override
71          public void actionPerformed(final ActionEvent ev) {
72              if (SearchUI.this.chooser.showOpenDialog(SearchUI.this) ==
JFileChooser.APPROVE_OPTION) {
73                  final File file =
SearchUI.this.chooser.getSelectedFile();
74
SearchUI.this.mainFilenameLabel.setText(file.getName());
75                  try {
76                      SearchUI.this.raster = ImageIO.read(file);
77                  } catch (final IOException e) {
78                      // TODO Auto-generated catch block
79                      e.printStackTrace();
80                  }
81                  SearchUI.this.mainImagePanel.resetHighlights();
82
SearchUI.this.mainImagePanel.setImage(SearchUI.this.raster);
83                  pack();
84                  SearchUI.this.mainImagePanel.repaint();
85              }
86          }
87      });
88
89      this.startButton.addActionListener(new ActionListener() {
90          @Override
91          public void actionPerformed(final ActionEvent ev) {
92              runSearch();
93          }
94      });
95
96      this.chooser.setMultiSelectionEnabled(false);
97      this.chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
98      this.chooser.setCurrentDirectory(new File("rgbelevation"));
99      add(mainPanel);
100     pack();
101     setVisible(true);
102 }
103
104 /**
105  * Clears output label and runs the search by calling
106  * {@link Searcher#runSearch(SearchListener)}.
107  */
108 private void runSearch() {
109     this.searcher = new BasicSearcher(this.raster, Configuration.side,
Configuration.deviationThreshold);
110     this.outputLabel.setText("information");
111     this.searcher.runSearch(this);
112 }
113
114 /**
```

Figure 4

```
115  * Implements {@link SearchListener#information(String)} by displaying
the
116  * information in the UI output label.
117  */
118 @Override
119 public void information(final String message) {
120     this.outputLabel.setText(message + "\n");
121 }
122
123 /**
124  * Implements {@link SearchListener#possibleMatch(int, long, long)} by
125  * displaying the information in the UI output label.
126  */
127 @Override
128 public void possibleMatch(final int position, final long elapsedTime,
final long positionsTriedSoFar) {
129     final int x = position % this.raster.getWidth();
130     final int y = position / this.raster.getWidth();
131     this.outputLabel.setText("Possible match at: [" + x + "," + y + "]
at " + (elapsedTime / 1000.0) + "s ("
132             + positionsTriedSoFar + " positions attempted)\n");
133     final Rectangle r = new Rectangle(x, y, Configuration.side,
Configuration.side);
134     this.mainImagePanel.addHighlight(r);
135 }
136
137 @Override
138 public void update(final int position, final long elapsedTime, final
long positionsTriedSoFar) {
139     final int x = position % this.raster.getWidth();
140     final int y = position / this.raster.getWidth();
141     this.outputLabel.setText("Update at: [" + x + "," + y + "] at " +
(elapsedTime / 1000.0) + "s ("
142             + positionsTriedSoFar + " positions attempted)\n");
143 }
144
145 private static void launch() {
146     new SearchUI();
147 }
148
149 private static class ImagePanel extends JPanel {
150     private static final long serialVersionUID = 1L;
151     private BufferedImage image;
152     private final List<Rectangle> highlights = new
ArrayList<Rectangle>();
153     public void setImage(final BufferedImage image) {
154         this.image = image;
155         double scale = 1;
156         if (image.getWidth() >= image.getHeight()) {
157             if (image.getWidth() > 800) {
158                 scale = 800.0 / image.getWidth();
159             }
160         } else {
161             if (image.getHeight() > 800) {
162                 scale = 800.0 / image.getHeight();
163             }
164         }
165         final Dimension d = new Dimension((int)
Math.ceil(image.getWidth() * scale),
166                 (int) Math.ceil(image.getHeight() * scale));
167         // System.out.println( d);
```

Figure 5

```
168             setPreferredSize(d);
169             invalidate();
170             repaint();
171         }
172
173         public void addHighlight(final Rectangle r) {
174             synchronized (this.highlights) {
175                 this.highlights.add(r);
176             }
177             repaint();
178         }
179
180         public void resetHighlights() {
181             synchronized (this.highlights) {
182                 this.highlights.clear();
183             }
184             repaint();
185         }
186
187         @Override
188         public void paintComponent(Graphics g) {
189             if (this.image != null) {
190                 g = g.create();
191                 final double scale = getWidth() / (double)
this.image.getWidth();
192                 // System.out.println( scale + "!");
193                 g.drawImage(this.image, 0, 0, getWidth(), (int)
(this.image.getHeight() * scale), this);
194                 // System.out.println( ">>>" + completed);
195                 g.setColor(Color.YELLOW);
196                 synchronized (this.highlights) {
197                     for (final Rectangle r : this.highlights) {
198                         final Rectangle s = new Rectangle((int) (r.x *
scale), (int) (r.y * scale),
199                                 (int) (r.width * scale), (int) (r.height *
scale));
200                         ((Graphics2D) g).draw(s);
201                         // System.out.println( r + " >> " + s);
202                     }
203                 }
204             }
205         }
206
207 }
208
209 public static void main(final String[] args) {
210     SwingUtilities.invokeLater(new Runnable() {
211         @Override
212         public void run() {
213             launch();
214         }
215     });
216 }
217}
```

Figure 6

Lines 209 to 216 in Figure 6 is the *main* method of *SearchUI.java* where the *launch* method is executed on the main *AWT* thread using an anonymous inner class that is separate from the thread the

java program executes on which improves execution time for the application. Lines 145 to 147 in Figure 5 is the *launch* method which instantiates a *SearchUI* instance which is a class that extends *JFrame* while implementing a *SearchListener* as depicted on line 34 in Figure 3.

After most of the private variables have been instantiated and assigned from within the constructor of *SearchUI*, between lines 48 and 102, the logic for button actions has been implemented using anonymous inner classes via the utilization of *ActionListeners*. When the *openBigButton* is clicked, a file chooser object is instantiated where a file can be opened. Once a file has been selected, the *mainFilenameLabel* variable is updated to store the name of the file. If the file was an image, it is stored as a *BufferedImage* where the image will then be displayed via the *mainImagePanel* variable and any highlighted regions from the program being executed on a previous image will be reset. An anonymous inner class is used to execute the *runSearch* method if the *ActionListener* on the *startButton* detects that it has been clicked.

Lines 108 to 112 in Figure 4 is the *runSearch* method which instantiates a *BasicSearcher* object similar to that found in Figure 1on line 19. The *outputLabel* variable is updated to display "Information" while the *searcher* object executes *runSearch* while reporting its progress to the *SearchListener* from *SearchUI* via the *this* keyword thus the *runSearch* method executed on the *searcher* keyword executes *runSearch* from the *AbstractSearcher* superclass of *BasicSearcher*.

The *update* method in Figure 5 is similar to the *update* method in Figure 1 however, the position is calculated from the image that is being displayed and produces output from the coordinates on the image. Similarly, the *information* methods in Figure 5 are similar in Figure 1 however, the *information* method in Figure 5 updates a text label on the user interface directly whilst the *information* method in Figure 1 produces command line output. Furthermore, *possibleMatch* method in Figure 5 is similar to the instance present in Figure 1 however, a *Rectangle* variable is instantiated with the position of the *possibleMatch* along with the *ConfigurationSide* to act as the width and height of the *Rectangle* where the *ConfigurationSide* variable originated from the *Configuration* class depicted in Figure 2. The *addHighlight* method is then executed on the *mainImagePanel* variable using the instantiated *Rectangle*.

Lines 149 to 207 is the nested *ImagePanel* class used for the *mainImagePanel* variable. When the variable is passed an image, the size of the panel is set according to the dimensions of the image via the *setImage* method. When a new image is set, the *resetHighlights* method is executed which synchronizes on the *highlights* array list to stop other areas of the program from accessing the *highlights* list and clears the *highlights* list then *repaint* is used to update the area the image is being displayed in. When the *addHighlights* method is executed, a similar synchronization is used on the *highlights* array list when adding a new *Rectangle* object to the array. Furthermore, in the *paintComponent* method, the *highlights* array list is synchronized before drawing the rectangular areas of possible matches on top of the image. Synchronizing on the *highlights* array list improves thread-safety by reducing access to the array list when it is in use thus improving the coherency of data in memory.

