HIGHER DIPLOMA IN COMPUTER SCIENCE -

CS211 - Algorithms and Data Structures - end of semester assignment

By Charles Byrne

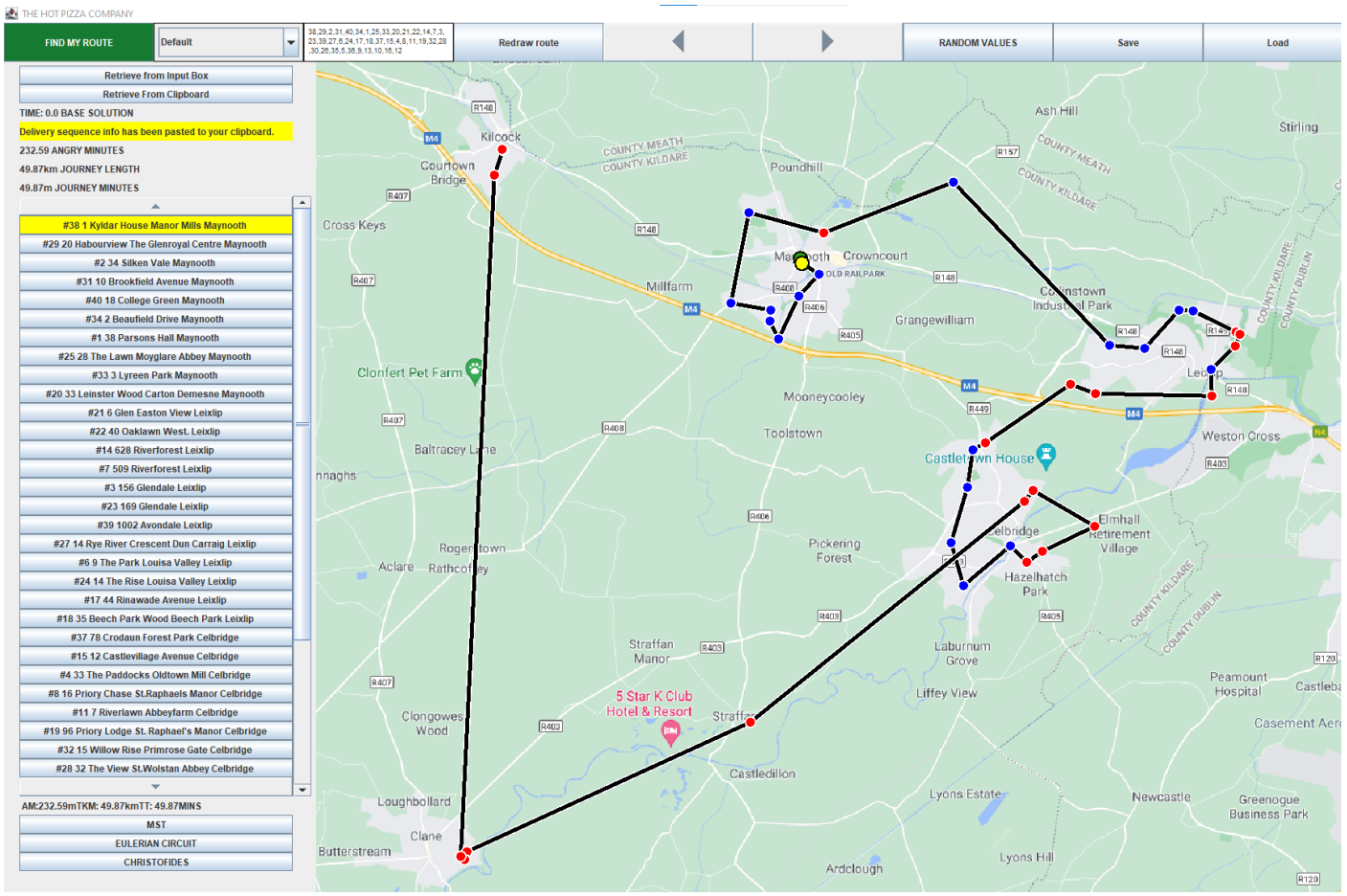
29th May 2021

97700266

1. Introduction

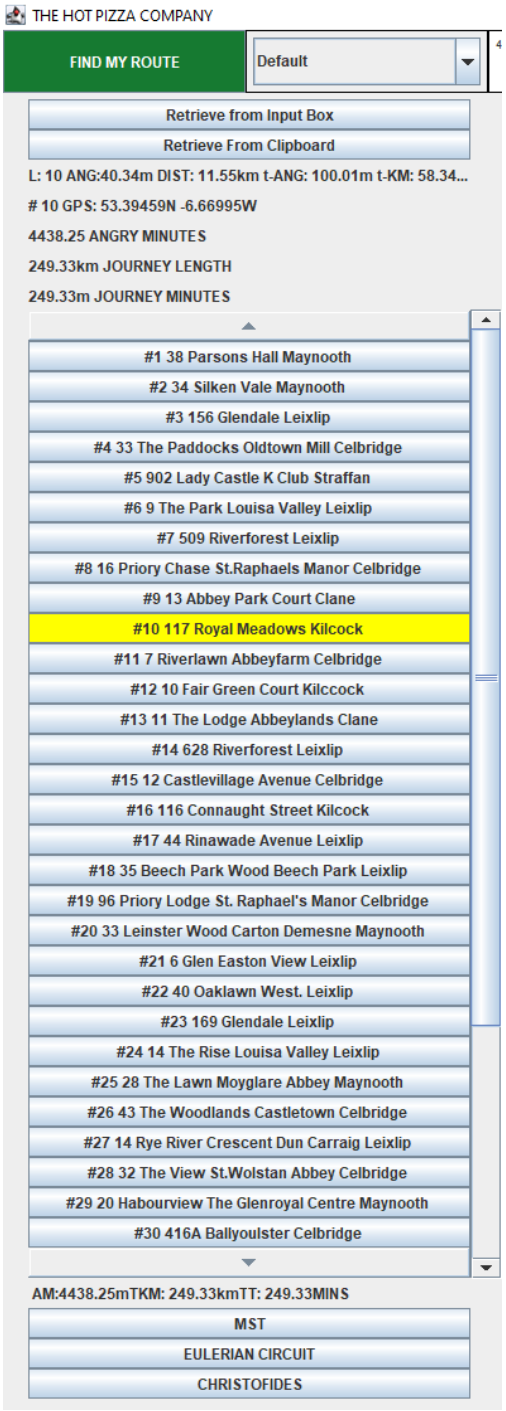
It might seem like a statement of the obvious but problem solving has two aspects, the problem itself and the amount of time available to solve it. One of the key virtues required for problem-solvers, be they in the IT or mathematical world, is tenacity. One must persevere until the difficulty is overcome. The counterbalance to this is prudence, we simply do not have an infinite amount of time to contemplate every complexity and sometimes it is simply not possible to arrive at a timely solution. In this case we must find a workable compromise.

Given that this was the very premise of this project it is ironic that I have had to find this balance in more ways than one, given time constraints during this frantic examination season.



Overview and main features

I had originally planned on using the Held–Karp algorithm, but after quite a lot of research I realised that this was going to be impossible, given the upper bound of 100 vertices and also the constraint of ten seconds computer runtime. Anything up to about 40 could have been doable, but after that the numbers become astronomical! I decided instead to try to implement the Christofides Algorithm, and after some effort I managed to implement it with just one minor adjustment to factor in time constraints and limited research time. My delight and relief at finishing this was tampered somewhat but the realisation that a much cruder solution was more effective, given the area and the number of points (up to 100).



The only part of the Christofides I trimmed was the search for the minimum perfect matching of the odd degrees. I used a Monte Carlo simulation which seemed to give quite a stable result. I tried other methods but abandoned them as the simple one was quicker every time. One method I thought would work was a kind of nearest-neighbour with randomness. Again it didn't seem to make a positive impact on delivery times or angry minutes.

It is possible to choose the Christofides algorithm and to compare the results of various methods.

Factoring in the minutes that customers were already waiting was difficult. Chasing after one angry customer can easy disgruntle ten others. The ripple effect makes such a strategy ineffective. I found that the best results came when I implemented the following, which seems very simple after Christofides:

- Use the nearest neighbour algorithm
- Then randomly swap vertices, keeping the swaps if angry minutes go down.

I tried other more complex switching algorithms but none of them seemed to have the desired effect.

Graphics

I put considerable effort into creating a usable graphical interface. Time constraints meant that I had to abandon polishing it as much as I would have liked. Features such as the zoom need adjustment.

A note on screen size and the map file

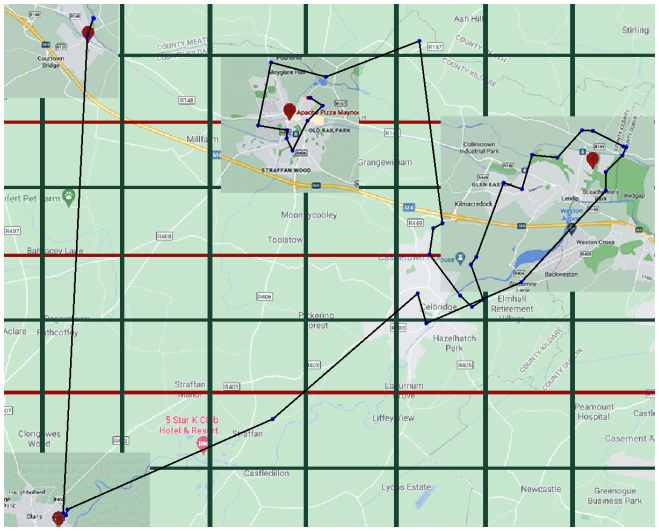
Please view this application in 1920 x 1200. I had hoped to implement screen resizing but time did not permit. The map.png file will be loaded immediately on startup. If the map does not load for some reason, please see line 2621, near the end.

What can you do?

- Data will automatically be loaded in from the clipboard. This was tested in Windows 10
- Results will automatically be returned to your clipboard.
- You can also paste data into the text area and copy the results from this after the travelling salesman algorithm has been executed.
- You can move through the delivery route by pressing the up or down buttons. The currently selected vertex is highlighted yellow.
- You can place a new vertex on the map by pressing “CTRL” and clicking on the map.
- You can select a vertex on the map and it will become highlighted in the list on the left.
- You can move a vertex on the map by selecting it and dragging.
- You can move a vertex, in terms of order, by pressing the “CTRL” button and the up or down keys respectively.
- You can delete a vertex by selecting it and pressing the “DELETE” button
- You can zoom in
    - by pressing “CTRL” and moving the mouse wheel
    - or by pressing the + or - keys
    - Due to map file constraints you cannot zoom out further, but this could be easily added if more map files were loaded.
        - Zoom is not complete: Dragging the map leaves some discrepancy between the graph and the map. Also, you cannot move a point when zoomed but you can insert one. (A tiny bit of mathematics I didn’t get around to but it shouldn’t be too difficult to implement.) Here I had to exercise prudence.
- You can create a random arrangement of vertices on the map.
- You can load and save vertices. The file format is a rudimentary .CSV file, without string encoding, so you cannot have “,” commas in the address line. Again, this is a simple thing to add at a later date.
- Customers who are going to be angry are marked red

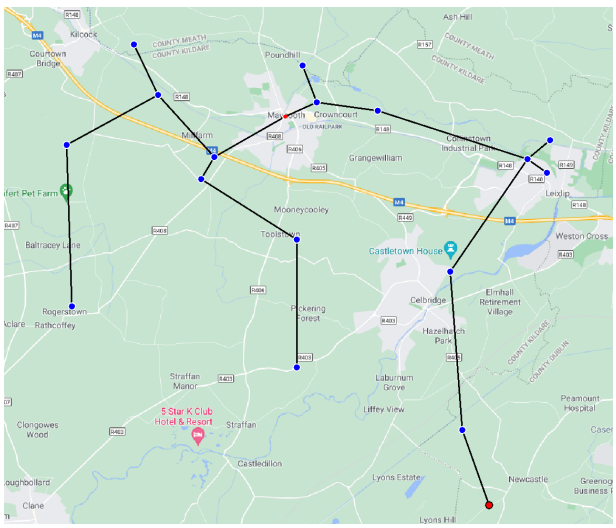
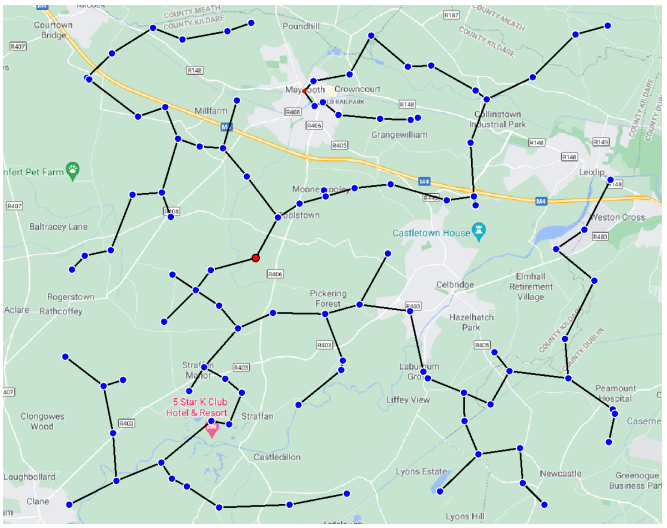
2. Creation:

There were many things to do in this creative endeavour. How do we, for instance, line up the GPS coordinates with the map points? For this, I superimposed four key points on the map, in order to pinpoint all the GPS locations in relation to the map. The following is a screenshot when I wasn’t quite in line, but almost:



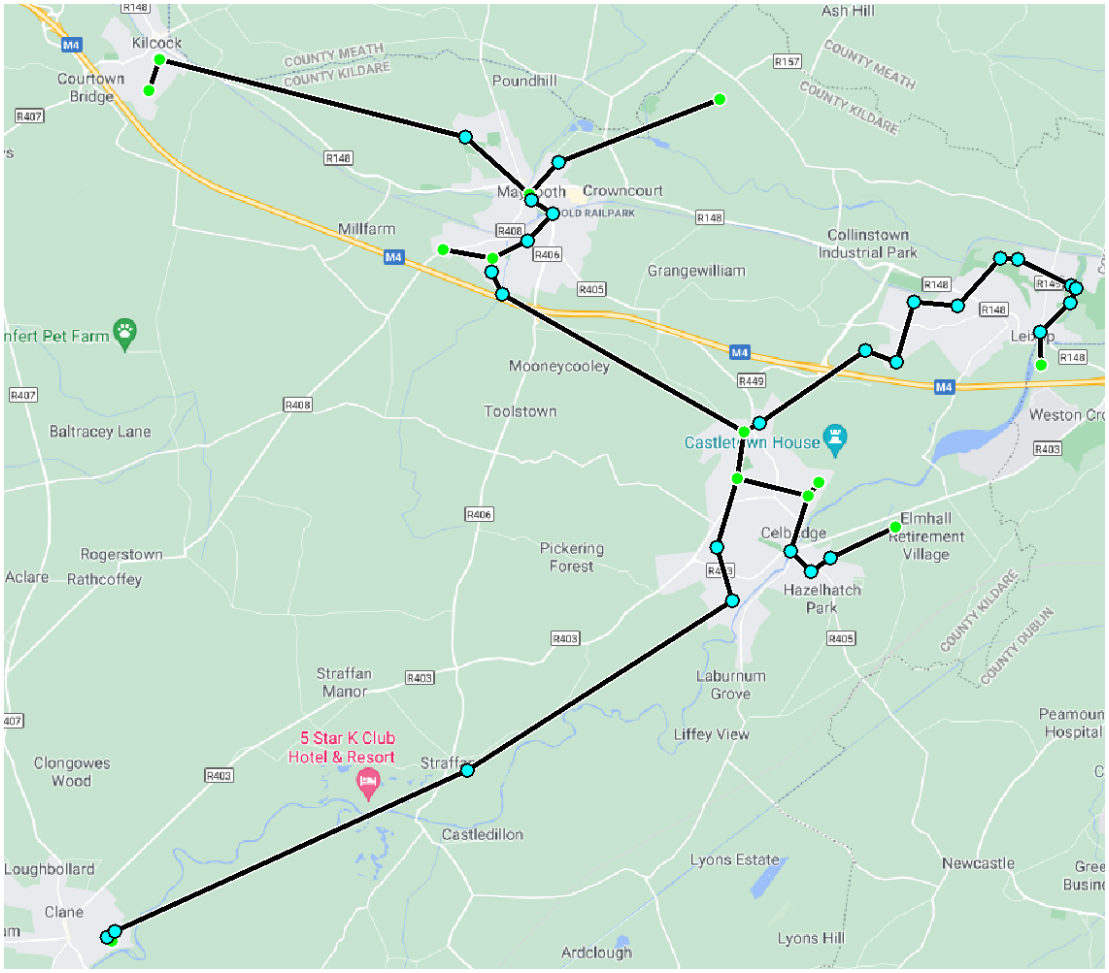
1. Algorithms used

Under the address list (if your monitor is big enough), I have left in some of the stages of the Christofides algorithm. The first is to create a minimum-spanning-tree. I have to say, I was very happy to see these graphics on my screen:

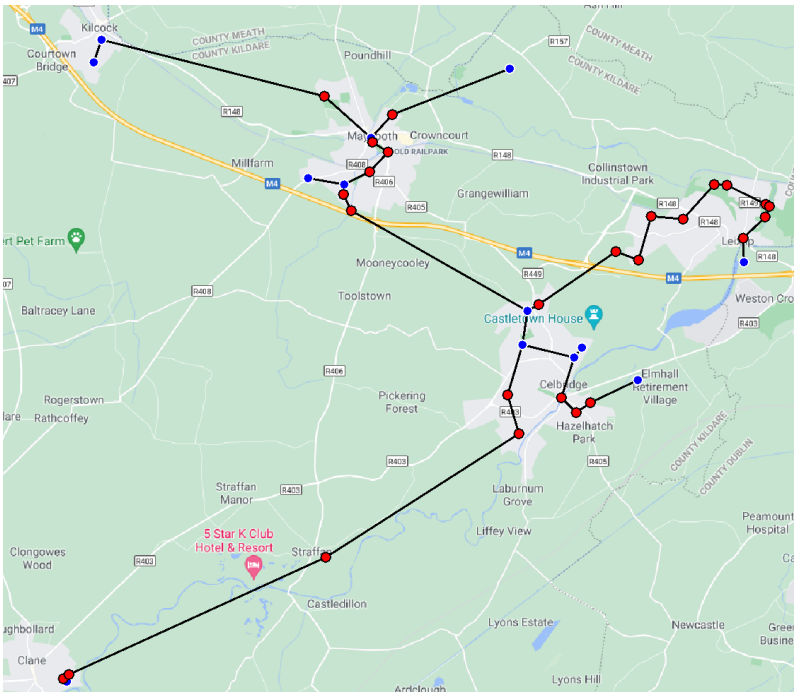
A simple MST - 17 addresses+Start = 17 nodes **vs** 100 addresses+Start = 100 nodes

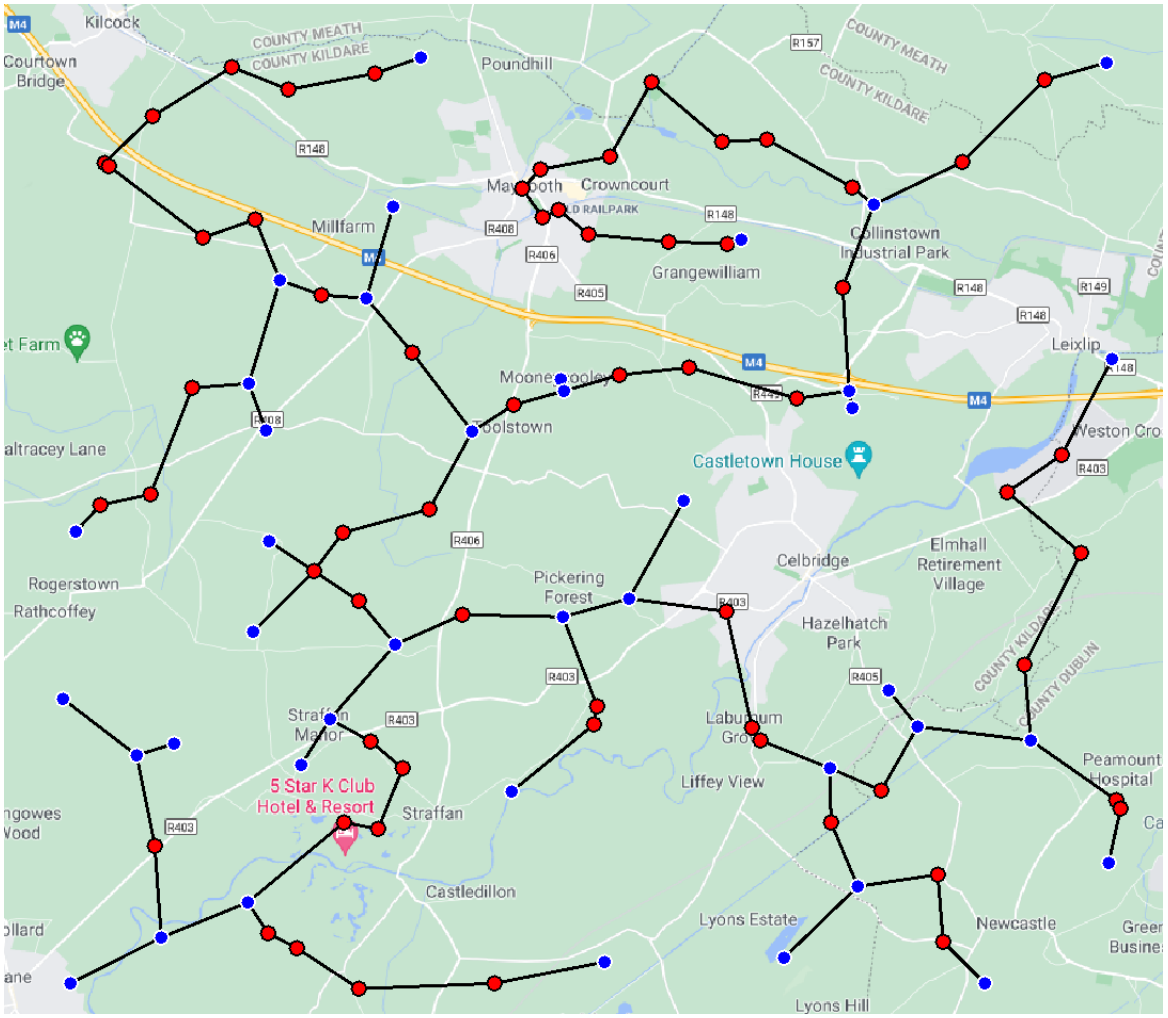
The next stage is to find the vertices with an odd number of edges coming out of the minimum spanning tree. Here they are coloured light green:



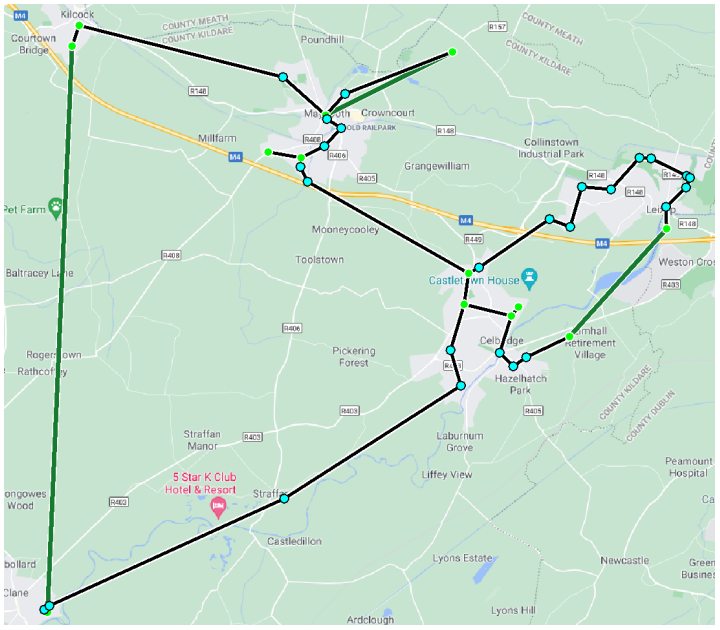
The idea is to make all these even, so that we can close off the circuit and complete the tour. We will find that this is always an even number. We pair them up by finding the cheapest combination of pairs. A hundred vertices averaged out at about 40 vertices. To brute-force this I worked out it would take about 40 factorial / (2^{20}). That's not going to happen in ten seconds!

There are other algorithms but I simplified this stage for the sake of simplicity and workable runtime. This is definitely a point (or vertex!) I wish to revisit at a later date.



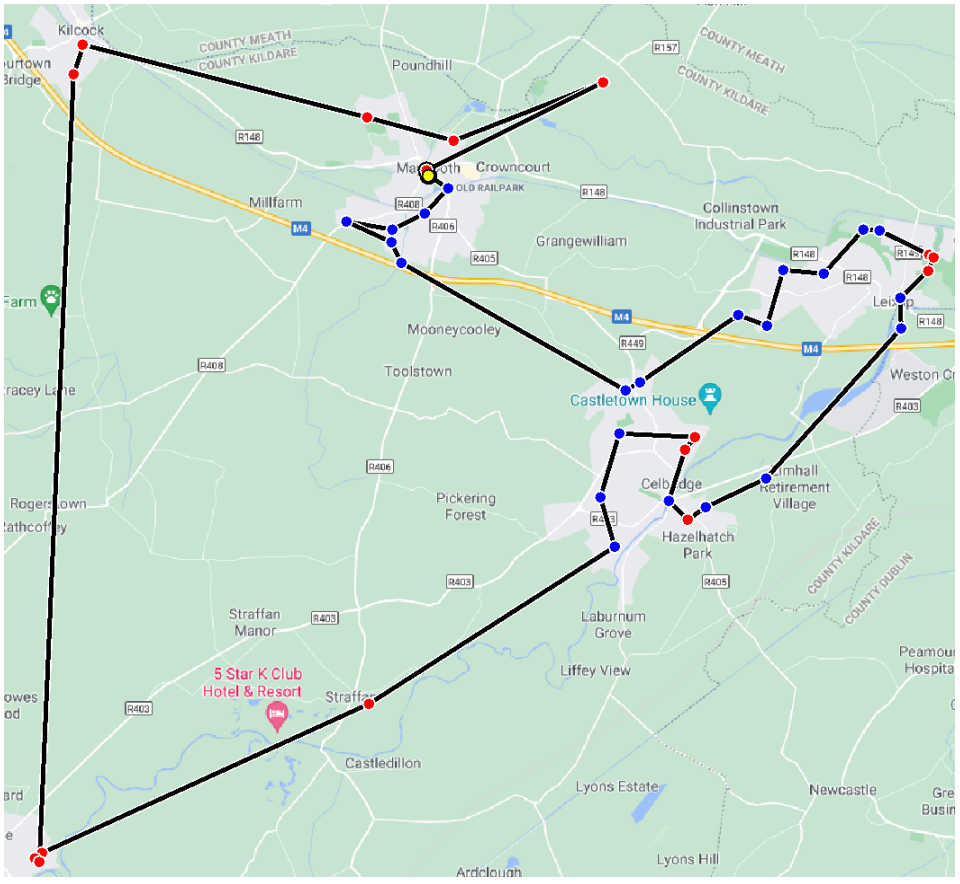


The following is an example of an Eulerian Multigraph:
The minimum-cost edges are combined with the Minimum Spanning Tree:

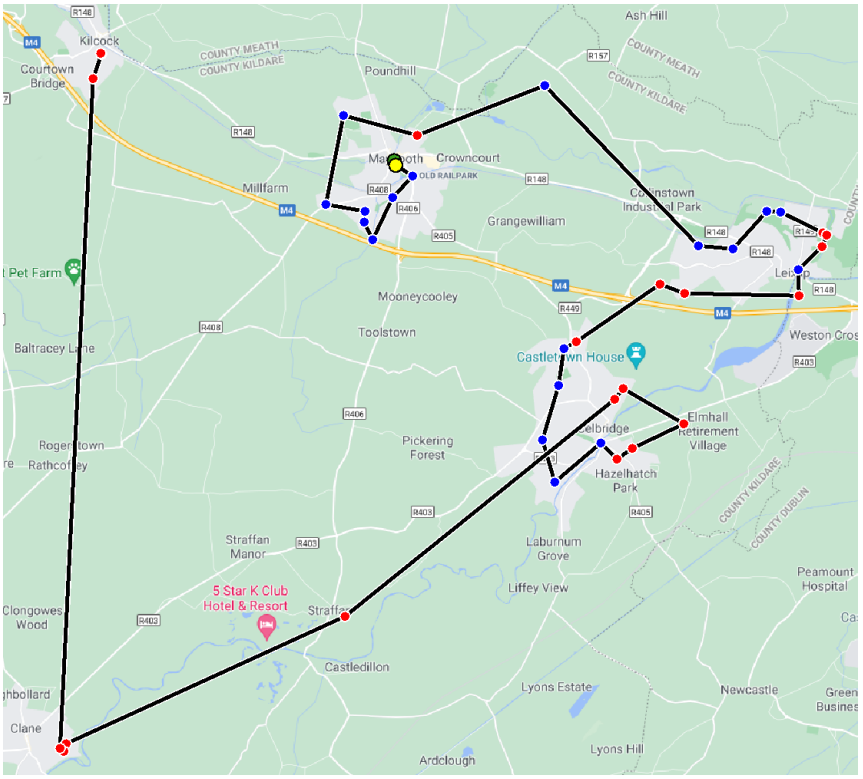


From this point, the algorithm is almost complete. We create a Eulerian Circuit by recursively finding the path that travels along each edge once and visits all the cities. After this we create a Hamiltonian Path by removing any return visits to vertices already visited.
1 → 4 → 3 → 4 → 5 would thus become 1 → 4 → 3 → 5. We now have our solution!

This looks nice, but was not as short as the cruder algorithm:



I thought that I would probably have to introduce some other randomness to account for already-waiting times, but this didn't seem to make a major difference. I also wondered how much the outcome would be affected by the fact that we don't have to make a complete circuit back to the pizza place.



I wasn't really disappointed with the result, more fascinated; and glad that I tried the Christofides algorithm. The wisdom I take from this is that we should try all *possible* paths in such endeavours, sometimes the simpler solution is actually the one for that particular case.

2. Data structures and code structure

I kept the data structures simple in this problem, everything uses arrays of some sort, since arrays are quite fast and their elements are very easily accessible. I think that I could make some elements more efficient but I saw no issues with lag on any occasion. There was a lot of manipulation of data but the data sizes weren't huge at all by modern standards.

