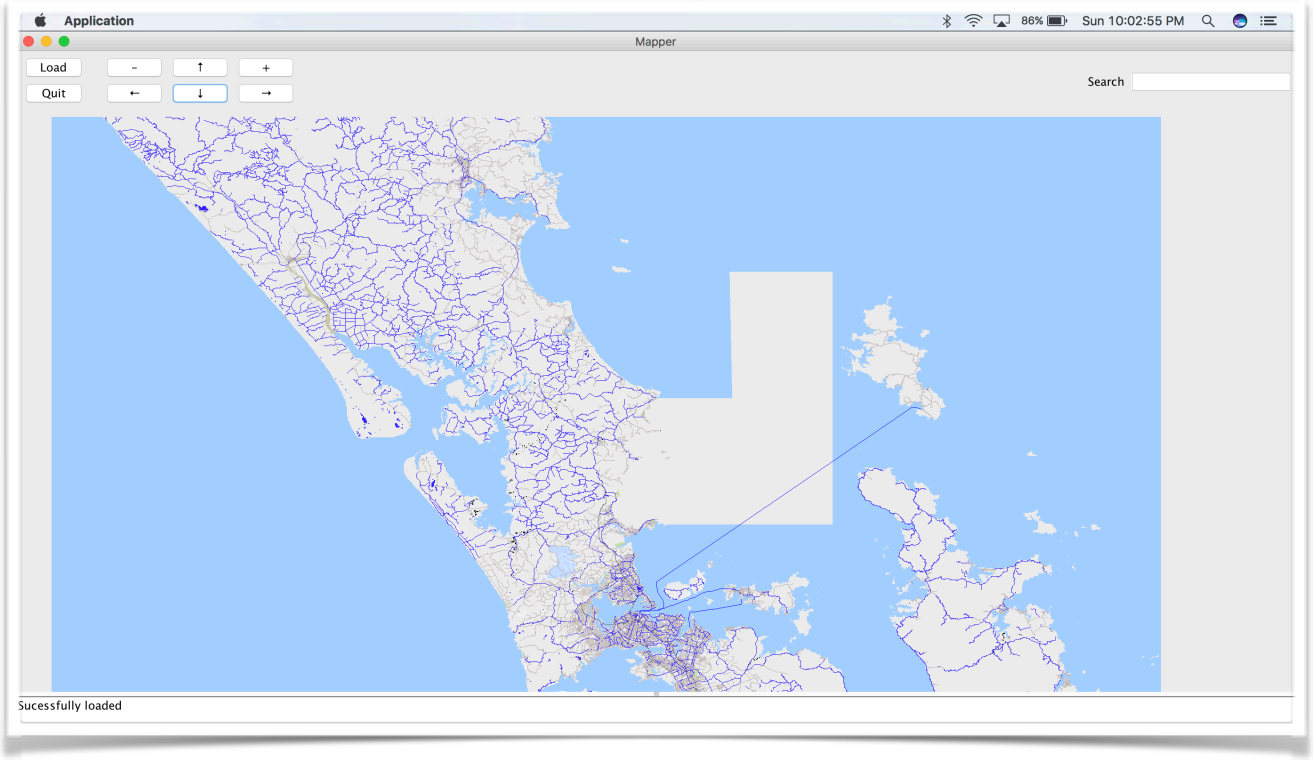
Auckland Road Map

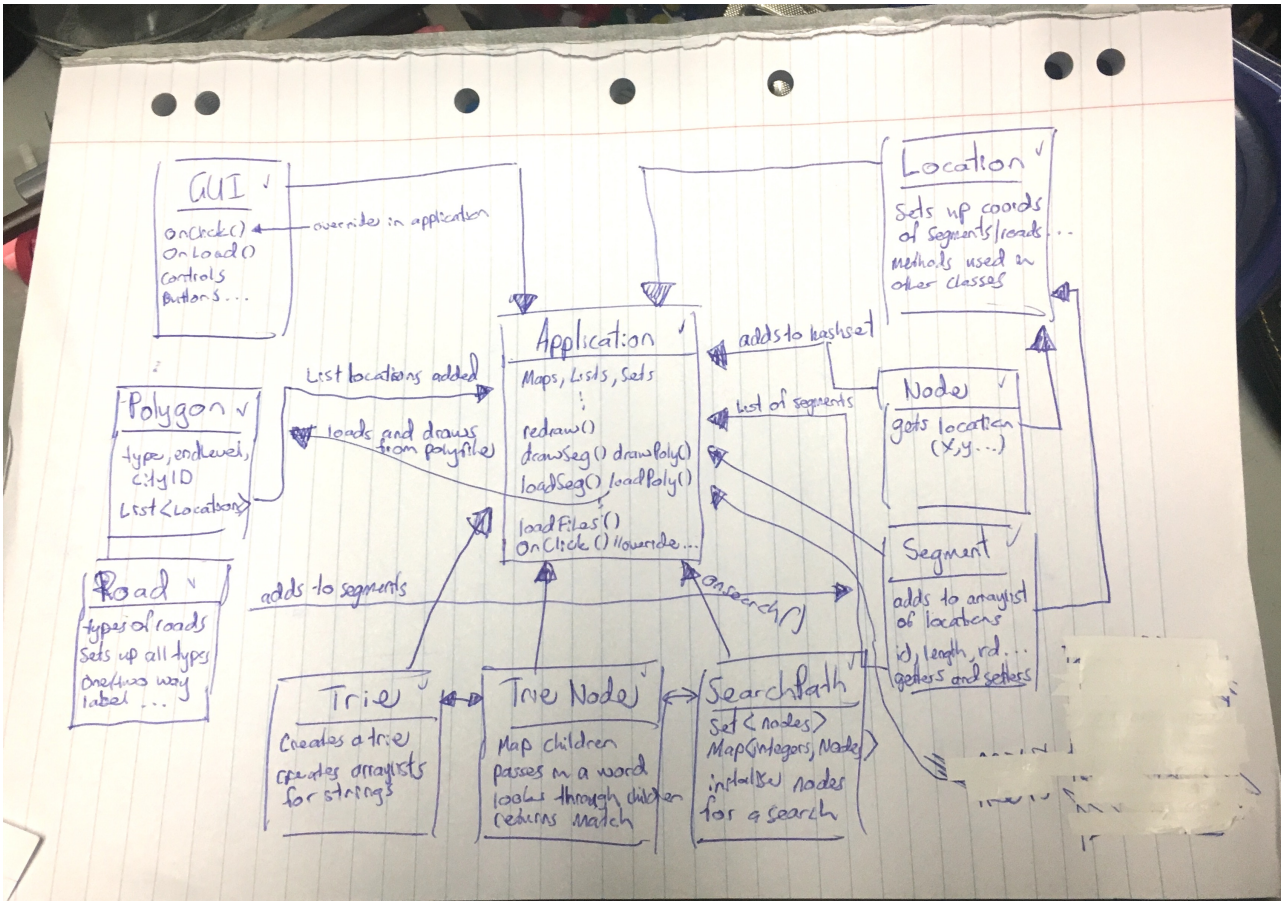
Assignment 1 Report

Daniel Walker - 26 March 2017



Plan showing how i would create classes required and a brief explanation of what everything needed to do.

It was essential i could visually see how the program was going to work. At first the diagram only had a few classes, and i added to it before i progressed to the next step.



APPLICATION CLASS

The application class is the main class where everything comes together to be executed. it consists of several methods, ranging from implementing the GUI, loading all the files such as setting up buffer readers to sort through the files for such parts as the polygons to load all the water/reserves which make the map look much nicer. It is also responsible for drawing the segments/polygons using the Graphics(2D) abstract classes. It has @override methods from the GUI such as onClick(), onLoad() which tell the program what to do when such an event happens.

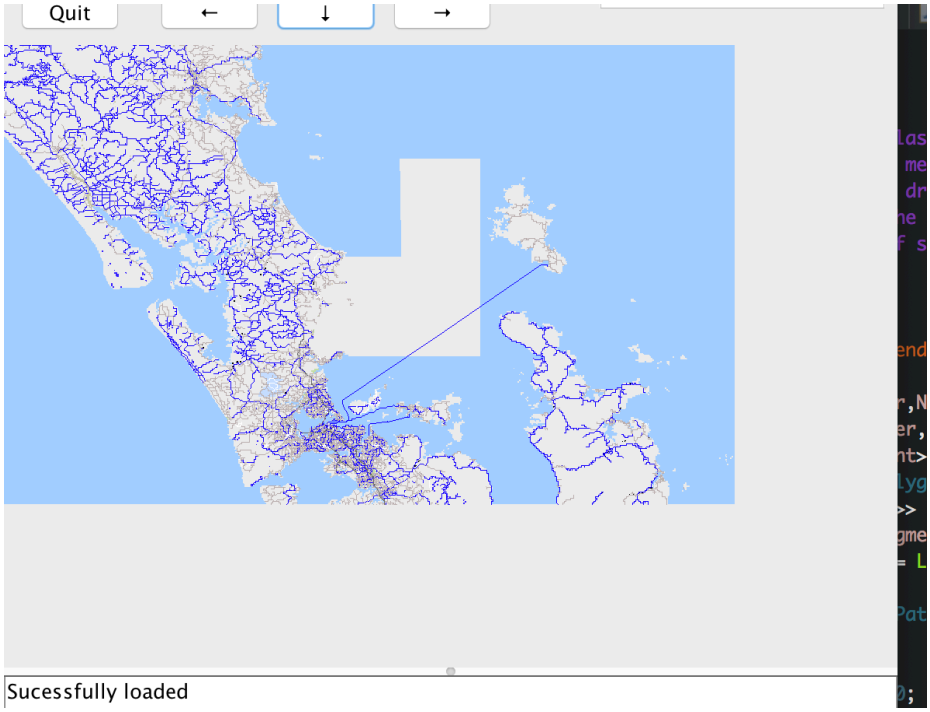
```
18
19 /**
20  * This is a small example class to demonstrate extending the GUI class and
21  * implementing the abstract methods. Instead of doing anything maps-related, it
22  * draws some squares to the drawing area which are removed when clicked. Some
23  * information is given in the text area, and pressing any of the navigation
24  * buttons makes a new set of squares.
25  *
26  * @author Daniel Walker
27  */
28 public class Application extends GUI {
29
30     public static Map<Integer,Node> nodes = new HashMap<Integer,Node>();
31     private static Map<Integer,Road> roads = new HashMap<Integer,Road>();
32     public static List<Segment> segments = new ArrayList<Segment>();
33     private List<Polygon> polygons = new ArrayList<Polygon>();
34     private List<Set<Segment>> selectedSeg = new ArrayList<Set<Segment>>(); // roads to highlight from search
35     public static HashSet<Segment> selectedPath = new HashSet<Segment>(); // roads to highlight for path finding
36     private Location origin = Location.newFromLatLon(-36.847622 , 174.763444); // Center of Auckland location
37     private Trie trie;
38     public searchPath searchPath;
39     public Node transitOne;
40     public Node transitTwo;
41     private double scale = 20; //zoom function
42
```