Comparisons

Progress while running

For the *Demo* application progress is easier to interpret as the console is continually updating when outputting new possible matches and information about where the program is currently searching however, while the program is running it is difficult to read through the output as it is constantly updating. Comparatively, in the *SearchUI* application, progress is difficult to interpret as the program doesn't update to show any information besides what is currently displayed. For instance, in Figure 5 the *outputLabel* variable should update following the progress of the *Search* object for possible matches and updates however, in practice, after the *startButton* is clicked the button remains in a locked state and the label storing "Information" isn't updated until the search is finished as seen in Figure 7 and Figure 8.

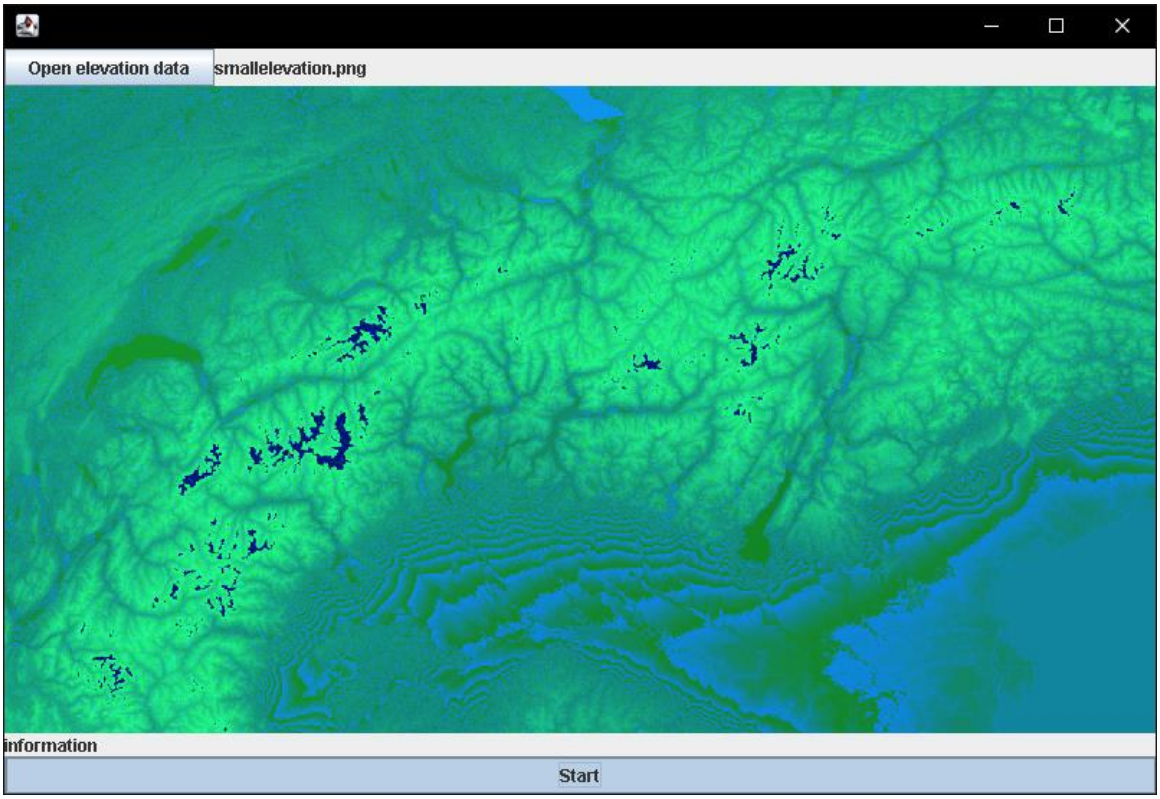

Figure 7

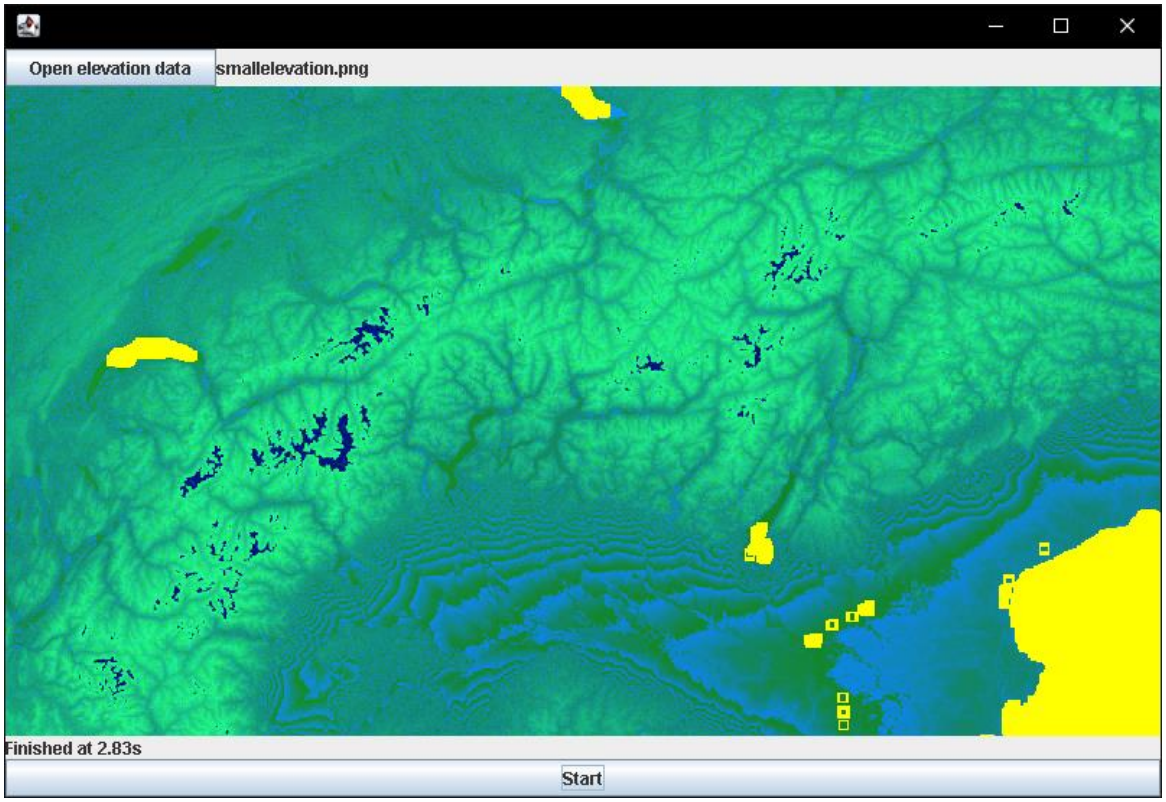

Figure 8

Cancelling the operation

For the *Demo* application, the program has no way to cancel the execution of the operation besides terminating the program thus, after terminating the program doesn't output any errors as seen in Figure 9 where console output has been halted after the program has been forcefully terminated. For the *SearchUI* program, trying to cancel the operation by clicking the close button doesn't yield results until the program has finished carrying out the search operation however, forcefully terminating the application via the development environment stops the application from executing. As both programs implement *BasicSearcher*, the program not producing errors or output due to abrupt termination may be due to *BasicSearcher* having a lack of support for a cancel mechanism.