The overall structure of the code is very functional, and simple. Of course it could be a bit more organised, but I think that I got a lot done in a very short space of time and never was I hindered in any way by chaotic code.

3. Learning outcomes

I had some experience writing Graphics in Java in the previous end-of-semester assignment but I was not at all fluent and found this exercise incredibly instructive. Everything was a struggle at first but then I gradually got more used to the way things work.

This project taught me a lot about time management, about keeping focused under pressure and about prioritisation and prudence. For the algorithm, I did as much research as I possibly could. To borrow an apt term, I would probably say that I brute-forced my way through the workload as much as I possibly could, and although Christofides didn't end up being the fastest solution to the problem I really was very happy that I followed that path.

APPENDIX:

Source Code:

```
// --------------------------------------------------------------------------
//
//                                          Charles Byrne
//                                          CS211
//                                          FINAL PROJECT
//                                          -----------------
//                                          29th May 2021
//
//                                          Tested on 1920 x 1200 screen
//                                          Windows 10 - Java in Eclipse
//
//
// --------------------------------------------------------------------------

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SpringLayout;
import javax.swing.Timer;
import javax.swing.border.Border;
import javax.swing.border.EmptyBorder;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.plaf.basic.BasicArrowButton;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GraphicsEnvironment;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.HeadlessException;
import java.awt.Image;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.StringSelection;
import java.awt.datatransfer.UnsupportedFlavorException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;
import java.awt.event.InputEvent;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
```

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.FileSystemNotFoundException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

class Address {
	int number;
	String postalAddress;
	double minutesWaiting, angriness;
	double gpsN, gpsW;
	int x, y;

	Address(String dataAsString) {
		String[] parts = dataAsString.split(",");
		if (parts.length != 5) {
			System.out
			.println("INVALID DATA IN. 5 Parts required ; only " + parts.length + " parts in " +
dataAsString);
			this.number = 0;
			this.postalAddress = "";
			this.minutesWaiting = 0;
			this.gpsN = 0;
			this.gpsW = 0;
		} else {
			this.number = Integer.parseInt(parts[0]);
			this.postalAddress = parts[1];
			this.minutesWaiting = Integer.parseInt(parts[2]);
			this.angriness = this.minutesWaiting - 30;

			this.gpsN = Double.parseDouble(parts[3]);
			this.gpsW = Double.parseDouble(parts[4]);
		}
	}

	Address(int number, String postalAddress, int minutesWaiting, double gpsN, double gpsW) {
		this.number = number;
		this.postalAddress = postalAddress;
		this.minutesWaiting = minutesWaiting;
		this.angriness = minutesWaiting - 30;
		this.gpsN = gpsN;
		this.gpsW = gpsW;
	}
}

class Trip {
	int from, to;
	double metres;

	Trip(int from, int to, double metres) {
		this.from = from;
		this.to = to;
		this.metres = metres;
	}
}

@SuppressWarnings("unchecked")
class FastDeliver_window extends JFrame {

	public static boolean[][] grid;
	public static boolean[][] gridBackup;
```

```
	static JButton[][] gridButtons;
	static JTextArea inputBox;
	private JTextArea pasteInTextField;
	static JButton drawOutRouteButton, buttonExit, buttonClear, buttonOnce, buttonRandom, buttonReset;
	static JButton buttonLoad, buttonSave, retrieveDataButton, findMyRoute, testButton, testButton2,
testButton3, testButton4,
	insertDataButton;
	static BasicArrowButton lastPointButton, nextPointButton, upScrollButton, downScrollButton;
	static JComboBox<String> algorithmChooser;
	static JPanel mapFrame;
	static JLabel spacerLabel, noticeboard2, noticeboard3;
	static BufferedImage image;
	int[][] minimumSpanningTree;
	int[] vDegreeCount;
	int mouseDown_x = 0;
	int mouseDown_y = 0;
	int mapX = 0;
	int mapY = 0;
	boolean deliveriesSorted = false;
	JLabel noticeboard, statsBoard1, statsBoard2, statsBoard3;
	JPanel canvas;
	Rectangle screenDimensions;
	BufferedImage img;
	JButton[] sideList;

	JFileChooser fileChooser;
	int sideListScroll = 0;
	JScrollBar sideScrollBar;
	KeyListen listener;
	int currentSelectedPoint = 0;
	JLayeredPane primaryPanel;
	int randSwapCount = 0;
	Random getRandom = new Random();

	public static int global_difX;
	public static int global_difY;
	public Address[] addresses = new Address[105];
	public int addressCount;
	public Trip[][] distanceMatrix, distanceMatrix_unsorted;
	int[] deliveries = new int[105];
	boolean[] isCustomerAngry = new boolean[105];

	public double currentAngriness;
	public Graphics2D g;
	public int global_i = 0;
	public int lastChanged = 0;
	public double map_x1 = -6.710951;
	public double map_x2 = -6.455690;
	public double map_y1 = 53.410032;
	public double map_y2 = 53.283897;
	public double mapZoom = 1;
	public double mapFit = 1.5;
	int gridXSize = 1600;
	int gridYSize = 1200;
	double gpsScaleX = 0.315;
	double gpsScaleY = 0.142;
	int currentClickedPoint = -1;
	String algorithmList[] = { "Default", "Method 2", "Christofides","Christofides + RAND" };
	int algorithmMethod = 0;
	long algorithmStartTime;
	double deliverySpeedMPS = 60000.0 / 3600.0; // metres per second
	ArrayList<Integer>[] edge; // TODO: replace other simple arrays
	int[][] tour;
	int currentTourIndex;
	int vertexCount = 0;
	int[][] eulerianMultigraph;

	public int sideListLastClicked = -1;
	public Address pizzaPlace;

	class KeyListen extends KeyAdapter {
		@Override
		public void keyPressed(KeyEvent e) {
			int keyPressed = e.getKeyCode();
```

```
                //        System.out.println("KEY" + );
                boolean isContolPressed = e.isControlDown();
                System.out.println(isContolPressed);
                if (keyPressed == KeyEvent.VK_UP) {
                        System.out.println("UP PRESSED");
                        if (isContolPressed) {
                                System.out.println("CTRL+UP ");
                                goSwap(-1);
                        }
                        goUpDown(-1);
                } else if (keyPressed == KeyEvent.VK_DOWN) {
                        System.out.println("DOWN PRESSED");
                        if (isContolPressed) {
                                System.out.println("CTRL+DOWN ");

                                goSwap(1);

                        }
                        goUpDown(1);

                } else if (keyPressed == KeyEvent.VK_DELETE) { // DELETE POINT ON MAP

                        if (currentSelectedPoint > 0 && currentSelectedPoint < addressCount) {
                                goDeletePoint();
                        }
                } else if (keyPressed == 45 || keyPressed == 109) {
                        // && ((e.getModifiersEx() & KeyEvent.CTRL_DOWN_MASK) != 0)
                        System.out.println("----");

                        zoom(-1);
                } else if (keyPressed == 61 || keyPressed == 107
                                // && ((e.getModifiersEx() & KeyEvent.CTRL_DOWN_MASK) != 0)
                                ) {
                        zoom(1);
                }
        }
}

void goDeletePoint() {
        System.out.println("DELETE DELIVERY #: " + currentSelectedPoint);
        System.out.println("DELETE POINT #: " + deliveries[currentSelectedPoint]);

        int pointToDelete = deliveries[currentSelectedPoint];
        for (int i = pointToDelete; i < addressCount - 1; i++) {
                addresses[i] = addresses[i + 1];
        }
        int n = 0;
        int thisDelivery;
        for (int i = 0; i < addressCount; i++) {
                if (deliveries[i] != pointToDelete) {
                        if (deliveries[i] > pointToDelete) {
                                thisDelivery = deliveries[i] - 1;
                        } else {
                                thisDelivery = deliveries[i];
                        }
                        deliveries[n] = thisDelivery;
                        n++;
                }
        }
        addressCount--;
        setMatrixEtc();

        if (currentSelectedPoint >= addressCount)
                currentSelectedPoint--;
        if (currentSelectedPoint < 1)
                currentSelectedPoint = -1;
        if (addressCount < 31)
                sideListScroll = 0;
        if (addressCount - sideListScroll < 30)
                sideListScroll--;
        if (currentSelectedPoint > addressCount) {
                currentSelectedPoint--;
                setSelectedPoint(currentSelectedPoint);
        }
```

```
            sideScrollBar.setValues(sideListScroll, (sideListScroll + 30), 0, addressCount - 1);
            printAddressList();
            refreshMap(addressCount, deliveriesSorted);
            showTotalStats();
        }

        void goSwap(int upOrDown) {
            if (currentSelectedPoint > 0) {
                int swapWith = currentSelectedPoint + upOrDown;
                if (swapWith > 0 && swapWith < addressCount) {
                    swap(currentSelectedPoint, swapWith);
                    printAddressList();
                }
            }
        }

        void goUpDown(int upOrDown) {
            System.out.print("currentSelectedPoint: " + currentSelectedPoint);
            System.out.print("sideListScroll: " + sideListScroll);
            int newValue = currentSelectedPoint + upOrDown;
            if (newValue < 1)
                newValue = 1;
            else if (newValue >= addressCount)
                newValue = addressCount - 1;

            if (newValue > 0 && newValue < addressCount) {
                currentSelectedPoint = newValue;
                if (currentSelectedPoint - sideListScroll < 1) {
                    sideListScroll--;
                    printAddressList();
                } else if ((currentSelectedPoint - sideListScroll) > 30) {
                    sideListScroll++;
                    printAddressList();
                }

                sideScrollBar.setValue(sideListScroll);
                setSideListButton(currentSelectedPoint);
                setSelectedPoint(currentSelectedPoint);
            }

        }

        void drawOutRoute() {

            if (addressCount < 1) {
                noticeboard.setText("NO ADRESSES YET. ctrl+click to add.");
            } else {
                showTotalStats();
                global_i = 0;
                timer.start();
            }

        }

        Timer timer = new Timer(100, new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                // for (int i = 0; i<10; i++)
                // switchTwo();
                if (global_i < (addressCount)) {
                    global_i++;
                    //             System.out.println("Delivery " + global_i + " House # " +
deliveries[global_i]);
                    refreshMap(global_i, true);
                } else {
                    timer.stop();

                }

            }
        });

        int[] pointByZoom(int x, int y) {
```

```
            double mapX_ = ((double) mapX) * 0.75;
            double mapY_ = ((double) mapY) * 0.75;
            int x2 = (int) ((double) ((x + mapX_) * mapZoom));
            int y2 = (int) ((double) ((y + mapY_) * mapZoom));
            int[] returnThis = { x2, y2 };

            return returnThis;
        }

        public void refreshMap(int numberOfPoints, boolean showRoute) {

            int x1, y1, x2, y2;
            double mapX_ = ((double) mapX) * 0.75;
            double mapY_ = ((double) mapY) * 0.75;
            BasicStroke basicStroke = new BasicStroke(5);
            BasicStroke thickerStroke = new BasicStroke(8);

            canvas.repaint();
            int xD = (int) (((double) screenDimensions.width) * (mapZoom * 0.82) * mapFit);
            int yD = (int) (((double) screenDimensions.height) * (mapZoom * 0.82) * mapFit);
            System.out.println(xD + " x " + yD);
            System.out.println("mapX; mapY: " + mapX + " ; " + mapY);
            g.drawImage(img, mapX, mapY, xD, yD, 0, 0, screenDimensions.width - mapX,
screenDimensions.height - mapY, null);

            g.setStroke(basicStroke);
            x2 = (int) ((double) ((addresses[deliveries[0]].x + mapX_) * mapZoom));
            y2 = (int) ((double) ((addresses[deliveries[0]].y + mapY_) * mapZoom));

            if (showRoute) {
                for (int i = 1; i < numberOfPoints; i++) {
                    x1 = x2;
                    y1 = y2;
                    x2 = (int) ((double) ((addresses[deliveries[i]].x + mapX_) * mapZoom));
                    y2 = (int) ((double) ((addresses[deliveries[i]].y + mapY_) * mapZoom));

                    g.setColor(Color.white);
                    g.setStroke(thickerStroke);
                    g.drawLine(x1, y1, x2, y2);
                    g.setColor(Color.black);
                    g.setStroke(basicStroke);
                    g.drawLine(x1, y1, x2, y2);
                }
            }

            x2 = (int) ((double) ((addresses[deliveries[0]].x + mapX_) * mapZoom));
            y2 = (int) ((double) ((addresses[deliveries[0]].y + mapY_) * mapZoom));
            g.setColor(Color.black);
            g.fillOval(x2 - 10, y2 - 10, 20, 20);
            g.setColor(new Color(0x379C1E));
            g.fillOval(x2 - 8, y2 - 8, 16, 16);

            for (int i = 1; i < numberOfPoints; i++) {
                x2 = (int) ((double) ((addresses[deliveries[i]].x + mapX_) * mapZoom));
                y2 = (int) ((double) ((addresses[deliveries[i]].y + mapY_) * mapZoom));

                if (currentSelectedPoint != i) {
                    g.setColor(Color.white);
                    g.fillOval(x2 - 8, y2 - 8, 16, 16);
                    g.setColor(Color.blue);
                    if (isCustomerAngry[i]) {
                        g.setColor(Color.red);
                    } else {
                        g.setColor(Color.blue);
                    }
                    g.fillOval(x2 - 6, y2 - 6, 12, 12);

                } else {
                    if (isCustomerAngry[i]) {
                        g.setColor(Color.red);
                    } else {
                        g.setColor(Color.black);
                    }
```

```
                        g.fillOval(x2 - 10, y2 - 10, 20, 20);
                        g.setColor(Color.yellow);
                        g.fillOval(x2 - 8, y2 - 8, 16, 16);
                }

        }

        this.requestFocusInWindow();

    }

    public static double haversineFormula(double lat1, double lon1, double lat2, double lon2) {

        int earthsRadiusMeters = 6371;
        double lat1Radians = lat1 * Math.PI / 180;
        double lat2Radians = lat2 * Math.PI / 180;
        double diffLatRadians = (lat2 - lat1) * Math.PI / 180;
        double diffLonRadians = (lon2 - lon1) * Math.PI / 180;

        double part1 = Math.pow(Math.sin(diffLatRadians / 2), 2)
                + Math.cos(lat1Radians) * Math.cos(lat2Radians) *
Math.pow(Math.sin(diffLonRadians / 2), 2);
        double distance = 2 * Math.atan2(Math.sqrt(part1), Math.sqrt(1 - part1));

        return (distance * earthsRadiusMeters) * 1000;
    }

    // ------------------------------------------------------------------------------------------

    public void setMatrixEtc() {

        double gpsN_lowest = map_y2;
        double gpsW_lowest = map_x1;

        int lastX = 0;
        int lastY = 0;
        Trip lowest;
        gridXSize = 1600;
        gridYSize = 1200;
        gpsScaleX = 0.315;
        gpsScaleY = 0.142;

        for (int i = 0; i < addressCount; i++) {
            int thisX = 20 + (int) (((addresses[i].gpsW - gpsW_lowest) / gpsScaleX) * gridXSize);
            int thisY = 1080 - (int) (((addresses[i].gpsN - gpsN_lowest) / gpsScaleY) *
gridYSize);
            addresses[i].x = thisX;
            addresses[i].y = thisY;