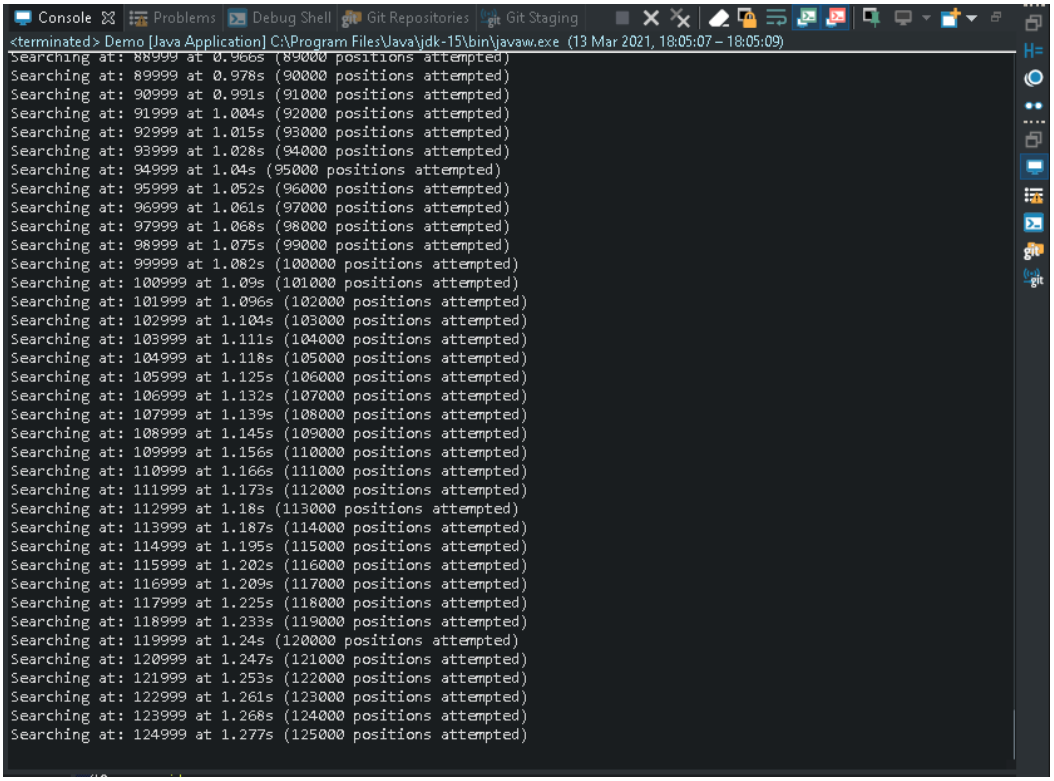

Figure 9

Threads of execution

The *Demo* application executes on one thread thus all the code executes concurrently as seen in Figure 10 however, in *SearchUI* the program executes on two threads, one thread for the java application and the *AWT-EventQueue* thread as seen in Figure 11. As *Demo* executes on one thread while *SearchUI* executes on two threads, the *SearchUI* application can execute faster than the *Demo* application but, in both cases, larger maps take a longer amount of time to search than smaller maps. This is evidenced by *SearchUI* executing in2.83 seconds as seen in Figure 8 while *Demo* takes 5.09 seconds to execute as seen in Figure 12 even though they are executing the same search operation. Furthermore, although *SearchUI* improves thread-safety by synchronising on the *highlights* array list as seen in Figure 6, the interface is nonresponsive when using a larger map to which the *startButton* remains in a locked state which could be caused by the calculations and updates to the interface all occurring on the same *AWT-EventQueue* thread. Lastly, Figure 5 depict logic for the *outputLabel* updating while the program is executing where the label remains unchanged during execution as seen in Figure 7.

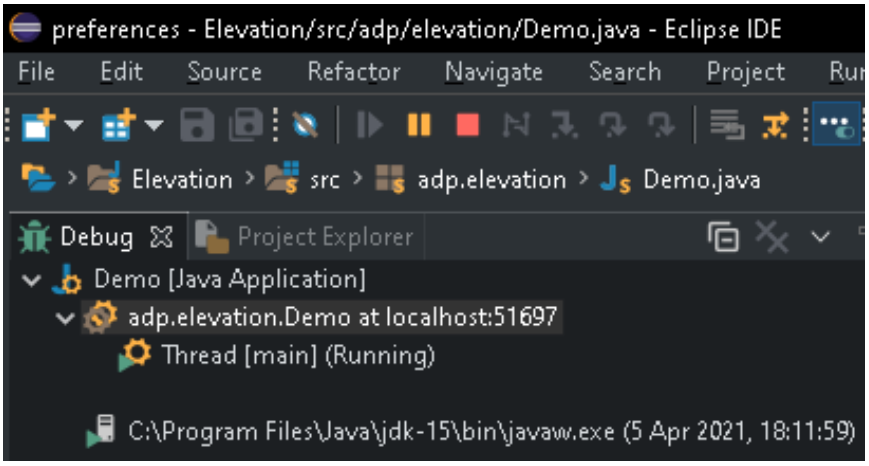

Figure 10

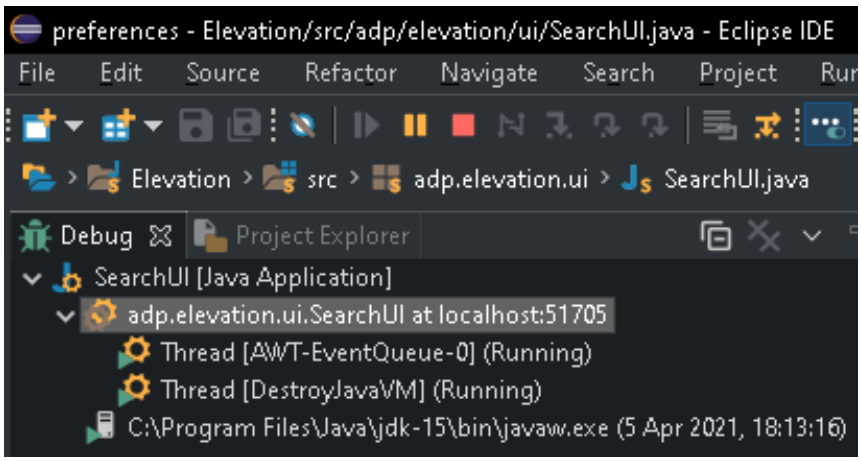

Figure 11

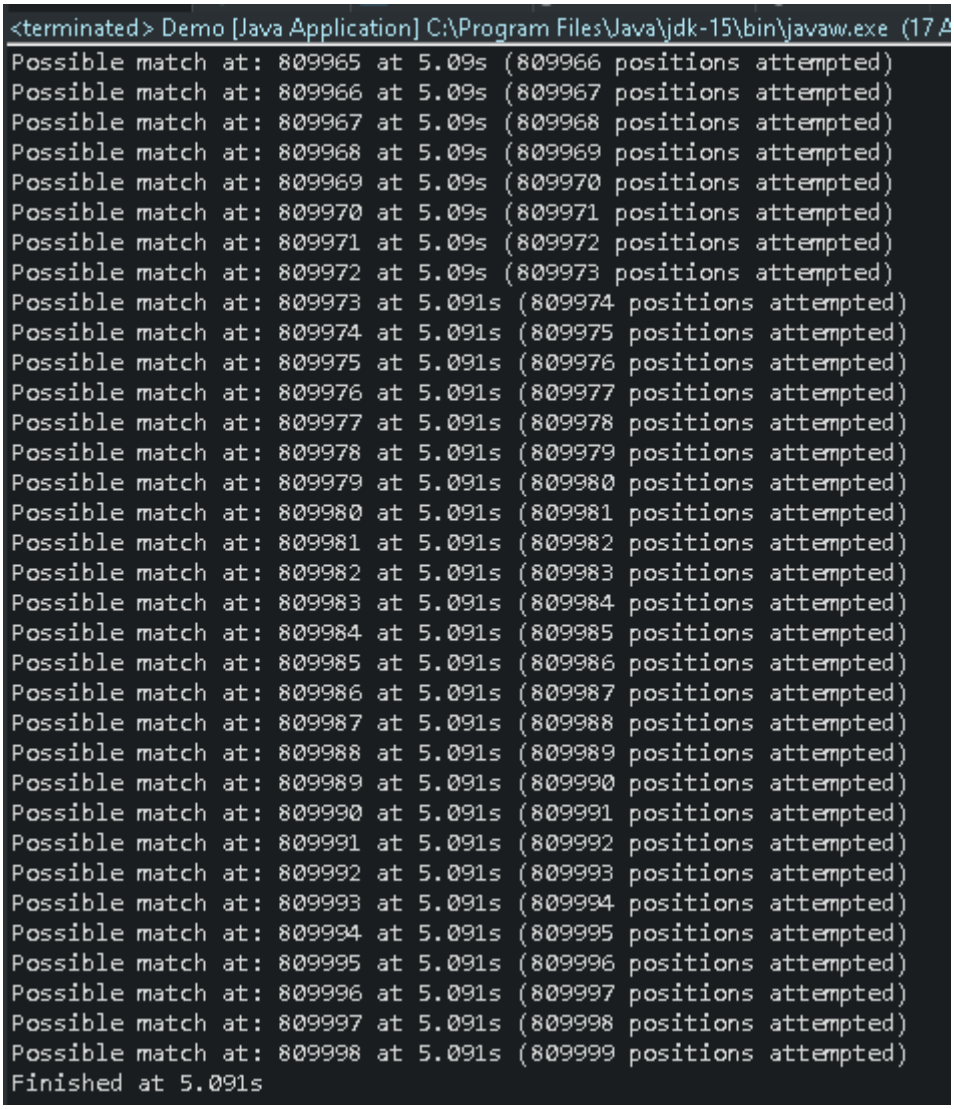
<terminated> Demo [Java Application] C:\Program Files\Java\jdk-15\bin\javaw.exe (17 A
Possible match at: 809965 at 5.09s (809966 positions attempted)
Possible match at: 809966 at 5.09s (809967 positions attempted)
Possible match at: 809967 at 5.09s (809968 positions attempted)
Possible match at: 809968 at 5.09s (809969 positions attempted)
Possible match at: 809969 at 5.09s (809970 positions attempted)
Possible match at: 809970 at 5.09s (809971 positions attempted)
Possible match at: 809971 at 5.09s (809972 positions attempted)
Possible match at: 809972 at 5.09s (809973 positions attempted)
Possible match at: 809973 at 5.091s (809974 positions attempted)
Possible match at: 809974 at 5.091s (809975 positions attempted)
Possible match at: 809975 at 5.091s (809976 positions attempted)
Possible match at: 809976 at 5.091s (809977 positions attempted)
Possible match at: 809977 at 5.091s (809978 positions attempted)
Possible match at: 809978 at 5.091s (809979 positions attempted)
Possible match at: 809979 at 5.091s (809980 positions attempted)
Possible match at: 809980 at 5.091s (809981 positions attempted)
Possible match at: 809981 at 5.091s (809982 positions attempted)
Possible match at: 809982 at 5.091s (809983 positions attempted)
Possible match at: 809983 at 5.091s (809984 positions attempted)
Possible match at: 809984 at 5.091s (809985 positions attempted)
Possible match at: 809985 at 5.091s (809986 positions attempted)
Possible match at: 809986 at 5.091s (809987 positions attempted)
Possible match at: 809987 at 5.091s (809988 positions attempted)
Possible match at: 809988 at 5.091s (809989 positions attempted)
Possible match at: 809989 at 5.091s (809990 positions attempted)
Possible match at: 809990 at 5.091s (809991 positions attempted)
Possible match at: 809991 at 5.091s (809992 positions attempted)
Possible match at: 809992 at 5.091s (809993 positions attempted)
Possible match at: 809993 at 5.091s (809994 positions attempted)
Possible match at: 809994 at 5.091s (809995 positions attempted)
Possible match at: 809995 at 5.091s (809996 positions attempted)
Possible match at: 809996 at 5.091s (809997 positions attempted)
Possible match at: 809997 at 5.091s (809998 positions attempted)
Possible match at: 809998 at 5.091s (809999 positions attempted)
Finished at 5.091s

Figure 12

SearchUIEnhancement & commentary

SearchUIEnhancement

```
1  package adp.elevation.ui;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Dimension;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.GridLayout;
9  import java.awt.Rectangle;
10 import java.awt.image.BufferedImage;
11 import java.io.File;
12 import java.io.IOException;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.concurrent.CancellationException;
16 import java.util.concurrent.ForkJoinPool;
17 import java.util.concurrent.ForkJoinTask;
18
19 import javax.imageio.ImageIO;
20 import javax.swing.JButton;
21 import javax.swing.JCheckBox;
22 import javax.swing.JFileChooser;
23 import javax.swing.JFrame;
24 import javax.swing.JLabel;
25 import javax.swing.JPanel;
26 import javax.swing.JProgressBar;
27 import javax.swing.SwingUtilities;
28
29 import adp.elevation.Configuration;
30 import adp.elevation.jar.Searcher;
31 import adp.elevation.jar.Searcher.SearchCancelledException;
32 import adp.elevation.jar.Searcher.SearchListener;
33
34 public class SearchUIEnhancement extends JFrame implements SearchLis-
tener {
35  private static final long serialVersionUID = 1L;
36
37  private final JButton openBigButton = new JButton("Open elevation
data");
38  private final JLabel mainFilenameLabel = new JLabel();
39
40  private final ImagePanel mainImagePanel = new ImagePanel();
41
42  private final JFileChooser chooser = new JFileChooser();
43
44  private final JLabel outputLabel = new JLabel("information");
45  private final JButton startButton = new JButton("Start");
46  private final JButton cancelButton = new JButton("Cancel");
47
48  private final JProgressBar progress = new JProgressBar();
49  private final JCheckBox isParallel = new JCheckBox("Run in parallel");
50
51  private volatile long startTime;
52  private volatile Searcher searcher;
53  private volatile Thread running;
54  private volatile ForkJoinPool pool = null;
55  private volatile BufferedImage raster;
56
57  /**
58   * Construct an SearchUIEnhancement and set it visible.
59   */
```

Figure 13

```
60  public SearchUIEnhancement() {
61      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // kill the appli-
cation on closing the window
62
63      final JPanel mainFilePanel = new JPanel(new BorderLayout());
64      mainFilePanel.add(this.openBigButton, BorderLayout.WEST);
65      mainFilePanel.add(this.mainFilenameLabel, BorderLayout.CENTER);
66      mainFilePanel.add(this.isParallel, BorderLayout.EAST);
67      final JPanel topPanel = new JPanel(new GridLayout(0, 1));
68      topPanel.add(mainFilePanel);
69
70      final JPanel imagePanel = new JPanel(new BorderLayout());
71      imagePanel.add(mainImagePanel, BorderLayout.CENTER);
72
73      final JPanel buttonPanel = new JPanel(new GridLayout(1, 2));
74      buttonPanel.add(this.startButton);
75      buttonPanel.add(this.cancelButton);
76
77      final JPanel bottomPanel = new JPanel(new BorderLayout());
78      bottomPanel.add(this.progress, BorderLayout.NORTH);
79      bottomPanel.add(this.outputLabel, BorderLayout.CENTER);
80      bottomPanel.add(buttonPanel, BorderLayout.SOUTH);
81
82      final JPanel mainPanel = new JPanel(new BorderLayout());
83
84      mainPanel.add(topPanel, BorderLayout.NORTH);
85      mainPanel.add(imagePanel, BorderLayout.CENTER);
86      mainPanel.add(bottomPanel, BorderLayout.SOUTH);
87
88      // w6 tutorial
89      this.openBigButton.addActionListener(ev -> {
90          if (this.chooser.showOpenDialog(this) == JFileChooser.AP-
PROVE_OPTION) {
91              final File file = this.chooser.getSelectedFile();
92              this.mainFilenameLabel.setText(file.getName());
93              try {
94                  this.raster = ImageIO.read(file);
95              } catch (final IOException e) {
96                  // TODO Auto-generated catch block
97                  e.printStackTrace();
98              }
99              mainImagePanel.resetHighlights();
100             mainImagePanel.setImage(this.raster);
101             pack();
102             mainImagePanel.repaint();
103         }
104     });
105
106     this.startButton.addActionListener(ev -> {
107         if (this.raster != null) {
108             mainImagePanel.resetHighlights();
109             mainImagePanel.setImage(this.raster);
110             pack();
111             mainImagePanel.repaint();
112             if (this.isParallel.isSelected()) {
113                 new Thread(() -> runParallelSearch()).start();
114             } else {
115                 new Thread(() -> runSearch()).start();
116             }
117         }
118     });
```

Figure 14

```
119
120     this.cancelButton.addActionListener(ev -> {
121         if (this.searcher != null) {
122             try {
123                 this.searcher.cancel();
124             } catch (SearchCancelledException SCE) {
125                 if (pool != null) {
126                     pool.shutdownNow();
127                 }
128                 this.running.interrupt();
129             }
130         }
131     });
132
133     this.chooser.setMultiSelectionEnabled(false);
134     this.chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
135     this.chooser.setCurrentDirectory(new File("rgbelevation"));
136
137     add(mainPanel);
138     pack();
139     setVisible(true);
140 }
141
142 /**
143  * Clears output label and runs the search by calling
144  * {@link Searcher#runSearch(SearchListener)}.