            // .fillRect(x, y, width, height);
        }

        double distance_;

        for (int i3 = 0; i3 < addressCount; i3++) {

            for (int i = 0; i < addressCount; i++) {

                // double distance=
                // haversineFormula(addresses[house1].gpsN,addresses[house1].gpsW,
                // addresses[house2].gpsN,addresses[house2].gpsW);

                distance_ = haversineFormula(addresses[i3].gpsN, addresses[i3].gpsW,
addresses[i].gpsN,
                        addresses[i].gpsW);

                distanceMatrix[i3][i] = new Trip(i3, i, distance_);
                distanceMatrix_unsorted[i3][i] = new Trip(i3, i, distance_);

            }
            lowest = new Trip(0, 0, 0);
            int lowestI;
            for (int ii = 0; ii < addressCount - 1; ii++) {
                lowest.from = distanceMatrix[i3][ii].from;
```

```
                        lowest.to = distanceMatrix[i3][ii].to;
                        lowest.metres = distanceMatrix[i3][ii].metres;
                        lowestI = ii;
                        for (int i = ii + 1; i < addressCount; i++) {
                                if (distanceMatrix[i3][i].metres < lowest.metres) {
                                        lowest.from = distanceMatrix[i3][i].from;
                                        lowest.to = distanceMatrix[i3][i].to;
                                        lowest.metres = distanceMatrix[i3][i].metres;
                                        lowestI = i;
                                }
                        } // inner for loop
                        Trip temp = new Trip(distanceMatrix[i3][ii].from, distanceMatrix[i3][ii].to,
                                        distanceMatrix[i3][ii].metres);

                        distanceMatrix[i3][ii].from = lowest.from;
                        distanceMatrix[i3][ii].to = lowest.to;
                        distanceMatrix[i3][ii].metres = lowest.metres;

                        distanceMatrix[i3][lowestI].from = temp.from;
                        distanceMatrix[i3][lowestI].to = temp.to;
                        distanceMatrix[i3][lowestI].metres = temp.metres;
                } // outer for loop

        }
    }

    double checkStopWatch(String display) {
        double currentTime = ((double) ((System.nanoTime() - algorithmStartTime) / 10000000) / 100);
        if (display.length() > 0) {
                noticeboard2.setText("TIME: " + currentTime + " " + display);
                noticeboard2.paintImmediately(noticeboard2.getVisibleRect());
        }
        return currentTime;
    }

    int checkStopWatch_QUICK() {
        return (int) ((System.nanoTime() - algorithmStartTime) / 10000000);
    }

    private final ActionListener goFindMyRoute = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                // JComponent source = (JComponent)e.getSource();

                algorithmStartTime = System.nanoTime();

                if (algorithmMethod == 0) { // DEFAULT ALGORITHM
                        algorithm1();
                } else if (algorithmMethod == 1) {
                        algorithm2();
                } else if (algorithmMethod == 2) {
                        goTest(3);
                } else if (algorithmMethod == 3) {
                        goTest(3, true); // add randomness
                } else {
                        algorithm1();
                }

                // double duration =
                checkStopWatch("");
                sendToNoticeboard("Delivery sequence info has been pasted to your clipboard.", true);
                System.out.println("addressCount " + addressCount);
                double[] distanceStats2 = timeToThisCustomer(addressCount - 1);
                noticeboard3.setText(" ANGRY MINS: " + roundTo2(distanceStats2[2]) + "m" + " total KM:
"
                                + roundTo2(distanceStats2[3]) + "km");
                noticeboard3.paintImmediately(noticeboard3.getVisibleRect());
                showTotalStats();
        }
    };

    // --------------------------------------------------------------------------------


    void algorithm1() {
```

```
sendToNoticeboard("Working out best route...");

currentSelectedPoint = 0;
setSideListButton(currentSelectedPoint);

System.out.println("WORKING OUT BEST ROUTE...");

String dataBack = "";

Random rand = new Random();

// # STEP 1 - CALCULATE THE BASE SOLUTION:
// ------------------------------------------------------------
// NEAREST NEIGHBOUR

System.out.println("asfsf");
boolean[] visited = new boolean[addressCount];
visited[0] = true;
int currentClosest;
for (int i = 1; i < addressCount; i++)
        visited[i] = false;

deliveries[0] = 0;
int deliveryCount = 1;
int currentPlace = 0; // Apache Pizza

currentClosest = 1;
while (deliveryCount < addressCount) {
        if (visited[distanceMatrix[currentPlace][currentClosest].to]) {
                currentClosest++;
        } else {
                deliveries[deliveryCount] = distanceMatrix[currentPlace][currentClosest].to;
                visited[distanceMatrix[currentPlace][currentClosest].to] = true;
                deliveryCount++;
                currentPlace = distanceMatrix[currentPlace][currentClosest].to;
                System.out.println(deliveryCount + ". " + currentPlace);
                currentClosest = 1;
        }
}

checkStopWatch("BASE SOLUTION");

currentAngriness = getAngriness(deliveries);
randSwapCount = 0;
double initalAngriness = currentAngriness;
long t = checkStopWatch_QUICK();
//                                      while (t<900) {
for (int i = 0; i < 10000000; i++) {
        switchTwo();
}
t = checkStopWatch_QUICK();
System.out.println("T:" + t);
//                                      }
System.out.println("READY TO GO...." + checkStopWatch_QUICK());

currentAngriness = getAngriness(deliveries);
System.out.println("currentAngriness " + currentAngriness);

String bestDeliveryOutput = "";
for (int i = 1; i < addressCount; i++) {
        if (i > 1)
                bestDeliveryOutput += ",";
        bestDeliveryOutput += distanceMatrix[deliveries[i]][0].from;
}

addToClipboard(bestDeliveryOutput);
System.out.println("Added to clipboard");
sendToNoticeboard("Delivery sequence info has been pasted to your clipboard.", true);

deliveriesSorted = true;
drawOutRouteButton.setEnabled(true);
sideScrollBar.setValues(sideListScroll, (sideListScroll + 30), 0, addressCount - 1);
printAddressList();
```

```
            drawOutRoute();
            pasteInTextField.setText(bestDeliveryOutput);

            currentSelectedPoint = 1;
            setSideListButton(currentSelectedPoint);

        } // END: ALGORITHM 1

        //
=============================================================================================================


        // ---------------------------------------------------------------------------


        void algorithm2() {

            sendToNoticeboard("Working out best route...");

            currentSelectedPoint = 0;
            setSideListButton(currentSelectedPoint);

            System.out.println("WORKING OUT BEST ROUTE...");

            String dataBack = "";

            Random rand = new Random();

            // # STEP 1 - CALCULATE THE BASE SOLUTION:
            // ------------------------------------------------------------
            // NEAREST NEIGHBOUR

            System.out.println("asfsf");
            boolean[] visited = new boolean[addressCount];
            visited[0] = true;
            int currentClosest;
            for (int i = 1; i < addressCount; i++)
                visited[i] = false;

            deliveries[0] = 0;
            int deliveryCount = 1;
            int currentPlace = 0; // Apache Pizza

            currentClosest = 1;
            while (deliveryCount < addressCount) {
                if (visited[distanceMatrix[currentPlace][currentClosest].to]) {
                    currentClosest++;
                } else {
                    deliveries[deliveryCount] = distanceMatrix[currentPlace][currentClosest].to;
                    visited[distanceMatrix[currentPlace][currentClosest].to] = true;
                    deliveryCount++;
                    currentPlace = distanceMatrix[currentPlace][currentClosest].to;
                    System.out.println(deliveryCount + ". " + currentPlace);
                    currentClosest = 1;
                }
            }

            checkStopWatch("BASE SOLUTION");

            currentAngriness = getAngriness(deliveries);
            randSwapCount = 0;
            double initalAngriness = currentAngriness;
            long t = checkStopWatch_QUICK();



            for (int i = 0; i < 10000000; i++) {
                switchTwo();
            }

            t = checkStopWatch_QUICK();
            System.out.println("T:" + t);
            //                                    }
```

```
                System.out.println("READY TO GO...." + checkStopWatch_QUICK());

                currentAngriness = getAngriness(deliveries);
                System.out.println("currentAngriness " + currentAngriness);

                String bestDeliveryOutput = "";
                for (int i = 1; i < addressCount; i++) {
                        if (i > 1)
                                bestDeliveryOutput += ",";
                        bestDeliveryOutput += distanceMatrix[deliveries[i]][0].from;
                }

                addToClipboard(bestDeliveryOutput);
                System.out.println("Added to clipboard");
                sendToNoticeboard("Delivery sequence info has been pasted to your clipboard.", true);

                deliveriesSorted = true;
                drawOutRouteButton.setEnabled(true);
                sideScrollBar.setValues(sideListScroll, (sideListScroll + 30), 0, addressCount - 1);
                printAddressList();

                drawOutRoute();
                pasteInTextField.setText(bestDeliveryOutput);

                currentSelectedPoint = 1;
                setSideListButton(currentSelectedPoint);

        } // END: ALGORITHM 2

        //
==========================================================================================================


        public boolean wasVisited(Trip[] tree, int treeSize, int from, int to) {
                for (int i = 0; i < treeSize; i++) {
                        System.out.println(i + " - " + tree[i].from + " - " + from + " - " + tree[i].to + " -
" + to);
                        if (tree[i].from == from && tree[i].to == to) {
                                System.out.println("TRUE");
                                return true;
                        }

                }
                return false;
        }

        double[] timeToThisCustomer(int customer) {
                // ------------------------------------------------------------------------------

                double currentAngrinesss = 0;
                double angryseconds = 0;
                double elapsedtime = 0;
                double totalDistance = 0;
                double distance = 0;
                double thisTime = 0;
                System.out.println("LEN DM: " + distanceMatrix_unsorted.length);

                for (int i = 0; i < customer; i++)
                        System.out.print(deliveries[i] + " ");
                System.out.println("-------------------------------------------");
                for (int i = 0; i < customer; i++) {
                        int house1 = deliveries[i];
                        int house2 = deliveries[i + 1];
                        System.out.println("H1" + house1 + " " + house2);
                        distance = distanceMatrix_unsorted[house1][house2].metres;
                        //            distance=
haversineFormula(addresses[house1].gpsN,addresses[house1].gpsW,
                        //                    addresses[house2].gpsN,addresses[house2].gpsW);

                        totalDistance += distance;
                        thisTime = distance / deliverySpeedMPS;
                        elapsedtime += thisTime;
                        currentAngrinesss = Math.max(0.0, addresses[house2].minutesWaiting * 60 + elapsedtime
- 1800);
```

```
                        isCustomerAngry[i + 1] = ((boolean) (currentAngrinesss > 0));
                        angryseconds += currentAngrinesss;
                }
                double totalDeliveryTime = elapsedtime / 60.0;
                double totalAngryMinutes = angryseconds / 60.0;

                double[] returnData = { (currentAngrinesss / 60), distance / 1000, totalAngryMinutes,
totalDistance / 1000,
                                totalDeliveryTime, thisTime };

                return returnData;

        }

        public double getAngriness(int[] deliveries) {

                double angryseconds = 0;
                double elapsedtime = 0;
                double distance;
                int house1, house2;
                int len = addressCount - 1;

                for (int i = 0; i < len; i++) {
                        house1 = deliveries[i];
                        house2 = deliveries[i + 1];
                        distance = distanceMatrix_unsorted[house1][house2].metres;
                        //            double
distance=haversineFormula(addresses[house1].gpsN,addresses[house1].gpsW,
                        //                      addresses[house2].gpsN,addresses[house2].gpsW);
                        elapsedtime = elapsedtime + distance / deliverySpeedMPS;
                        angryseconds = angryseconds + Math.max(0.0, addresses[house2].minutesWaiting * 60 +
elapsedtime - 1800);
                }

                double totalAngryMinutes = angryseconds / 60.0;

                return totalAngryMinutes;

        } // END: getAngriness()

        public static double getDistance(Double lat1, Double lon1, Double lat2, Double lon2) {
                final int R = 6371; // Radius of the earth

                Double latDistance = toRad(lat2 - lat1);
                Double lonDistance = toRad(lon2 - lon1);
                Double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
                                + Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) * Math.sin(lonDistance / 2) *
Math.sin(lonDistance / 2);
                Double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
                Double distance = R * c;
                return distance * 1000; // distance in metres
        }

        public static Double toRad(Double value) {
                return value * Math.PI / 180;
        }

        public void showAngriness() {
                String dataBack = "";
                double clock = 0;
                double angriness;
                double distance;
                double totalAngriness = 0;
                for (int i = 1; i < addressCount; i++) {
                        distance = distanceMatrix[i - 1][i].metres;
                        clock += distance;
                        angriness = (addresses[deliveries[i - 1]].minutesWaiting + clock) - 30;
                        if (angriness > 0) {
                                totalAngriness += angriness;
                        }
                        dataBack += "CLOCK: " + (int) (clock) + " DISTANCE: " + i + " = "
                                        + ((double) Math.round((distance * 100)) / 100) + "; ANGRINESS: " + i +
" = "
```

```
                                        + ((double) Math.round((angriness * 100)) / 100) + ";  TOTAL ANGRINESS:
= " + totalAngriness
                                        + "\r\n";

                }

                dataBack += "TOTAL ANGRINESS: " + ((double) Math.round((totalAngriness * 100)) / 100) +
"\r\n";
                inputBox.setText(dataBack);

        }

        public void updateAngriness(double timeTaken) {
                for (int i = 0; i < addressCount; i++) {
                        addresses[i].angriness += timeTaken;
                }
        }

        public int findAngriest() {
                int angriest = 0;
                double angriest_val = addresses[0].angriness;
                for (int i = 1; i < addressCount; i++) {
                        if (addresses[i].angriness > angriest_val) {
                                angriest_val = addresses[i].angriness;
                                angriest = i;
                        }
                }
                return angriest;
        }

        public static void setCell(int x, int y) {

                if (grid[y][x]) {
                        gridButtons[y][x].setOpaque(true);

                        gridButtons[y][x].setBackground(Color.GREEN);
                        // gridButtons[y][x].setForeground(Color.WHITE);
                        // gridButtons[y][x].setText("X");
                } else {
                        gridButtons[y][x].setOpaque(false);
                        gridButtons[y][x].setBackground(null);
                        // gridButtons[y][x].setForeground(Color.BLACK);
                        // gridButtons[y][x].setText("O");
                }
                // gridButtons[y][x].setText(x+":"+y);

        } // setCell()

        private final ActionListener resetListener = new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        // EMPTY
                }
        };

        private final ActionListener loadListener = new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        int result = fileChooser.showOpenDialog(FastDeliver_window.this);
                        if (result == JFileChooser.APPROVE_OPTION) {
                                File selectedFile = fileChooser.getSelectedFile();
                                go_Load(selectedFile.getAbsolutePath(), true);
                        }

                }
        };

        private final ActionListener saveListener = new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        int result = fileChooser.showSaveDialog(FastDeliver_window.this);
                        if (result == JFileChooser.APPROVE_OPTION) {
                                File selectedFile = fileChooser.getSelectedFile();
                                go_Save(selectedFile.getAbsolutePath());
```

```
                }
            }
        };

        private final ActionListener playListener = new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                drawOutRoute();
            }
        };

        public void swap(int p, int p2) {
            int t = deliveries[p];
            deliveries[p] = deliveries[p2];
            deliveries[p2] = t;
        }


        public void switchTwo() {

            int p, p2;
            do {
                p = getRandom.nextInt(addressCount - 1) + 1; // dont switch the pizza shop!
                p2 = getRandom.nextInt(addressCount - 1) + 1;
            } while (p == p2);
            swap(p, p2);
            double a = getAngriness(deliveries);
            if (a >= currentAngriness) {
                swap(p, p2);
            } else {
                currentAngriness = a;
                lastChanged = p;
                System.out.println(p + " swap " + (p + 1) + ", " + currentAngriness);
                randSwapCount++;
            }

        }

public void switchFour() {

            double thisA, bestA= Double.MAX_VALUE;
            int[] p = new int[4];
            int best = 0;

            for (int i=0; i<4; i++) {
                p[i] = getRandom.nextInt(addressCount - 1) + 1; // dont switch the pizza shop!

            }
                                                                // ABCD
            swap(p[0], p[1]);
            swap(p[2], p[3]);
            thisA = getAngriness(deliveries);
            if (thisA < bestA) {
                best = 1;
                bestA = thisA;
            }
                                                                // BADC

            swap(p[0], p[3]);
            swap(p[1], p[2]);
            thisA = getAngriness(deliveries);
            if (thisA < bestA) {
                best = 2;
                bestA = thisA;
            }
                                                                // CDAB

            swap(p[0], p[1]);
            swap(p[2], p[3]);
            thisA = getAngriness(deliveries);
            if (thisA < bestA) {
                best = 3;
                bestA = thisA;
            }
```

```
                                                        // DCBA
                if (best == 3) return;

                swap(p[0], p[1]);
                swap(p[2], p[3]);

                                                        // CDAB
                if (best == 2) return;

                swap(p[0], p[3]);
                swap(p[1], p[2]);

                if (best == 1) return;
                                                        // BADC
                swap(p[0], p[1]);
                swap(p[2], p[3]);                   // ABCD


        } // END: SWITCH-FOUR


        public void addToClipboard(String text) {
                StringSelection stringSelection = new StringSelection(text);
                Clipboard clpbrd = Toolkit.getDefaultToolkit().getSystemClipboard();
                clpbrd.setContents(stringSelection, null);
        }

        public String retrieveClipboard() {
                try {
                        return (String)
Toolkit.getDefaultToolkit().getSystemClipboard().getData(DataFlavor.stringFlavor);
                } catch (HeadlessException | UnsupportedFlavorException | IOException e) {
                        System.out.println("SOMETHING ELSE IN THE CLIPBOARD!");
                        //                      e.printStackTrace();
                        return "";
                }

        }

        private final ActionListener goRetrieveDataFromClipboard = new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        String clipBoardData = retrieveClipboard();
                        if (processDataIn(clipBoardData, "Data found in clipboard.") < 1) {
                                sendToNoticeboard("NO DATA IN CLIPBOARD", false);
                        }
                }
        };

        public static String createRandomAddress() {

                Random r = new Random();
                String[] streetNamesA = { "College", "Melody", "Hawthorn", "Sycamore", "Cottage", "Lake",
"Greenfields",
                                "Mountain", "Stream", "Lake" };
                String[] streetNamesB = { "Rise", "Drive", "Avenue", "Gardens", "Green", "Park", "Heights",
"Valley", "View",
                                "Manor", "Terrace" };
                String[] RoadNamesA = { "Dublin", "Main", "Wide", "Narrow", "Scenic", "Winding", "Straight",
"Side" };
                String[] RoadNamesB = { "Road", "Road", "Way" };
                String[] towns = { "Maynooth", "Celbridge", "Leixlip", "Kilcock", "Clane" };
                String line1 = "";
                int houseNumber = r.nextInt(401);
                if (houseNumber > 0) {
                        line1 = houseNumber + " ";
                }
                line1 += streetNamesA[r.nextInt(streetNamesA.length)] + ' ' +
streetNamesB[r.nextInt(streetNamesB.length)];

                String townAndCounty = towns[r.nextInt(towns.length)];

                return line1 + " " + townAndCounty;

        } // END: createRandomAddress()
```

```
	void scatterRandomPoints() {
		String dataBack = "";
		double gpsN_lowest = 53.2908;
		double gpsN_highest = 53.39847;
		double gpsW_lowest = -6.67836;
		double gpsW_highest = -6.48193;
		double gpsN_range1 = 0.10767;
		double gpsW_range1 = 0.19643;

		Random rand = new Random();
		int len = rand.nextInt(90) + 11; // 101;
		deliveries = new int[105];
		System.out.println("LEN : " + len);
		deliveriesSorted = false;
		drawOutRouteButton.setEnabled(false);
		addressCount = len;
		// "0, Apache Pizza,0, , \r\n"
		addresses[0] = pizzaPlace;
		deliveries[0] = 0;
		for (int i = 1; i < len; i++) {
			deliveries[i] = i;
			addresses[i] = new Address(i, createRandomAddress(), 0, (gpsN_lowest +
(rand.nextDouble() * gpsN_range1)),
					(gpsW_lowest + (rand.nextDouble() * gpsW_range1)));
			dataBack = "" + addresses[i].minutesWaiting + "---- [" + addresses[i].number + "] (" +
addresses[i].gpsN
					+ "," + addresses[i].gpsW + ") " + addresses[i].postalAddress + "\r\n";
		}
		addressCount = len;

		// find range of grid;
		gpsN_lowest = addresses[0].gpsN;
		gpsN_highest = addresses[0].gpsN;
		gpsW_lowest = addresses[0].gpsW;
		gpsW_highest = addresses[0].gpsW;

		for (int i = 1; i < addressCount; i++) {
			if (addresses[i].gpsN < gpsN_lowest) {
				gpsN_lowest = addresses[i].gpsN;
			}
			if (addresses[i].gpsN > gpsN_highest) {
				gpsN_highest = addresses[i].gpsN;
			}
			if (addresses[i].gpsW < gpsW_lowest) {
				gpsW_lowest = addresses[i].gpsW;
			}
			if (addresses[i].gpsW > gpsW_highest) {
				gpsW_highest = addresses[i].gpsW;
			}
		}

		double gridx = haversineFormula(gpsW_lowest, gpsN_lowest, gpsW_highest, gpsN_lowest);
		double gridy = haversineFormula(gpsW_lowest, gpsN_lowest, gpsW_lowest, gpsN_highest);

		dataBack = "GPS_N low/high: " + gpsN_lowest + " to " + gpsN_highest + "\r\n";
		dataBack += "GPS_W low/high: " + gpsW_lowest + " to " + gpsW_highest + "\r\n";
		gpsW_range1 = gpsW_highest - gpsW_lowest;
		gpsN_range1 = gpsN_highest - gpsN_lowest;
		dataBack += "GPS_W RANGE: " + (gpsW_range1) + "\r\n";
		;
		dataBack += "GPS_N RANGE: " + (gpsN_range1) + "\r\n";
		;

		dataBack += "GRID X: " + gridx + " km" + "\r\n";
		dataBack += "GRID Y: " + gridy + " km" + "\r\n";

		System.out.println("gpsN_lowest-gpsN_highest " + gpsN_lowest + " - " + gpsN_highest);
		System.out.println("gpsW_lowest-gpsW_highest " + gpsW_lowest + " - " + gpsW_highest);
		System.out.println("gpsN_range, gpsW_range: " + gpsN_range1 + ", " + gpsW_range1);

		inputBox.setText(dataBack);

		// JFrame frame = new JFrame();
```

```
        // JPanel mapFrame = new JPanel();

        // do you drawing somewhere else, maybe a different thread

        g.setColor(Color.yellow);
        int gridXSize = 1280;
        int gridYSize = 1040;
        int lastX = 0;
        int lastY = 0;
        double gpsScale = (gpsW_range1 > gpsN_range1) ? gpsW_range1 : gpsN_range1;

        for (int i = 0; i < addressCount; i++) {
                int thisX = 50 + (int) (((addresses[i].gpsW - gpsW_lowest) / gpsScale) * gridXSize);
                int thisY = 50 + (int) (((addresses[i].gpsN - gpsN_lowest) / gpsScale) * gridYSize);
                addresses[i].x = thisX;
                addresses[i].y = thisY;
                g.fillRect(thisX, thisY, 8, 8);
                // .fillRect(x, y, width, height);
        }
        for (int i = 0; i < addressCount; i++) {
                deliveries[i] = i;
        }
        for (int i = 0; i < addressCount; i++)
                System.out.print(deliveries[i] + " ");
        System.out.println("-----------------------------------------------" + addressCount);

        setMatrixEtc();
        currentSelectedPoint = 1;
        sideListScroll = 0;
        sideScrollBar.setValues(0, (sideListScroll + 30), 0, addressCount - 1);
        printAddressList();
        refreshMap(addressCount, false);
        showTotalStats();
        sendToNoticeboard((addressCount - 1) + " new Random Addreses added.");

} // END: scatterRandomPoints()

private final ActionListener setRandomPoints = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                scatterRandomPoints();
        }
};

private final ActionListener getDataFromInputBox = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                //
                String dataIn = pasteInTextField.getText();

                processDataIn(dataIn, "Data found in clipboard");
        }
};

private final ActionListener goToLastOrNextPoint = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                //
                JButton source = (JButton) e.getSource();
                int value = Integer.parseInt((String) source.getClientProperty("value"));
                goUpDown(value);
        }
};

// ######################################################################

// ######################################################################

public JPanel buildMapFrame() {

        image = new BufferedImage(screenDimensions.width - 604, screenDimensions.height - 94,
                        BufferedImage.TYPE_INT_RGB);
        System.out.println("CANVAS: ");
        System.out.println(screenDimensions.width - 300);
        System.out.println(screenDimensions.height - 100);

        JPanel panel = new JPanel();
```

```
                canvas = new JPanel() {
                        @Override
                        protected void paintComponent(Graphics g) {
                                super.paintComponent(g);
                                Graphics2D g2d = (Graphics2D) g;
                                g2d.setStroke(new BasicStroke(6.0F));
                                g.drawImage(image, 0, 0, Color.WHITE, this);
                        }
                };
                g = (Graphics2D) image.getGraphics();
                mapFrame = new JPanel();

                // canvas.setSize(1200,1000);
                canvas.setLocation(new Point(0, 0));

                canvas.setBackground(new Color(0x23234f));
                canvas.setVisible(true);

                //              int newImageWidth = imageWidth * zoomLevel;
                //              int newImageHeight = imageHeight * zoomLevel;
                //              BufferedImage resizedImage = new BufferedImage(newImageWidth , newImageHeight,
);
                //              Graphics2D g = resizedImage.createGraphics();
                //              g.drawImage(originalImage, 0, 0, newImageWidth , newImageHeight , null);
                //              g.dispose();

                //        ImageIcon icon=new ImageIcon(img);
                //        JFrame frame=new JFrame();
                //
                //        JLabel image_=new JLabel();
                //        image_.setIcon(icon);
                //        frame.setVisible(true);

                mapFrame.setLayout(new BorderLayout()); // <== make panel fill frame
                // mapFrame.add(canvas, BorderLayout.CENTER);
                mapFrame.setVisible(true);

                //        panel.setLayout(new GridLayout(1,2));
                // panel.add(mapFrame);

                return canvas;

        }

        public JPanel buildTopButtons() {

                JPanel panel = new JPanel(new GridLayout(1, 9));

                // ---------------------------------------------------

                findMyRoute = new JButton("FIND MY ROUTE");
                findMyRoute.setOpaque(true);
                findMyRoute.setBackground(new Color(0x167a31));
                findMyRoute.setForeground(Color.white);
                findMyRoute.addActionListener(goFindMyRoute);
                panel.add(findMyRoute);

                Border lBorder = BorderFactory.createLineBorder(Color.BLACK);

                algorithmChooser = new JComboBox<>(algorithmList);
                algorithmChooser.addActionListener(setAgorithm);
                panel.add(algorithmChooser);
                algorithmChooser.setBorder(
                                BorderFactory.createCompoundBorder(lBorder, BorderFactory.createEmptyBorder(5,
5, 5, 5)));

                pasteInTextField = new JTextArea("");
                pasteInTextField.setLineWrap(true);


                pasteInTextField.setFont(pasteInTextField.getFont().deriveFont(9f));
                pasteInTextField.setBorder(
                        BorderFactory.createCompoundBorder(lBorder, BorderFactory.createEmptyBorder(5, 5, 5,
5)));
```

```
        panel.add(pasteInTextField);



        drawOutRouteButton = new JButton("Redraw route");
        drawOutRouteButton.addActionListener(playListener);
        drawOutRouteButton.setEnabled(false);
        panel.add(drawOutRouteButton);

        lastPointButton = new BasicArrowButton(BasicArrowButton.WEST);
        lastPointButton.putClientProperty("value", "-1");
        lastPointButton.addActionListener(goToLastOrNextPoint);
        panel.add(lastPointButton);

        nextPointButton = new BasicArrowButton(BasicArrowButton.EAST);
        nextPointButton.putClientProperty("value", "1");
        nextPointButton.addActionListener(goToLastOrNextPoint);
        panel.add(nextPointButton);

        buttonOnce = new JButton("RANDOM VALUES");
        buttonOnce.addActionListener(setRandomPoints);
        panel.add(buttonOnce);

        buttonSave = new JButton("Save");
        buttonSave.addActionListener(saveListener);
        panel.add(buttonSave);

        buttonLoad = new JButton("Load");
        buttonLoad.addActionListener(loadListener);
        panel.add(buttonLoad);

        buttonExit = new JButton("EXIT");
        buttonExit.setMnemonic(KeyEvent.VK_C);
        buttonExit.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        System.exit(0);
                }
        });
        panel.add(buttonExit);

        return panel;
}

// ###############################################

public void sendToNoticeboard(String st) {
        sendToNoticeboard(st,false);
}

public void sendToNoticeboard(String st, boolean hilight) {
        noticeboard.setVisible(true);
        noticeboard.setText(st);
        if (hilight) {
                noticeboard.setBackground(Color.yellow);
                noticeboard.setOpaque(true);
        } else {
                noticeboard.setBackground(null);
                noticeboard.setOpaque(false);
        }
        noticeboard.paintImmediately(noticeboard.getVisibleRect());

}

public void setSideListButton(int location) {
        currentSelectedPoint = location;
        int buttonClicked = location - sideListScroll - 1;
        if (sideListLastClicked > -1) {
                sideList[sideListLastClicked].setOpaque(true);
                sideList[sideListLastClicked].setBackground(null);
        }
        // if (sideListLastClicked != buttonClicked) {
        if (buttonClicked >= 0 && buttonClicked < 30) {
                sideList[buttonClicked].setOpaque(true);
                sideList[buttonClicked].setBackground(Color.YELLOW);
```

```
                        sideListLastClicked = buttonClicked;

                        sendToNoticeboard("# "+ addresses[deliveries[location]].number + " GPS: " +
addresses[deliveries[location]].gpsN + "N "
                                        + addresses[deliveries[location]].gpsW + "W ");

                }
                // }
        }

        void setSelectedPoint(int location) {
                if (location > 0 && location < addressCount) {

                        int[] xyZoomed = pointByZoom(addresses[deliveries[location]].x,
addresses[deliveries[location]].y);
                        refreshMap(addressCount, deliveriesSorted);
                        canvas.repaint();
                        g.setColor(Color.black);
                        g.fillOval(xyZoomed[0] - 8, xyZoomed[1] - 8, 16, 16);
                        g.setColor(Color.yellow);
                        g.fillOval(xyZoomed[0] - 6, xyZoomed[1] - 6, 12, 12);