145  */
146 private void runSearch() {
147     this.running = Thread.currentThread();
148     this.searcher = new DevelopedSearcher(this.raster, Configura-
tion.side, Configuration.deviationThreshold);
149     new Thread(() -> updateProgress()).start();
150     information("information");
151     this.progress.setValue(0);
152     this.progress.setStringPainted(true);
153     this.searcher.runSearch(this);
154 }
155
156 private <T> void runParallelSearch() {
157     // TODO Auto-generated method stub
158     startTime = System.currentTimeMillis();
159     this.running = Thread.currentThread();
160     this.searcher = new RASearcher(this.raster, 0, (this.ras-
ter.getWidth() * this.raster.getHeight()) - 1, this,
161             startTime, 0);
162     new Thread(() -> updateProgress()).start();
163     information("information");
164     this.progress.setValue(0);
165     this.progress.setStringPainted(true);
166     pool = new ForkJoinPool(); // fixes error when trying to allocate
jobs after the pool has been shutdown
167     try {
168         pool.invoke((ForkJoinTask<?>) this.searcher);
169     } catch (CancellationException CE) {
170         information("Aborted\n");
171     }
172     pool.shutdown();
173     information("Finished at " + ((System.currentTimeMillis() - start-
Time) / 1000.0) + "s \n");
174 }
175
```

Figure 15

```
176 private void updateProgress() {
177     float currentProgress = this.searcher.numberOfPositionsTriedSo-
Far();
178     float total = this.searcher.numberOfPositionsToTry();
179 
180     while ((currentProgress < total) && (!this.running.isInter-
rupted())) {
181         try {
182             currentProgress = this.searcher.numberOfPositionsTriedSo-
Far();
183             float percent = (currentProgress / total) * 100;
184             this.progress.setValue(Math.round(percent));
185             Thread.sleep(1000);
186         } catch (InterruptedException e) {
187             // TODO Auto-generated catch block
188             e.printStackTrace();
189         }
190     }
191 }
192 
193 /**
194  * Implements {@link SearchListener#information(String)} by displaying
the
195  * information in the UI output label.
196  */
197 @Override
198 public synchronized void information(final String message) {
199     if (!this.running.isInterrupted()) {
200         SwingUtilities.invokeLater(() -> this.outputLabel.setText(mes-
sage + "\n"));
201         // this.this.outputLabel.setText(message + "\n");
202     } else {
203         SwingUtilities.invokeLater(() -> this.out-
putLabel.setText("Aborted\n" + "\n"));
204     }
205 }
206 
207 /**
208  * Implements {@link SearchListener#possibleMatch(int, long, long)} by
209  * displaying the information in the UI output label.
210  */
211 @Override
212 public synchronized void possibleMatch(final int position, final long
elapsedTime, final long positionsTriedSoFar) {
213     final int x = position % this.raster.getWidth();
214     final int y = position / this.raster.getWidth();
215     information("Possible match at: [" + x + "," + y + "] at " +
(elapsedTime / 1000.0) + "s ("
216             + positionsTriedSoFar + " positions attempted)\n");
217     if (!this.running.isInterrupted()) {
218         final Rectangle r = new Rectangle(x, y, Configuration.side,
Configuration.side);
219         mainImagePanel.addHighlight(r);
220     }
221 }
222 
223 @Override
224 public synchronized void update(final int position, final long
elapsedTime, final long positionsTriedSoFar) {
225     final int x = position % this.raster.getWidth();
226     final int y = position / this.raster.getWidth();
```

Figure 16

```
227     information("Update at: [" + x + "," + y + "] at " + (elapsedTime /
1000.0) + "s (" + positionsTriedSoFar
228             + " positions attempted)\n");
229 }
230
231 private synchronized static void launch() {
232     new SearchUIEnhancement();
233 }
234
235 private static class ImagePanel extends JPanel {
236     private static final long serialVersionUID = 1L;
237
238     private BufferedImage image;
239
240     private final List<Rectangle> highlights = new ArrayList<Rectan-
gle>();
241
242     public void setImage(final BufferedImage image) {
243         this.image = image;
244
245         double scale = 1;
246
247         if (image.getWidth() >= image.getHeight()) {
248             if (image.getWidth() > 800) {
249                 scale = 800.0 / image.getWidth();
250             }
251         } else {
252             if (image.getHeight() > 800) {
253                 scale = 800.0 / image.getHeight();
254             }
255         }
256         final Dimension d = new Dimension((int) Math.ceil(im-
age.getWidth() * scale),
257                 (int) Math.ceil(image.getHeight() * scale));
258         // System.out.println( d);
259         setPreferredSize(d);
260
261         invalidate();
262         repaint();
263     }
264
265     public void addHighlight(final Rectangle r) {
266         synchronized (this.highlights) {
267             this.highlights.add(r);
268         }
269         repaint();
270     }
271
272     public void resetHighlights() {
273         synchronized (this.highlights) {
274             this.highlights.clear();
275         }
276         repaint();
277     }
278
279     @Override
280     public void paintComponent(Graphics g) {
281         if (this.image != null) {
282             g = g.create();
283             final double scale = getWidth() / (double) this.im-
age.getWidth();
```

Figure 17

```
284             // System.out.println( scale + "!");
285             g.drawImage(this.image, 0, 0, getWidth(), (int) (this.im-
age.getHeight() * scale), this);
286             // System.out.println( ">>>" + completed);
287             g.setColor(Color.YELLOW);
288             synchronized (this.highlights) {
289                 for (final Rectangle r : this.highlights) {
290                     final Rectangle s = new Rectangle((int) (r.x *
scale), (int) (r.y * scale),
291                                 (int) (r.width * scale), (int) (r.height *
scale));
292                     ((Graphics2D) g).draw(s);
293                     // System.out.println( r + " >> " + s);
294                 }
295             }
296         }
297     }
298
299 }
300
301 public static void main(final String[] args) {
302     SwingUtilities.invokeLater(() -> launch());
303 }
304}
```

Figure 18

DevelopedSearcher

```
1 package adp.elevation.ui;
2
3 import java.awt.image.BufferedImage;
4
5 import adp.elevation.jar.AbstractSearcher;
6
7 public class DevelopedSearcher extends AbstractSearcher {
8
9   public DevelopedSearcher(BufferedImage raster, int side, double devia-
tionThreshold) {
10      super(raster, side, deviationThreshold);
11      // TODO Auto-generated constructor stub
12  }
13
14  @Override
15  public synchronized void cancel() {
16      throw new SearchCancelledException();
17  }
18
19}
```

Figure 19

All source code can be found at (Bhatti, 2021).

Objectives

Searcher Thread

Figure 13 to Figure 18 contains the code for the *SearchUIEnhancement* class. Instead of using an anonymous inner class to queue the *launch* method on the *AWTEventQueue* thread as seen in Figure 6 lines 210 to 215, a lambda expression has been implemented with the aid of (Oracle, 1995) in Figure 18 on line 302 which effectively performs the same operation but in a more concise manner.