                }
                double[] distanceStats = timeToThisCustomer(location);
                noticeboard2.setText("L: " + location + " ANG:" + roundTo2(distanceStats[0]) + "m" + " DIST: "
                                + roundTo2(distanceStats[1]) + "km" + " t-ANG: " + roundTo2(distanceStats[2]) +
"m" + " t-KM: "
                                + roundTo2(distanceStats[3]) + "km");

                showTotalStats();
        }

        double roundTo2(double n) {
                return ((double) ((int) (n * 100)) / 100);
        }

        void showTotalStats() {
                double[] distanceStats2 = timeToThisCustomer(addressCount - 1);
                System.out.println("DISTANCE STATS");
                noticeboard3.setText(" AM:" + roundTo2(distanceStats2[2]) + "m" + "TKM: " +
roundTo2(distanceStats2[3]) + "km"
                                + "TT: " + roundTo2(distanceStats2[4]) + "MINS");
                noticeboard3.paintImmediately(noticeboard3.getVisibleRect());
                statsBoard1.setText(roundTo2(distanceStats2[2]) + " ANGRY MINUTES");
                statsBoard2.setText(roundTo2(distanceStats2[3]) + "km JOURNEY LENGTH");
                statsBoard3.setText(roundTo2(distanceStats2[4]) + "m JOURNEY MINUTES");
                statsBoard1.paintImmediately(statsBoard1.getVisibleRect());
                statsBoard2.paintImmediately(statsBoard2.getVisibleRect());
                statsBoard1.paintImmediately(statsBoard1.getVisibleRect());

        }

        ActionListener sideListListener = new ActionListener() {
                public void actionPerformed(ActionEvent e) {

                        JButton source = (JButton) e.getSource();
                        int id = Integer.parseInt((String) source.getClientProperty("id"));

                        System.out.print("ID CLICKED" + id);

                        if (id >= 0 && id < 30) {
                                requestFocus();
                                int location = id + sideListScroll + 1;
                                setSideListButton(location);
                                setSelectedPoint(location);

                        }

                }
        };

        ActionListener setAgorithm = new ActionListener() {
                public void actionPerformed(ActionEvent e) {
```

```
                String selectedITEM = (String) algorithmChooser.getSelectedItem();
                System.out.println("SELECTED: " + selectedITEM);

                int picked = Arrays.asList(algorithmList).indexOf(selectedITEM);
                System.out.println("PICKED: " + picked);

                algorithmMethod = picked;

        }
};

ActionListener testListener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {

                JButton source = (JButton) e.getSource();
                int id = Integer.parseInt((String) source.getClientProperty("id"));
                boolean noAddresses = false;

                if (id == 1) {
                        if (addressCount > 1) {
                                goTest(1);
                        } else {
                                noAddresses = true;
                        }
                } else if (id == 2) {
                        if (addressCount > 1) {
                                goTest(2);
                        } else {
                                noAddresses = true;
                        }
                } else if (id == 3) {
                        if (addressCount > 1) {
                                goTest(3);
                        } else {
                                noAddresses = true;
                        }
                }

                if (noAddresses) {
                        sendToNoticeboard("PLEASE ADD SOME ADDRESSES!");
                }


        }
};


int translateX(int x) {
        return (int) ((double) ((x + (((double) mapX) * 0.75)) * mapZoom));
}

int translateY(int y) {
        return (int) ((double) ((y + (((double) mapY) * 0.75)) * mapZoom));
}

int[] removeFromPile(int[] pile, int removeThis) {

        int[] newPile = new int[pile.length - 1];

        for (int i = 0; i < removeThis; i++) {
                newPile[i] = pile[i];
        }

        int len = pile.length - 1;

        for (int i = removeThis; i < len; i++) {
                newPile[i] = pile[i + 1];
        }

        return newPile;
}

int[] addToOnTree(int[] onTree, int addThis) {
```

```
        int[] newOnTree = new int[onTree.length + 1];

        for (int i = 0; i < onTree.length; i++) {
                newOnTree[i] = onTree[i];
        }
        newOnTree[onTree.length] = addThis;

        return newOnTree;
}

int[] closestAvailablePoint(int[] pile, int[] onTree) {

        int closestPoint = -1;
        int closestBranch = -1;
        double thisDistance;
        double closestDistance = Double.MAX_VALUE;
        for (int b = 0; b < onTree.length; b++) {

                for (int i = 0; i < pile.length; i++) {
                        thisDistance = distanceMatrix_unsorted[onTree[b]][pile[i]].metres;
                        if (thisDistance < closestDistance) {
                                closestBranch = b;
                                closestPoint = i;
                                closestDistance = thisDistance;
                        }
                }
        } // outer loop

        int[] points = { closestBranch, closestPoint };
        return points;

} // find closest branch

void drawBranch(int pointA, int pointB, Color colour1, Color colour2) {
        drawBranch( pointA,  pointB,  colour1,  colour2, 5, 8);
}

void drawBranch(int pointA, int pointB, Color colour1, Color colour2, int stroke1, int stroke2) {
        int x1 = translateX(addresses[pointA].x);
        int y1 = translateY(addresses[pointA].y);
        int x2 = translateX(addresses[pointB].x);
        int y2 = translateY(addresses[pointB].y);
        BasicStroke basicStroke = new BasicStroke(stroke1);
        BasicStroke thickerStroke = new BasicStroke(stroke2);
        g.setStroke(thickerStroke);
        g.setColor(colour2);
        g.drawLine(x1, y1, x2, y2);
        g.setStroke(basicStroke);
        g.setColor(colour1);
        g.drawLine(x1, y1, x2, y2);
}

int[] findOddDegreeVertices(int[] verteces) {
        int odvCount = 0;
        System.out.print("findOddDegreeVertices: ");
        for (int v = 0; v < verteces.length; v++) {
                if (verteces[v] % 2 != 0) {
                        odvCount++;
                }
        }

        int[] oddVertices = new int[odvCount];

        int ov = 0;
        for (int v = 0; v < verteces.length; v++) {
                if (verteces[v] % 2 != 0) {
                        oddVertices[ov] = v;
                        ov++;
                        System.out.print(v + "; ");
                }
        }
        System.out.println(" COUNT: " + odvCount);
        return oddVertices;
```

```
} // END: findOddDegreeVertices()

int[][] almostPerfectMatching(int[] oddVertices) {
        double lowestCost = Double.MAX_VALUE;

        // FIND ANY MATCH(
        int matchesRequired = oddVertices.length / 2;
        int[][] pairs = new int[matchesRequired][2];

        if (matchesRequired < 1)
                return null;

        int verticesLeft = oddVertices.length;
        int node1, node2, lastVertex;

        int[] verticesCopy = new int[oddVertices.length];
        for (int i = 0; i < verticesCopy.length; i++) {
                verticesCopy[i] = i;
        }
        lastVertex = verticesLeft - 1;
        int paired = 0;

        while (verticesLeft > 2) {
                node1 = getRandom.nextInt(verticesLeft);
                pairs[paired][0] = verticesCopy[node1];
                if (node1 != lastVertex) {
                        verticesCopy[node1] = verticesCopy[lastVertex];
                }
                verticesLeft--;
                lastVertex--;

                node2 = getRandom.nextInt(verticesLeft);
                pairs[paired][1] = verticesCopy[node2];
                if (node2 != lastVertex) {
                        verticesCopy[node2] = verticesCopy[lastVertex];
                }
                lastVertex--;

                paired++;
                verticesLeft--;
        }

        pairs[paired][0] = verticesCopy[0];
        pairs[paired][1] = verticesCopy[1];

        return pairs;
}

// ------------------------------------------------------------------------------------------

int[][] nearestNeighbourPairMatching(int[] oddVertices) {
        double lowestCost = Double.MAX_VALUE;

        // FIND ANY MATCH(
        int matchesRequired = oddVertices.length / 2;
        int[][] pairs = new int[matchesRequired][2];

        if (matchesRequired < 1)
                return null;

        int verticesLeft = oddVertices.length;
        int node1, node2, lastVertex;

        int[] verticesCopy = new int[oddVertices.length];
        for (int i = 0; i < verticesCopy.length; i++) {
                verticesCopy[i] = i;
        }
        lastVertex = verticesLeft - 1;
        int paired = 0;
        double shortestTrip, thisTrip;
        int[] shortestPair = { -1, -1 };

        while (verticesLeft > 2) {
                shortestTrip = Double.MAX_VALUE;
```

```
                        shortestPair[0] = -1;
                        shortestPair[1] = -1;
                        for (int p1 = 0; p1 < verticesLeft; p1++) {
                                for (int p2 = 0; p2 < verticesLeft; p2++) {
                                        if (p1 != p2) {
                                                thisTrip =
distanceMatrix_unsorted[oddVertices[p1]][oddVertices[p2]].metres;
                                                if (thisTrip < shortestTrip) {
                                                        shortestTrip = thisTrip;
                                                        shortestPair[0] = p1;
                                                        shortestPair[1] = p2;
                                                }
                                        }
                                }
                        }

                        node1 = shortestPair[0];
                        node2 = shortestPair[1];
                        pairs[paired][0] = verticesCopy[node1];
                        pairs[paired][1] = verticesCopy[node2];

                        if (node1 != lastVertex) {
                                verticesCopy[node1] = verticesCopy[lastVertex];
                        }
                        verticesLeft--;
                        lastVertex--;

                        if (node2 != lastVertex) {
                                verticesCopy[node2] = verticesCopy[lastVertex];
                        }
                        lastVertex--;

                        paired++;
                        verticesLeft--;
                }

                pairs[paired][0] = verticesCopy[0];
                pairs[paired][1] = verticesCopy[1];

                return pairs;
        }

        // --------------------------------------------------------------------------------------------

        double getCostOfPairs(int[][] pairs, int[] oddVertices) {
                double totalCost = 0;
                for (int i = 0; i < pairs.length; i++) {
                        totalCost +=
distanceMatrix_unsorted[oddVertices[pairs[i][0]]][oddVertices[pairs[i][1]]].metres;
                }
                return totalCost;
        }

        int[][] getRandomBestPairs(int[] oddVertices, int tries) {
                int[][] bestPairs = null;
                double lowestCost = Double.MAX_VALUE;
                double thisCost;

                for (int q = 0; q < 3000001; q++) {
                        int[][] randMatches = almostPerfectMatching(oddVertices);
                        thisCost = getCostOfPairs(randMatches, oddVertices);
                        if (thisCost < lowestCost) {
                                lowestCost = thisCost;
                                bestPairs = randMatches; // saves address
                                System.out.println(" Better COST: " + thisCost);
                        }
                }
                return bestPairs;
        }

        int[][] trySwappingPairs(int[] oddVertices, int tries) {
                int[][] bestPairs = null;
                double lowestCost = Double.MAX_VALUE;
                double thisCost;
```

```
        for (int q = 0; q < 3000001; q++) {
                int[][] randMatches = almostPerfectMatching(oddVertices);
                thisCost = getCostOfPairs(randMatches, oddVertices);
                if (thisCost < lowestCost) {
                        lowestCost = thisCost;
                        bestPairs = randMatches; // saves address
                        System.out.println(" Better COST: " + thisCost);
                }
        }
        return bestPairs;
}

// --------------------------------------------------------------------------------

int[] getHamiltonianCircuit() {

        // start with pizza-place vertex:

        boolean[] beenHere = new boolean[vertexCount];
        int hamiltonian[] = new int[addressCount];
        int hCount = 0;

        makeEulerianPath(0);
        for (int i = 0; i < currentTourIndex; i++) {
                if (!beenHere[tour[i][0]]) {
                        hamiltonian[hCount] = tour[i][0];
                        beenHere[tour[i][0]] = true;
                        hCount++;
                }
        }

        return hamiltonian;

}

void removeEdge(Integer u, Integer v) {
        edge[u].remove(v);
        edge[v].remove(u);
}

void addEdge(Integer u, Integer v) {
        edge[v].add(u);
        edge[u].add(v);
}

void makeEulerianPath(int u) {

        for (int i = 0; i < edge[u].size(); i++) {
                int v = edge[u].get(i);

                if (edgeCheck(u, v)) {
                        tour[currentTourIndex][0] = u;
                        tour[currentTourIndex][1] = v;
                        currentTourIndex++;
                        removeEdge(u, v);
                        makeEulerianPath(v);
                }
        }
}

boolean edgeCheck(int u, int v) { // is edge next

        if (edge[u].size() == 1)
                return true;

        boolean[] beenHere = new boolean[vertexCount];
        int rv1 = reachableVertices(u, beenHere);

        removeEdge(u, v);
        beenHere = new boolean[vertexCount];
        int rv2 = reachableVertices(u, beenHere);

        addEdge(u, v);
```

```
                return ((boolean) (rv1 <= rv2));

        }

        int reachableVertices(int v, boolean[] beenHere) {

                int rv = 1;
                beenHere[v] = true;
                for (int x : edge[v])
                        if (!beenHere[x])
                                rv += reachableVertices(x, beenHere);
                return rv;
        }

        // ------------------------------------------------------------------------------

        void paintParts(int whatToPaint, boolean showNodes) {

                if (minimumSpanningTree == null)
                        return;

                // 1. Minimum Spanning Tree;
                // 2. Minimum Perfect Matching

                double mapX_ = ((double) mapX) * 0.75;
                double mapY_ = ((double) mapY) * 0.75;

                int xD = (int) (((double) screenDimensions.width) * (mapZoom * 0.82) * mapFit);
                int yD = (int) (((double) screenDimensions.height) * (mapZoom * 0.82) * mapFit);

                System.out.println(xD + " x " + yD);
                System.out.println("mapX; mapY: " + mapX + " ; " + mapY);

                g.drawImage(img, mapX, mapY, xD, yD, 0, 0, screenDimensions.width - mapX,
screenDimensions.height - mapY, null);

                if (whatToPaint == 2 && eulerianMultigraph != null) {
                        for (int i = 0; i < eulerianMultigraph.length; i++) {
                                drawBranch(eulerianMultigraph[i][0], eulerianMultigraph[i][1], new
Color(0x167a31), Color.white,7,10);

                        }

                }

                if (whatToPaint > 0) {
                        for (int i = 0; i < minimumSpanningTree.length; i++) {
                                drawBranch(minimumSpanningTree[i][0], minimumSpanningTree[i][1], Color.black,
Color.white);
                        }
                        canvas.repaint();
                }

                if (showNodes) {
                        int x2, y2;
                        for (int i = 0; i < addressCount; i++) {
                                x2 = (int) ((double) ((addresses[i].x + mapX_) * mapZoom));
                                y2 = (int) ((double) ((addresses[i].y + mapY_) * mapZoom));

                                if (vDegreeCount[i] % 2 != 0) {
                                        g.setColor(Color.white);
                                        g.fillOval(x2 - 8, y2 - 8, 16, 16);
                                        g.setColor(Color.GREEN);
                                        g.fillOval(x2 - 6, y2 - 6, 12, 12);
                                } else {
                                        g.setColor(Color.black);
                                        g.fillOval(x2 - 8, y2 - 8, 16, 16);
                                        g.setColor(Color.CYAN);
                                        g.fillOval(x2 - 6, y2 - 6, 12, 12);
                                }

                        }
                }
```

```
} // END: paintParts

void goTest(int showHowMuch) {
	goTest(showHowMuch, false);
}

void goTest(int showHowMuch, boolean addRandomness) {

	int[] point;

	// STEP 1: CALCULATE THE MINIMUM SPANNING TREE :

	int nodeCount = 0;
	minimumSpanningTree = new int[addressCount - 1][2];
	// vertex count - always node count +1
	vDegreeCount = new int[addressCount];
	// initialise array
	for (int n = 0; n < vDegreeCount.length; n++)
		vDegreeCount[n] = 0;

	Random r = new Random();
	int firstPoint, lastPoint, newPoint;
	int[] pile = new int[addressCount];
	for (int i = 0; i < addressCount; i++) {
		pile[i] = i;
	}

	firstPoint = r.nextInt(addressCount);
	int[] onTree = { firstPoint };

	pile = removeFromPile(pile, firstPoint);

	point = closestAvailablePoint(pile, onTree);
	newPoint = point[1];

	onTree = addToOnTree(onTree, pile[newPoint]);
	pile = removeFromPile(pile, newPoint);

	int closestPoint = distanceMatrix[firstPoint][1].to;
	System.out.println("CLOS" + firstPoint + " _ " + closestPoint);

	minimumSpanningTree[nodeCount][0] = firstPoint;
	minimumSpanningTree[nodeCount][1] = closestPoint;
	vDegreeCount[minimumSpanningTree[nodeCount][0]]++;
	vDegreeCount[minimumSpanningTree[nodeCount][1]]++;
	nodeCount++;

	// now go through tree and get closest from pile

	while (pile.length > 0) {

		point = closestAvailablePoint(pile, onTree);

		System.out.println("ADD " + pile[point[1]]);

		minimumSpanningTree[nodeCount][0] = onTree[point[0]];
		minimumSpanningTree[nodeCount][1] = pile[point[1]];
		vDegreeCount[minimumSpanningTree[nodeCount][0]]++;
		vDegreeCount[minimumSpanningTree[nodeCount][1]]++;
		nodeCount++;

		onTree = addToOnTree(onTree, pile[point[1]]);
		pile = removeFromPile(pile, point[1]);

	}

	if (showHowMuch == 1) {
		paintParts(1, true);
		return;
	}

	// STEP 2: FIND ODD DEGREE VERTICES:

	System.out.println("NODES: " + nodeCount);
```

```
		int[] oddVertices = findOddDegreeVertices(vDegreeCount);
		System.out.println("oddVertices " + oddVertices.length);

		int[][] bestPairs = null;

		System.out.println("getRandomBestPairs");
		bestPairs = getRandomBestPairs(oddVertices, 6000000);
		// System.out.println("nearestNeighbourPairMatching");
		//              bestPairs = nearestNeighbourPairMatching(oddVertices);

		double thisCost = getCostOfPairs(bestPairs, oddVertices);
		System.out.println("COST::::" + thisCost);

		int eularianLength = minimumSpanningTree.length + bestPairs.length;
		System.out.println("NEW LENGTH: " + (eularianLength));

		eulerianMultigraph = new int[eularianLength][2];
		for (int i = 0; i < minimumSpanningTree.length; i++) {
			eulerianMultigraph[i][0] = minimumSpanningTree[i][0];
			eulerianMultigraph[i][1] = minimumSpanningTree[i][1];
		}
		for (int i = 0; i < bestPairs.length; i++) {
			eulerianMultigraph[minimumSpanningTree.length + i][0] = oddVertices[bestPairs[i][0]];
			eulerianMultigraph[minimumSpanningTree.length + i][1] = oddVertices[bestPairs[i][1]];
		}

		int[] vDegreeCount2 = new int[addressCount];
		vertexCount = eulerianMultigraph.length;
		for (int i = 0; i < eulerianMultigraph.length; i++) {
			System.out.println("graph.addEdge(" + eulerianMultigraph[i][0] + ", " +
eulerianMultigraph[i][1] + ");");
			vDegreeCount2[eulerianMultigraph[i][0]]++;
			vDegreeCount2[eulerianMultigraph[i][1]]++;
		}
		int oddCount = 0;
		for (int i = 0; i < addressCount; i++) {
			if (vDegreeCount2[i] % 2 != 0) {
				oddCount++;
				System.out.println("ODD: " + i + " " + vDegreeCount2[i]);
			}
		}
		System.out.println("ODD NOW: " + oddCount);

		System.out.println("MATCHES");
		// ----------------------------------------------------

		if (showHowMuch == 2) {
			paintParts(2, true);
			return;
		}

		tour = new int[vertexCount][2];
		currentTourIndex = 0;

		edge = new ArrayList[vertexCount];
		for (int i = 0; i < vertexCount; i++) {
			edge[i] = new ArrayList<>();
		}

		for (int i = 0; i < eulerianMultigraph.length; i++) {
			addEdge(eulerianMultigraph[i][0], eulerianMultigraph[i][1]);
		}

		int[] hamiltonian = getHamiltonianCircuit();


		currentAngriness = getAngriness(hamiltonian);

		for (int i = 1; i < hamiltonian.length; i++) {
			System.out.print(hamiltonian[i] + ",");
		}

		deliveries = hamiltonian;
```

```
        if (addRandomness) {
                for (int i = 0; i < 10000000; i++) {
                        switchTwo();
                }
        }


        currentAngriness = getAngriness(deliveries);
        System.out.println("currentAngriness " + currentAngriness);
        String bestDeliveryOutput = "";
        for (int i = 1; i < addressCount; i++) {
                if (i > 1)
                        bestDeliveryOutput += ",";
                bestDeliveryOutput += addresses[deliveries[i]].number;
        }

        addToClipboard(bestDeliveryOutput);
        System.out.println("Added to clipboard");
        sendToNoticeboard("Delivery sequence info has been pasted to your clipboard.", true);

        deliveriesSorted = true;
        drawOutRouteButton.setEnabled(true);

        sideScrollBar.setValues(sideListScroll, (sideListScroll + 30), 0, addressCount - 1);
        printAddressList();

        drawOutRoute();
        pasteInTextField.setText(bestDeliveryOutput);

        this.requestFocusInWindow();
}

double[] pointToGPS(int x, int y) {

        // x = (int)((double)((x+(((double)mapX)*0.75)) * mapZoom));
        // y = (int)((double)((y+(((double)mapY)*0.75)) * mapZoom));

        double gps_W = 1312 - (x / mapZoom) + (mapX * 0.75);
        double gps_N = 1080 - (y / mapZoom) + (mapY * 0.75);

        //                      double gps_W = 1312-x;
        //                      double gps_N = 1080-y;

        System.out.println(" GPS: " + gps_W + ": " + gps_N);

        gps_N = (gps_N / 8450) + 53.283897;
        gps_W = -6.455690 - (gps_W / 5060);// ;

        double[] returnVals = { gps_N, gps_W };
        return returnVals;
}

int addPointToMap(int x, int y) {

        //                      double gps_W = 1312-(x/mapZoom)+(mapX*0.75);
        //                      double gps_N = 1080-(y/mapZoom)+(mapY*0.75);
        //
        //              System.out.println(" GPS: " + gps_W + ": " + gps_N);
        //
        //
        //      gps_N = (gps_N / 8450) + 53.283897;
        //      gps_W = -6.455690 - (gps_W / 5060);// ;

        double[] gpsPoints = pointToGPS(x, y);

        double gps_N = gpsPoints[0];
        double gps_W = gpsPoints[1];

        System.out.println(" GPS: " + gps_W + ": " + gps_N);
        System.out.println("new Address(0, ADDR ,0, " + gps_N + ", " + gps_W + ");");
        if (deliveries.length<=(addressCount)) {
                int[] deliveriesMore = new int[105];
                for (int i=0; i<deliveries.length; i++) {
```

```
                        deliveriesMore[i] = deliveries[i];
                }
                deliveries = deliveriesMore;
        }

        if (addressCount < 100) {
                addresses[addressCount] = new Address(addressCount, createRandomAddress(), 0, gps_N,
gps_W);
                deliveries[addressCount] = addressCount;
                addressCount++;
                setMatrixEtc();
                sideScrollBar.setValues(sideListScroll, (sideListScroll + 30), 0, addressCount - 1);
                printAddressList();
                refreshMap(addressCount, true);
                g.setColor(Color.black);
                g.fillOval(mouseDown_x - 8, mouseDown_y - 8, 16, 16);
                g.setColor(Color.green);
                g.fillOval(mouseDown_x - 6, mouseDown_y - 6, 12, 12);
                canvas.repaint();

                return addressCount;
        } else {
                sendToNoticeboard("100 Deliveries Maximum", true);
                return -1;

        }

        // x2= (int)((double)((addresses[deliveries[0]].x+mapX_) * mapZoom));
        //               y2= (int)((double)((addresses[deliveries[0]].y+mapY_) * mapZoom));

} // END: addPointToMap()

// ---------------------------------------------------------------------

void pressedUpOrDown(int upOrDown) {
        if (upOrDown == -1)
        if (sideListScroll > 0) {
                sideListScroll--;
                sideScrollBar.setValue(sideListScroll);
                setSideListButton(currentSelectedPoint);
                printAddressList();
        } else {
                if (sideListScroll < (addressCount - 31)) {
                        sideListScroll++;
                        sideScrollBar.setValue(sideListScroll);
                        setSideListButton(currentSelectedPoint);
                        printAddressList();
                }
        }
}

void doCounstructorStuff() {

        distanceMatrix = new Trip[105][105];
        distanceMatrix_unsorted = new Trip[105][105];

        primaryPanel = new JLayeredPane();

        pizzaPlace = new Address(0, "Apache Pizza", 0, 53.38197, -6.59274);
        // 53.381296472014036, -6.5929810570971785);
        deliveriesSorted = false;
        currentClickedPoint = -1;

        screenDimensions = GraphicsEnvironment.getLocalGraphicsEnvironment().getMaximumWindowBounds();
        System.out.println("GE: " + screenDimensions);
        primaryPanel.setBounds(0, 0, screenDimensions.width, screenDimensions.height);
        JPanel superList = new JPanel(new GridLayout(45, 1));

        superList.setBounds(20, 50, 350, screenDimensions.height-70);
        superList.setMaximumSize(new Dimension(350, screenDimensions.height-70));

        insertDataButton = new JButton("Retrieve from Input Box");
        insertDataButton.addActionListener(getDataFromInputBox);
        superList.add(insertDataButton);
```

```
retrieveDataButton = new JButton("Retrieve From Clipboard");
retrieveDataButton.addActionListener(goRetrieveDataFromClipboard);
superList.add(retrieveDataButton);

noticeboard2 = new JLabel("");
superList.add(noticeboard2);


noticeboard = new JLabel("");
superList.add(noticeboard);
statsBoard1 = new JLabel("");
superList.add(statsBoard1);
statsBoard2 = new JLabel("");
superList.add(statsBoard2);
statsBoard3 = new JLabel("");
superList.add(statsBoard3);

sideList = new JButton[32];
// upScrollButton, downScrollButton;

upScrollButton = new BasicArrowButton(BasicArrowButton.NORTH);
superList.add(upScrollButton);
upScrollButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                pressedUpOrDown(-1);
        }
});

for (int k = 0; k < 30; k++) {
        sideList[k] = new JButton();
        sideList[k].putClientProperty("id", "" + k);
        sideList[k].setVisible(false);
        sideList[k].addActionListener(sideListListener);
        superList.add(sideList[k]);
}

downScrollButton = new BasicArrowButton(BasicArrowButton.SOUTH);
superList.add(downScrollButton);
downScrollButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

                pressedUpOrDown(1);

        }
});

noticeboard3 = new JLabel("");
superList.add(noticeboard3);

testButton = new JButton("MST");
testButton.addActionListener(testListener);
testButton.putClientProperty("id", "1");
superList.add(testButton);
testButton2 = new JButton("EULERIAN CIRCUIT");
testButton2.putClientProperty("id", "2");
testButton2.addActionListener(testListener);
superList.add(testButton2);
testButton3 = new JButton("CHRISTOFIDES");
testButton3.putClientProperty("id", "3");
testButton3.addActionListener(testListener);
superList.add(testButton3);
spacerLabel = new JLabel("");
superList.add(spacerLabel);
class MyAdjustmentListener implements AdjustmentListener {
        public void adjustmentValueChanged(AdjustmentEvent e) {
                int newScrollValue = e.getValue();
                System.out.println("Slider's position is " + newScrollValue);

                int lastScroll = sideListScroll;
                sideListScroll = newScrollValue;
                int difference = lastScroll - newScrollValue;
                if (difference != 0) {
                        setSideListButton(currentSelectedPoint);
```

```
                        }

                        printAddressList();
                        // frame.repaint();
                }
        }

        sideScrollBar = new JScrollBar(JScrollBar.VERTICAL, 0, 100, 0, 100);

        sideScrollBar.setBounds(370, 222, 25, 770);
        sideScrollBar.addAdjustmentListener(new MyAdjustmentListener());
        primaryPanel.add(sideScrollBar);

        buildMapFrame();
        canvas.addMouseListener(new MouseAdapter() {
                public void mousePressed(MouseEvent e) {
                        mouseDown_x = e.getX();
                        mouseDown_y = e.getY();
                        System.out.println("mousePressed X: " + mouseDown_x + " \t Y: " + mouseDown_y);
                        if ((e.getModifiersEx() & InputEvent.CTRL_DOWN_MASK) != 0) {
                                System.out.println("ADD POINT TO MAP:");
                                currentClickedPoint = addPointToMap(mouseDown_x, mouseDown_y);

                        } // END: CTRL+MOUSE-CLICK
                        else {
                                int difX, difY, difXY;
                                int minDif = 11;
                                int selected = 0;
                                int ii;
                                for (int i = 0; i < addressCount; i++) {
                                        ii = deliveries[i];
                                        difX = Math.abs(addresses[ii].x - mouseDown_x);
                                        difY = Math.abs(addresses[ii].y - mouseDown_y);

                                        if (difX < 5 && difY < 5) {
                                                difXY = difX + difY;
                                                if (difXY < minDif) {
                                                        minDif = difXY;
                                                        selected = i;
                                                }
                                        }
                                } // for loop
                                if (selected > 0) {
                                        noticeboard.setText("WT:" + addresses[selected].minutesWaiting +
"POINT: " + selected + " "
                                                        + addresses[selected].gpsN + ", " +
addresses[selected].gpsW + " "
                                                        + addresses[selected].postalAddress);
                                        currentSelectedPoint = selected;
                                        if ((currentSelectedPoint - sideListScroll) > 30) {
                                                sideListScroll = currentSelectedPoint - 30;
                                                sideScrollBar.setValue(sideListScroll);
                                        }
                                        if ((currentSelectedPoint - sideListScroll) < 1) {
                                                sideListScroll = 0;
                                                sideScrollBar.setValue(sideListScroll);
                                        }
                                        setSelectedPoint(currentSelectedPoint);
                                        setSideListButton(currentSelectedPoint);

                                        printAddressList();
                                        // SELECTPOINT
                                        currentClickedPoint = currentSelectedPoint;
                                } else {
                                        currentClickedPoint = -1;
                                }
                        } // END: CLICK MAP INITIALLY

                }

                public void mouseReleased(MouseEvent e) {
                        int x = e.getX();
                        int y = e.getY();
```

```
                        //System.out.println("mouseReleased-DIFF X: " + x + " \t Y: " + y);
                        //System.out.println("DRAGGED: " + mapX + " \t Y: " + mapY);
                        currentClickedPoint = -1;
                }
        });
        canvas.addMouseMotionListener(new MouseAdapter() {
                public void mouseDragged(MouseEvent e) {
                        int x = e.getX();
                        int xD = x - mouseDown_x;
                        int y = e.getY();
                        int yD = y - mouseDown_y;
                        System.out.println("DRAG; zoom= " + mapZoom);
                        if (currentClickedPoint > 0) { // DRAG POINT
                                double[] gpsPoints = pointToGPS(x, y);
                                addresses[deliveries[currentClickedPoint]].x = x;
                                addresses[deliveries[currentClickedPoint]].y = y;
                                addresses[deliveries[currentClickedPoint]].gpsN = gpsPoints[0];
                                addresses[deliveries[currentClickedPoint]].gpsW = gpsPoints[1];
                                noticeboard.setText("MOVE: X:" + x + ", Y:" + y + "; " + gpsPoints[0] +
"; " + gpsPoints[1]);
                        } // ZOOM MAP
                        else {
                                if (mapZoom > 1) {
                                        int minX = 0 - (int) ((mapZoom - 1) * 800);
                                        int minY = 0 - (int) ((mapZoom - 1) * 700);
                                        System.out.println("MIN: " + minX + ": " + minY);
                                        mapX += xD;
                                        mapY += yD;
                                        mouseDown_x = x;
                                        mouseDown_y = y;