Similarly, lambda expressions have been used in Figure 14 and Figure 15 to carry out the same button operations like those found in Figure 4.

In Figure 14 on lines 106 to 118, a condition is used to check if the *raster* variable currently contains an image to resolve a *NullPointerException* being thrown due to attempting to run a search when the raster doesn't contain an image. Lines 99 to 102 have been repeated inside of the lambda expression for the start button so that the raster can be repopulated with highlights for points of elevation if a user attempts to run a search after a search has already finished. The *isParallel* variable is a checkbox used to store whether a user wants to run a parallel search where, in this instance, the variable will store false as we will be running a non-parallelised search thus the *else* condition on line 114 to 116 will be triggered to execute the *runSearch* method. Line 115 calls the *runSearch* method from within an anonymous thread as depicted in (Child, 2020) which allows the *runSearch* method to execute on a dedicated worker thread so that the user interface doesn't freeze.

Progress bar

In Figure 15 on line 147, the *running* variable stores the current thread the *runSearch* method is executing on as a variable that can be accessed later. The *searcher* variable stores an instance of *DevelopedSearcher* which takes similar arguments to those found in Figure 4 for the *BasicSearcher* instance however, in Figure 19 we see that these parameters are passed to the *AbstractSearcher* class via the *super* method on line 10. Line 149 in Figure 15 executes the *updateProgress* method on a separate thread to monitor the progress of the *Searcher* object wherein Figure 16 on lines 176 to 191, the *currentProgress* is calculated and updated each second on the progress bar. If line 185 wasn't included, the progress bar would update more smoothly as the *currentProgress* of the *Searcher* would be polled constantly however, as stated in the specification line 185 pauses the execution of the thread handling the progress bar so that the progress bar updates each second where (Oracle, 1993) was used to gain a better understanding of the methods that manipulate the *JProgressBar* variable.

Updating text area

Similar to Figure 5, *information, possibleMatch* and *update* methods are present as seen in Figure 16 and Figure 17 however, to allow the text area to be updated with output from the *Searcher* object, methods found in (Child, 2021) have been used to queue updates made to the *outputLabel* from the *Searcher* object on the *AWTEventQueue* thread as seen in Figure 16 on line 200. Instead of passing updates made from the *possibleMatch* and *update* methods directly to the *SwingUtilities.invokeLater* method, we have called the *information* method from inside both of these methods to handle the queueing of updates to the *outputLabel* onto the *AWTEventQueue* thread.

Cancel operation

Unlike in the default *SearchUI* class where clicking the close button during a search won't close the application window until the search has finished, the interface no longer freezes during execution thus when a user clicks the close button the application immediately closes regardless of if a search has finished or not as the interface, search and progress bar are all executing on dedicated threads in *SearchUIEnhancement*. Lines 120 to 131 in Figure 14 handle the cancel operation. A check is performed on *Searcher* to avoid a *NullPointerException* if a user clicks the cancel button when a search isn't executing. Line 123 calls the *cancel* method in *DevelopedSearcher* which throws a *SearchCancelledException* which is caught in the *catch* block on line 124 to then execute an *interrupt* on the thread running the search which causes the progress bar and raster to stop updating due to the *isInterrupted* check being performed on line 180, 199 and 217 in Figure 16. Instead of throwing a *SearchCancelledException*, we could have bypassed the *cancel* method entirely by interrupting the *running* thread directly from the cancel button to achieve the same result. We favoured the implementation that throws a *SearchCancelledException* as executing an interrupt directly could be scheduled by the CPU at a later time however, catching the thrown exception and then interrupting the searcher thread ensures that the CPU schedules the interrupt immediately as control is passed from the

searcher thread to the *AWTEventQueue* thread when the exception is caught. Furthermore, calling the *cancel* method allows for future developers to modify how the *cancel* operation will be handled via their class which subclasses the *AbstractSearcher* class without needing to manipulate the *SearchUIEnhancement* class. Lastly, the *isInterrupted* check on line 199 would execute the code on line 203 to display an "Aborted" message in the text area on the user interface as seen in Figure 20. (Oracle, 1995) was used to place the buttons, text area and progress bar under the raster.

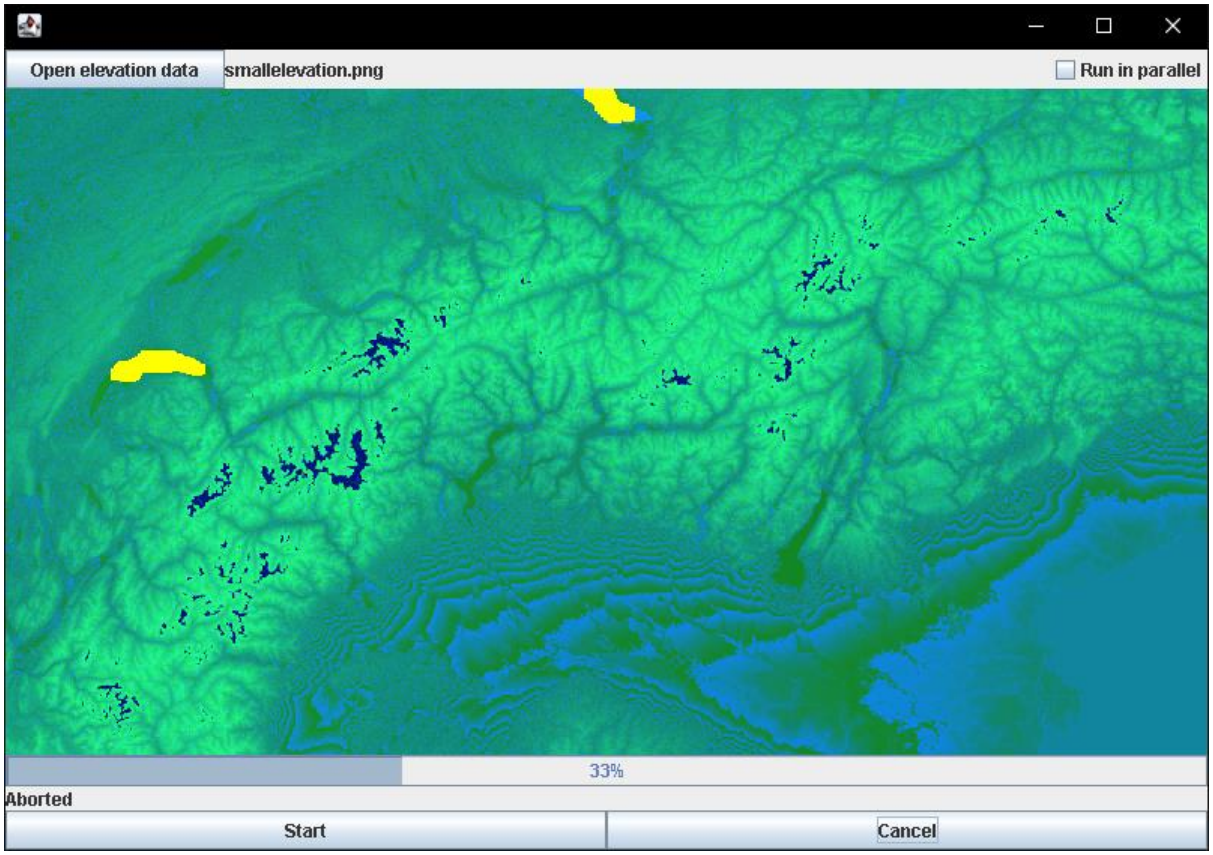

Figure 20

Thread-safety

As described in (Child, 2021), the final keyword is used to create immutable objects in *SearchUIEnhancement* to ensure the value of variables can't be changed after being instantiated while the volatile keyword is used for the *raster*, *searcher* and currently *running* thread to allow for thread-safety when each variable is being manipulated or accessed by multiple threads. Furthermore, the *synchronized* keyword has been used in multiple methods in *SearchUIEnhancement* and on the *cancel* method in *DevelopedSearcher* so that only one thread may access the method being called which resolves concurrency issues when multiple threads modify a specific variable such as the *highlights* or *outputLabel* variable.