                                        System.out.println("ZOOM: " + mapZoom);
                                        if (mapX > 0)
                                                mapX = 0;
                                        if (mapY > 0)
                                                mapY = 0;
                                        if (mapX < minX)
                                                mapX = minX;
                                        if (mapY < minY)
                                                mapY = minY;
                                } else {
                                        mapX = 0;
                                        mapY = 0;
                                        mapZoom = 1;
                                }
                        }
                        setMatrixEtc();
                        refreshMap(addressCount, deliveriesSorted);
                        showTotalStats();

                }
        });

        canvas.setBounds(400, 50, screenDimensions.width, screenDimensions.height);

        primaryPanel.add(superList);
        primaryPanel.add(canvas);
        primaryPanel.addMouseWheelListener(wheelMoved);
        listener = new KeyListen();
        this.setFocusable(true);
        this.requestFocus();
        this.addKeyListener(listener);

        JPanel secondaryPanel = buildTopButtons();

        secondaryPanel.setBounds(0, 0, screenDimensions.width, 50);
        primaryPanel.setLayout(null);
        primaryPanel.add(secondaryPanel);
        //              inputBox.setPreferredSize(new Dimension(50,1000));
        inputBox = new JTextArea("", 1, 1);
        // primaryPanel.add(inputBox,BorderLayout.NORTH);

        getContentPane().setLayout(new BorderLayout());
```

```
        getContentPane().add(primaryPanel, BorderLayout.CENTER);

        setVisible(false);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setAlwaysOnTop(true);
        pack();
        setExtendedState(JFrame.MAXIMIZED_BOTH);

        String sampleData = "1,38 Parsons Hall Maynooth ,4,53.37521,-6.6103\r\n"
                        + "2,34 Silken Vale Maynooth ,6,53.37626,-6.59308\r\n"
                        + "3,156 Glendale Leixlip ,18,53.37077,-6.48279\r\n"
                        + "4,33 The Paddocks Oldtown Mill Celbridge ,8,53.3473,-6.55057\r\n"
                        + "5,902 Lady Castle K Club Straffan ,11,53.31159,-6.60538\r\n"
                        + "6,9 The Park Louisa Valley Leixlip ,3,53.36115,-6.48907\r\n"
                        + "7,509 Riverforest Leixlip ,10,53.37402,-6.49363\r\n"
                        + "8,16 Priory Chase St.Raphaels Manor Celbridge ,7,53.33886,-6.55468\r\n"
                        + "9,13 Abbey Park Court Clane,13,53.2908,-6.67746\r\n"
                        + "10,117 Royal Meadows Kilcock ,12,53.39459,-6.66995\r\n"
                        + "11,7 Riverlawn Abbeyfarm Celbridge ,3,53.33239,-6.55163\r\n"
                        + "12,10 Fair Green Court Kilccock ,7,53.39847,-6.66787\r\n"
                        + "13,11 The Lodge Abbeylands Clane,12,53.29128,-6.67836\r\n"
                        + "14,628 Riverforest Leixlip ,5,53.37416,-6.49731\r\n"
                        + "15,12 Castlevillage Avenue Celbridge ,8,53.35298,-6.54921\r\n"
                        + "16,116 Connaught Street Kilcock ,4,53.39839,-6.66767\r\n"
                        + "17,44 Rinawade Avenue Leixlip ,20,53.36141,-6.51834\r\n"
                        + "18,35 Beech Park Wood Beech Park Leixlip ,14,53.36287,-6.52468\r\n"
                        + "19,96 Priory Lodge St. Raphael's Manor Celbridge ,2,53.33835,-6.53984\r\n"
                        + "20,33 Leinster Wood Carton Demesne Maynooth ,7,53.39351,-6.5542\r\n"
                        + "21,6 Glen Easton View Leixlip ,15,53.36883,-6.51468\r\n"
                        + "22,40 Oaklawn West. Leixlip ,8,53.36833,-6.50589\r\n"
                        + "23,169 Glendale Leixlip ,24,53.37043,-6.48193\r\n"
                        + "24,14 The Rise Louisa Valley Leixlip ,15,53.36115,-6.48907\r\n"
                        + "25,28 The Lawn Moyglare Abbey Maynooth ,7,53.38895,-6.60579\r\n"
                        + "26,43 The Woodlands Castletown Celbridge ,12,53.34678,-6.53415\r\n"
                        + "27,14 Rye River Crescent Dun Carraig Leixlip ,8,53.36518,-6.48913\r\n"
                        + "28,32 The View St.Wolstan Abbey Celbridge ,10,53.33751,-6.53173\r\n"
                        + "29,20 Habourview The Glenroyal Centre Maynooth ,9,53.37954,-6.58793\r\n"
                        + "30,416A Ballyoulster Celbridge ,5,53.34133,-6.51856\r\n"
                        + "31,10 Brookfield Avenue Maynooth ,8,53.36976,-6.59828\r\n"
                        + "32,15 Willow Rise Primrose Gate Celbridge ,19,53.33591,-6.53566\r\n"
                        + "33,3 Lyreen Park Maynooth ,26,53.38579,-6.58673\r\n"
                        + "34,2 Beaufield Drive Maynooth ,10,53.37414,-6.60028\r\n"
                        + "35,28 The Avenue Castletown Celbridge ,18,53.34514,-6.53615\r\n"
                        + "36,4 Abbey Park Grove Clane ,14,53.29206,-6.67685\r\n"
                        + "37,78 Crodaun Forest Park Celbridge ,15,53.35401,-6.54603\r\n"
                        + "38,1 Kyldar House Manor Mills Maynooth ,29,53.38122,-6.59226\r\n"
                        + "39,1002 Avondale Leixlip ,22,53.36869,-6.48314\r\n"
                        + "40,18 College Green Maynooth ,5,53.37247,-6.60044";
        inputBox.setText(sampleData);

        String clipBoardData = retrieveClipboard();
        System.out.println(clipBoardData);

        if (processDataIn(clipBoardData, "Data found in clipboard") < 1) {
                System.out.println("NO DATA IN CLIPBOARD - inserting sample data");
                processDataIn(sampleData, "Sample Data loaded");
        }

}

// ######################################################################

int processDataIn(String dataIn) {
        return processDataIn(dataIn, "");
}

int processDataIn(String dataIn, String description) {

        String[] clipBoardRows = dataIn.split("[\\r\\n]+");
        System.out.println(clipBoardRows.length);

        addresses[0] = pizzaPlace;
```

```
                deliveriesSorted = false;
                int rowCount = 0;
                int n = 0;
                int errorCount = 0;
                int minutesWaiting = 0;
                double GPS_n = 0;
                double GPS_w = 0;
                for (int i = 0; i < clipBoardRows.length; i++) {
                        String currentCol[] = clipBoardRows[i].split(",");
                        System.out.println(currentCol.length);
                        if (currentCol.length == 5) {
                                errorCount = 0;
                                try {
                                        n = Integer.parseInt(currentCol[0]);
                                        currentCol[2] = currentCol[2].replaceAll("\\s+", "");
                                        currentCol[3] = currentCol[3].replaceAll("\\s+", "");
                                        currentCol[4] = currentCol[4].replaceAll("\\s+", "");
                                        minutesWaiting = Integer.parseInt(currentCol[2]);
                                        GPS_n = Double.parseDouble(currentCol[3]);
                                        GPS_w = Double.parseDouble(currentCol[4]);
                                } catch (NumberFormatException nfe) {
                                        errorCount++;
                                        sendToNoticeboard("NUMBER PARSE ERROR", true);
                                        System.out.println("'" + currentCol[2] + "' ; '" + currentCol[3] + "' ;
'" + currentCol[4]);
                                }

                                String currentAddress = currentCol[1];

                                if (errorCount == 0) {
                                        System.out.println(rowCount);
                                        rowCount++;
                                        addresses[rowCount] = new Address(n, currentAddress, minutesWaiting,
GPS_n, GPS_w);
                                }

                        } else {
                                System.out.println("STRING: " + currentCol.length + " : " + currentCol);
                        }

                }

                System.out.println("Rows: " + rowCount);

                // System.out.println("1. " + addresses[1].number); // + " ; " +
                // addresses[i].postalAddress + " ; "
                //                      + addresses[i].minutesWaiting + " ; " + addresses[i].gpsN + " ; " +
addresses[i].gpsW );
                if (rowCount > 0) {
                        addressCount = rowCount + 1;
                        sendToNoticeboard(description, true);
                        System.out.println("Data found in clipboard");
                        addresses[0] = pizzaPlace;
                        for (int i = 0; i < rowCount; i++) {
                                System.out.println(" I = " + i + " " + addresses[i]);
                                System.out.println(i + ". " + addresses[i].number + " ; " +
addresses[i].postalAddress + " ; "
                                                + addresses[i].minutesWaiting + " ; " + addresses[i].gpsN + " ;
" + addresses[i].gpsW);
                        }
                        pasteInTextField.setText(rowCount + " Addresses Found");
                        setMatrixEtc();
                        deliveries = new int[105];
                        for (int ii = 0; ii < addressCount; ii++) {
                                deliveries[ii] = ii;
                        }
                        refreshMap(addressCount, false);
                        sideListScroll = 0;

                        sideScrollBar.setValues(0, (sideListScroll + 30), 0, addressCount - 1);
                        printAddressList();
                } else {
                        pasteInTextField
```

```
                        .setText("Paste Data in here or click ADD FROM CLIPBOARD to take data from your
clipboard.");
                }

                setVisible(true);
                return rowCount;

        } // END: processDataIn()

        final MouseWheelListener wheelMoved = new MouseWheelListener() {

                @Override
                public void mouseWheelMoved(MouseWheelEvent e) {
                        int wheelMove = e.getWheelRotation();
                        if ((e.getModifiersEx() & InputEvent.CTRL_DOWN_MASK) != 0) {
                                System.out.println("WHEEL MOVED - ZOOM!");


                                if (wheelMove < 0) {
                                        zoom(wheelMove);

                                } else {
                                        zoom(wheelMove);
                                }
                        } else {
                                // wheel moved
                                e.getComponent().getParent().dispatchEvent(e);

                        }
                }

        };

        public void zoom(int wheelMove) {
                double wheelFraction = wheelMove;
                wheelFraction /= 2;
                mapZoom += wheelFraction;
                if (mapZoom < 1) {
                        mapZoom = 1;
                        mapX = 0;
                        mapY = 0;
                }
                System.out.println(mapZoom + " __ " + wheelFraction);
                refreshMap(addressCount, deliveriesSorted);
        }

        // ########################################################################

        public void printAddressList() {
                String st;
                System.out.println("SIDELIST" + sideListScroll);
                if (sideListScroll < 0)
                        sideListScroll = 0;
                for (int k = 0; k < 30; k++) {
                        if ((k + (sideListScroll + 1) < addressCount)) {
                                st = "#" + addresses[deliveries[k + (sideListScroll) + 1]].number + " "
                                                + addresses[deliveries[k + (sideListScroll) + 1]].postalAddress;
                                sideList[k].setText(st);
                                sideList[k].setVisible(true);
                        } else {
                                st = "";
                                sideList[k].setVisible(false);
                        }
                }
                boolean showScrolls;
                if (addressCount > 30) {
                        showScrolls = true;
                } else {
                        showScrolls = false;
                }
                // sideScrollBar.setValues(sideListScroll, (sideListScroll+30), 0,
                // addressCount-1);
                upScrollButton.setEnabled(showScrolls);
                downScrollButton.setEnabled(showScrolls);
```

```
		sideScrollBar.setEnabled(showScrolls);

	}

	public FastDeliver_window() {

		super("THE HOT PIZZA COMPANY");
		fileChooser = new JFileChooser();
		fileChooser.setMultiSelectionEnabled(false);
		FileNameExtensionFilter filter = new FileNameExtensionFilter("CSV files (*csv)", "csv");
		fileChooser.setFileFilter(filter);
		fileChooser.addChoosableFileFilter(new FileNameExtensionFilter("txt files (*.txt)", "txt"));

		img = null;
		try {
			img = ImageIO.read(new File("map.png"));  // MAP FILE
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		doCounstructorStuff();
		// refreshMap();
	}

	private static String loadDATA(String filename) {

		File aFile = new File(filename);
		StringBuffer contents = new StringBuffer();
		BufferedReader input = null;

		try {
			input = new BufferedReader(new FileReader(aFile));
			String line = null;

			while ((line = input.readLine()) != null) {
				contents.append(line);

				contents.append(System.getProperty("line.separator"));
			}
		} catch (FileNotFoundException ex) {
			System.out.println(
					"Can't find the file '" + filename + "' - are you sure the file is in
this location: " + filename);

			return null;
		} catch (IOException ex) {
			System.out.println("Input output exception while processing file");
			ex.printStackTrace();
		} finally {
			try {
				if (input != null) {
					input.close();
				}
			} catch (IOException ex) {
				System.out.println("Input output exception while processing file");
				ex.printStackTrace();
			}
		}
		return contents.toString();
	}

	public void refreshGrid() {
		getContentPane().removeAll();

		doCounstructorStuff();
	}

	void go_Save(String filename) {
		String thisAddress;

		int len = filename.length();

		// add .csv if necessary:
```

```
                if (len < 1 || addressCount < 1)
                        return;

                if (len < 3) {
                        filename += ".csv";
                } else {
                        boolean dotFound = false;
                        for (int i = 0; i < len; i++) {
                                if (filename.charAt(i) == '.') {
                                        dotFound = true;
                                        break;
                                }
                        }
                        if (!dotFound)
                                filename += ".csv";
                }

                try {
                        FileWriter myWriter = new FileWriter(filename);
                        for (int i = 1; i < addressCount; i++) {
                                thisAddress = addresses[deliveries[i]].number + ", " +
addresses[deliveries[i]].postalAddress + ", "
                                                + (int) addresses[deliveries[i]].minutesWaiting + ", " +
addresses[deliveries[i]].gpsN + ", "
                                                + addresses[deliveries[i]].gpsW;
                                myWriter.write(thisAddress + "\r\n");
                        }
                        myWriter.close();
                        this.setTitle("HOT PIZZA COMPANY - " + filename);
                        System.out.println("Saved");
                        sendToNoticeboard(filename + " SAVED.");
                } catch (IOException error) {
                        sendToNoticeboard("File Write Error.");
                        System.out.println("File Write Error.");
                        error.printStackTrace();
                }
        }

        void go_Load(String filename, boolean show) {

                String fileLoadedData = loadDATA(filename);

                this.setTitle("HOT PIZZA COMPANY - " + filename);

                if (processDataIn(fileLoadedData, "Data loaded from file.") < 1) {
                        noticeboard.setText("NO VALID DATA IN FILE");
                }

        } // END: go_Load

}

public class TSproblem {

        public static FastDeliver_window gridWindow;

        public static void deliverPizza() {

                gridWindow = new FastDeliver_window();
        }

        public static void main(String[] args) {

                deliverPizza();
        }

}
```