Parallelised Searcher & commentary

RASearcher

```
1  package adp.elevation.ui;
2
3  import java.awt.image.BufferedImage;
4  import java.util.concurrent.RecursiveAction;
5
6  import adp.elevation.Configuration;
7  import adp.elevation.jar.Searcher;
8
9  public class RASearcher extends RecursiveAction implements Searcher {
10  private static final long serialVersionUID = 1L;
11  private final BufferedImage raster;
12  private final int startPos;
13  private final int endPos;
14  private final int side = Configuration.side;
15  private final double Threshold = Configuration.deviationThreshold;
16  private final SearchUIEnhancement masterListener;
17  private final long startTime;
18  private volatile Searcher search1;
19  private volatile Searcher search2;
20  private volatile int currentPos;
21  private volatile int counter;
22
23  public RASearcher(BufferedImage raster, int startPos, int endPos,
SearchUIEnhancement masterListener,
24          long startTime, int counter) {
25      // TODO Auto-generated constructor stub
26      this.raster = raster;
27      this.startPos = startPos;
28      this.currentPos = startPos;
29      this.endPos = endPos;
30      this.masterListener = masterListener;
31      this.startTime = startTime;
32      this.counter = counter;
33  }
34
35  @Override
36  protected void compute() {
37      // TODO Auto-generated method stub
38      if (this.numberOfPositionsToTry() < this.Threshold) {
39          runSearch(new SearchListener() {
40              @Override
41              public synchronized void update(int position, long
elapsedTime, long positionsTriedSoFar) {
42                  // TODO Auto-generated method stub
43                  return;
44              }
45
46              @Override
47              public synchronized void possibleMatch(int position, long
elapsedTime, long positionsTriedSoFar) {
48                  // TODO Auto-generated method stub
49                  masterListener.possibleMatch(position, elapsedTime,
positionsTriedSoFar);
50              }
51
52              @Override
53              public synchronized void information(String message) {
54                  // TODO Auto-generated method stub
55                  return;
56              }
57          });
```

Figure 21

```
58              return;
59          }
60
61          int split = this.numberOfPositionsToTry() / 2;
62          this.search1 = new RASearcher(this.raster, this.startPos,
this.endPos - split, this.masterListener, this.startTime,
63                  this.counter);
64          this.search2 = new RASearcher(this.raster, this.startPos + split,
this.endPos, this.masterListener, this.startTime,
65                  this.counter);
66          invokeAll((RASearcher) this.search1, (RASearcher) this.search2);
67      }
68
69      // code below here is from Mikes AsbtractSearcher
70
71      @Override
72      public final int numberOfPositionsToTry() {
73          // TODO Auto-generated method stub
74          if(this.search1 == null && this.search2 == null) {
75              return this.endPos - this.startPos;
76          }
77          return this.search1.numberOfPositionsToTry() +
this.search2.numberOfPositionsToTry();
78      }
79
80      @Override
81      public final int numberOfPositionsTriedSoFar() {
82          // TODO Auto-generated method stub
83          if(this.search1 == null && this.search2 == null) {
84              return this.counter;
85          }
86          return this.search1.numberOfPositionsTriedSoFar() +
this.search2.numberOfPositionsTriedSoFar();
87      }
88
89      @Override
90      public void runSearch(SearchListener listener) throws
SearchCancelledException {
91          // TODO Auto-generated method stub
92          synchronized (listener) {
93              while (true) {
94                  final int foundMatch = this.findMatch(listener,
this.startTime);
95                  if (foundMatch >= 0) {
96                      listener.possibleMatch(foundMatch,
System.currentTimeMillis() - this.startTime,
97                              numberOfPositionsTriedSoFar());
98                  } else {
99                      break;
100                 }
101             }
102         }
103     }
104
105     @Override
106     public synchronized void reset() {
107
108     }
109
110     @Override
111     public synchronized void cancel() {
```

Figure 22

```
112     // TODO Auto-generated method stub
113     throw new SearchCancelledException();
114 }
115
116 private synchronized int findMatch(final SearchListener listener, final
long startTime) {
117     while (numberOfPositionsTriedSoFar() < numberOfPositionsToTry()) {
118         final boolean hit = tryPosition();
119         this.currentPos++;
120         this.counter++;
121         if (hit) {
122             return this.currentPos - 1;
123         }
124     }
125     return -1;
126 }
127
128 protected synchronized boolean tryPosition() {
129
130     final int x1 = this.currentPos % this.raster.getWidth();
131     final int y1 = this.currentPos / this.raster.getWidth();
132
133     double max = -Double.MAX_VALUE;
134     double min = Double.MAX_VALUE;
135     final double[] heights = new double[this.side * this.side];
136     int count = 0;
137     for (int x2 = 0; x2 < this.side; x2++) {
138         if (x1 + x2 >= this.raster.getWidth()) {
139             break;
140         }
141         for (int y2 = 0; y2 < this.side; y2++) {
142             if (y1 + y2 >= this.raster.getHeight()) {
143                 break;
144             }
145
146             long elevation = this.raster.getRGB(x1 + x2, y1 + y2);
147             elevation = elevation & 0xFFFFFFFFL; // mask off signed
upper 32 bits
148             double trueElevation = (long) ((elevation * 0.1) - 10000);
149             heights[count++] = trueElevation;
150
151             if (trueElevation >= max) {
152                 max = trueElevation;
153             }
154             if (trueElevation <= min) {
155                 min = trueElevation;
156             }
157         }
158     }
159
160     final double stdev = standardDevPop(heights, count);
161     return stdev < this.Threshold;
162 }
163
164 private synchronized double standardDevPop(final double[] array, final
int size) {
165     double sum = 0;
166     for (int i = 0; i < size; i++) {
167         sum += array[i];
168     }
169     final double mean = sum / size;
```

Figure 23

```
170      double variance = 0;
171      for (int i = 0; i < size; i++) {
172          final double dev = array[i] - mean;
173          variance += dev * dev;
174      }
175      variance /= size;
176      return Math.sqrt(variance);
177  }
178
179}
```

Figure 24

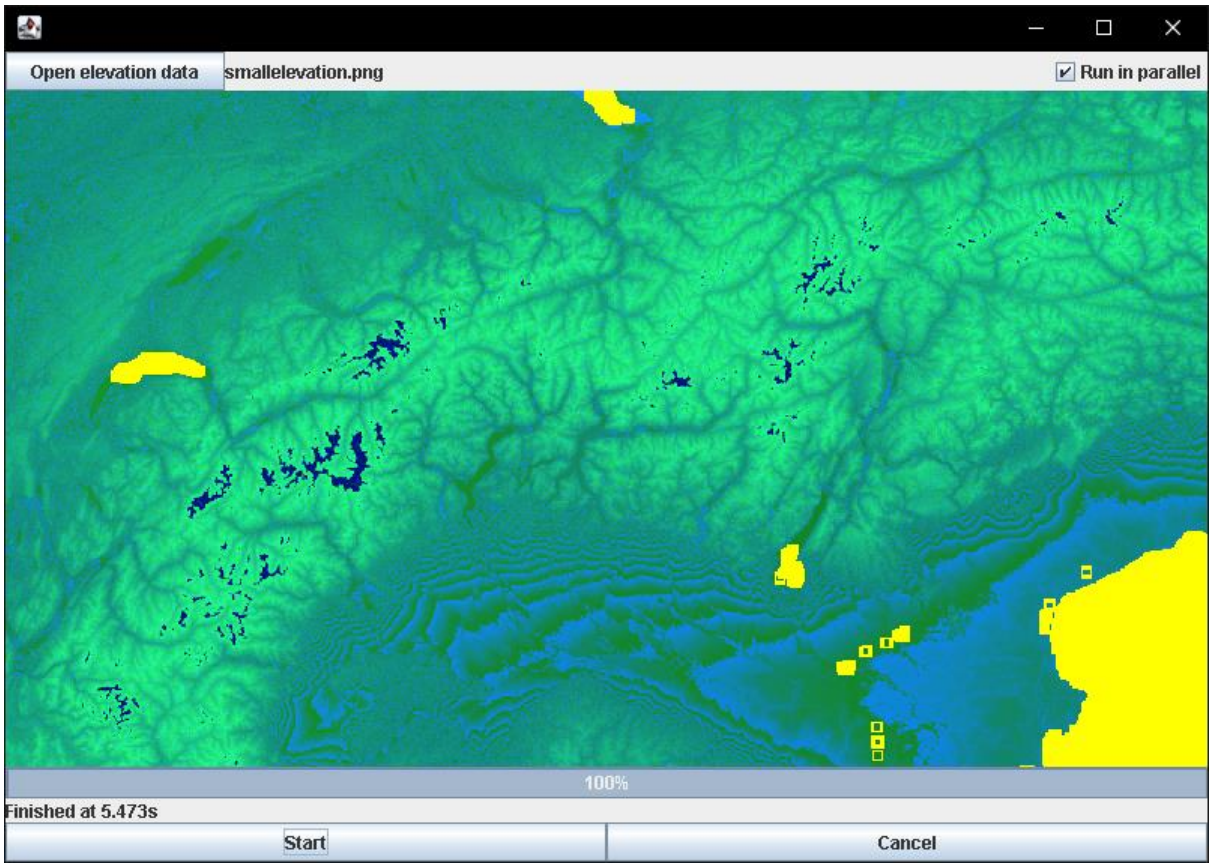

Figure 25

Objectives

Searcher object

The program executes from the *SearchUIEnhancement* class depicted from Figure 13 to Figure 18 where (JavaTpoint, 2011) was used to implement the checkbox that if *isParallel* is selected, as seen in Figure 25, line 113 is executed instead to execute the *runParallelSearch* method. Similar to the unparalleled search, the *running* variable stores the current thread on which the *runParallelSearch* method is executing on. The custom searcher is then initialised and passed the image, the current start position, the end position, an instance of *SearchUIEnhancement*, the time when the *runParallelSearch* method was called and 0 to initialise the counter as seen from the constructor of *RASearcher* in Figure 21. The *runParallelSearch* method then starts the progress bar on another thread like the implementation for the *runSearch* method.

(Child, 2020) was used to implement the *ForkJoinPool* in the *runParallelSearch* method however, as the *searcher* object reports that it's an instance of *Searcher*, the typecasting of *ForkJoinTask<?>* was

used to pass the *searcher* object to be invoked in the pool on line 168 in Figure 15. The *RASearcher* object extends *RecursiveAction* to subdivide tasks among multiple threads while implementing *Searcher* to carry out searches for points of elevation between a range of start and end positions thus making parallel searches execute faster than unparalleled searches.

Progress bar

Inside the *RASearcher* class, the *compute* method determines that if the *numberOfPositionsToTry* is less than the *Threshold*, the *runSearch* method is executed with a *SearchListener* instance depicted by the anonymous class in Figure 21. An anonymous class was used as it allows us to report possible matches to the *masterListener*, which is the instance of *SearchUIEnhancecment* passed to the *RASearcher* class on instantiation, while discarding any information or update messages as suggested in the specification. Otherwise, new *RASearcher* instances are created by splitting the number of positions to search through between them using a fork/join framework as shown on lines 61 to 66.

The code below line 69 has been adapted from the *AbstractSearcher* class for the logic of how a *searcher* object would determine points of elevation thus the *runSearch, findMatch, tryPosition* and *standardDevPop* methods have been significantly unaltered besides the omission to any calls to the *update* and *information* methods. The *numberOfPositionsToTry* and *numberOfPositionsTriedSoFar* methods have been altered so that when there are multiple instances of *RASearcher*, these methods return values associated with all of the instances of *RASearcher* so that the progress bar can update according to the correct *numberOfPositionsToTry* and *numberOfPositionsTriedSoFar*.

Cancel operation

Similar to the implementation in *DevelopedSearcher*, when the cancel button is clicked a *SearchCancelledException* is thrown to pass control from the thread running the search to the *AWTEventQueue* so that the thread running the search can be interrupted immediately. However, as the *runParallelSearch* method was executed the *pool* variable won't be null thus line126 in Figure 15 is executed so that all of the threads in the *ForkJoinPool* are terminated before the thread running the search is terminated to avoid any errors which could occur from a child thread running without their parent thread running. The *catch* block in the *runParallelSearch* method is used to catch a *CancellationException* from the *pool* variable being shut down prematurely, as described in (Programming, 2012). This allows us to report to the user interface that the search has been aborted. If a new *ForkJoinPool* wasn't instantiated each time the *runParallelSearch* method was called, a *RejectedExecutionException* would be thrown as after the *shutdown* or *shutdownNow* method has been executed on the *pool* variable, the *pool* variable would no longer be able to allocate a new job to the thread pool. If the cancel button isn't clicked, a parallel search will execute until completion of the search. To report to the interface that the search has completed, line 173 in Figure 15 updates the *outputLabel* to depict that the search has finished in a given amount of time.

Thread-safety

Similar to *SearchUIEnhancement*, most methods in *RASearcher* have implemented the *synchronized* keyword so that only 1 thread may execute that method at any given time. The *synchronized* keyword has also been used in the *runSearch* method on the *listener* object to block other threads from triggering the *findMatch* and *possibleMatch* methods. To make the timer for *RASearcher* concurrent between each thread, it is initialized outside of the *RASearcher* class and is given the *final* keyword when stored inside the *startTime* variable inside the *RASearcher* class so that the timer doesn't reset when interacting with new *RASearcher* instances. Most variables in *RASearcher* have been initialised with a *final* keyword so that the value given to each of these variables isn't changed when multiple threads are manipulating them however, the *search1, search2, currentPos* and *counter* variables are specified with a *volatile* keyword as they may change when manipulated on different threads. We could have adapted a *counter* from (Oracle, 1995) to implement a concurrent counter which would report the correct *numberOfPositionsTriedSoFar*, however after attempting to implement this we

experienced a circular lock issue caused by multiple threads trying to update the *counter* variable at the same time while other threads were trying to access a different method thus, due to time constraints we are unable to identify where the circular lock has originated from. The current implementation satisfies most of the objectives identified within the specification however, we haven't been able to mend the incorrect reporting of the *numberOfPositinosTriedSoFar* on each thread as each thread is updating its instance of the *counter* variable to its respective *SearchListener* instance.

References

Bhatti, K., 2021. *Elevation.* [Online]
Available at: https://github.com/k5924/Elevation
[Accessed 19 April 2021].

Child, M., 2020. *Concurrency.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1543265
[Accessed 17 April 2021].

Child, M., 2020. *Parallel Programming - Fork/Join.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1543279
[Accessed 23 April 2021].

Child, M., 2021. *Approaches to thread safety.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1543272
[Accessed 17 April 2021].

Child, M., 2021. *Single-threaded architectures.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1543276
[Accessed 17 April 2021].

JavaTpoint, 2011. *Java JCheckBox.* [Online]
Available at: https://www.javatpoint.com/java-jcheckbox
[Accessed 23 April 2021].

Oracle, 1993. *Class JProgressBar.* [Online]
Available at: https://docs.oracle.com/javase/8/docs/api/javax/swing/JProgressBar.html
[Accessed 17 April 2021].

Oracle, 1995. *How to Use GridLayout.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html
[Accessed 17 April 2021].

Oracle, 1995. *Lambda Expressions.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html#lambda-expressions-in-gui-applications
[Accessed 17 April 2021].

Oracle, 1995. *Synchronized Methods.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html
[Accessed 23 April 2021].

Programming, C. o., 2012. *Advanced Java: Multi-threading Part 14 - Interrupting Threads.* [Online]
Available at: https://www.youtube.com/watch?v=6HydEu75iQI&t=630s
[Accessed 23 April 2021].