The above screenshot shows several fields creating a range of collections to which methods call on the data inside the collections. Such as drawing the segments require data inside the segments array list which has the info required for drawing in a particular location on the map.

What my program does and doesn't do

- Reads all data files required -> roadID, roadSegments, nodeId, polygons
    - Map structure used for roads and nodes
    - Array list for segments and polygons
- Displays road data using segments

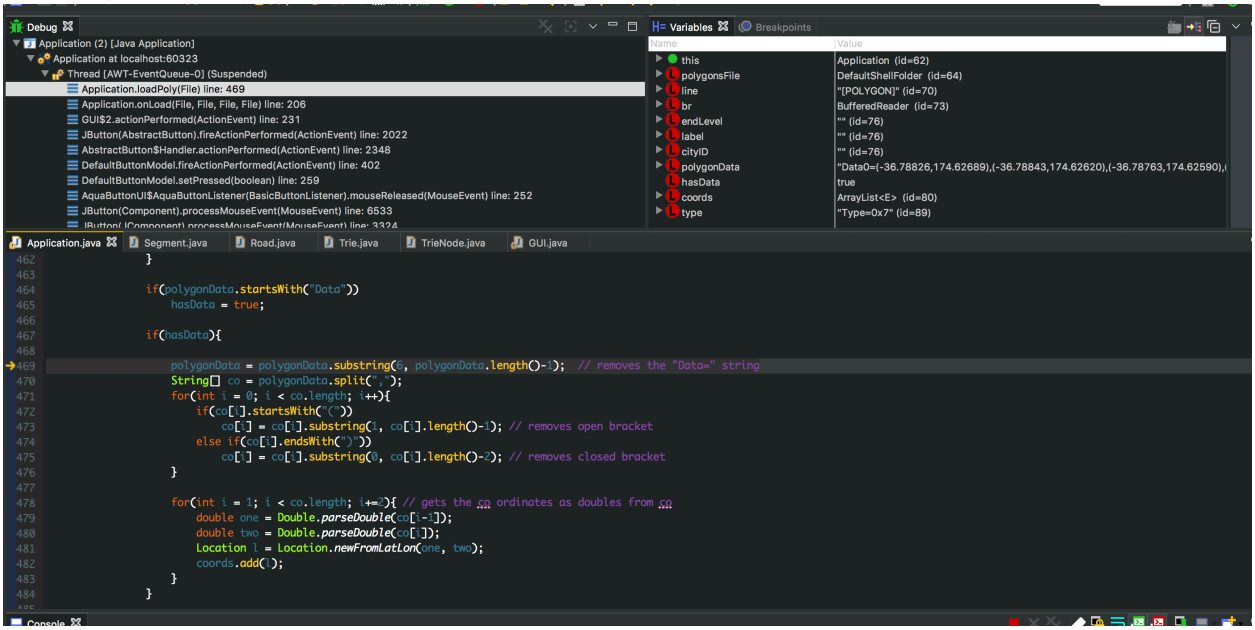
- Displays polygon data to some extent. Doesn't work 100% as there are a few errors that didn't load correctly/missing data. The large file takes quite a while to load, but this is the result when it loads:
- There is some missing polygon data as shown. Most likely because i didn't account for every type of polygon. I needed to come up with a better solution that would automatically account for all types of polygons as there was too many to add manually.
- The zoom in/out function works correctly, but uses the top left corner as the anchor so zoom in doesn't go through centre. Panning N,S,E,W also works. This can be adjusted using the scale which will alter how much the map moves with every click.
- Searching for a road using a Trie structure works well. I did start to work on a dropdown box, with a limit of 10 options, but did not have time to finish this. You need to type the road exactly right, and the searched road will highlight in red with a thicker stroke.
- Using the mouse to click on an intersection will display the nearest road names at that intersection. It will not highlight the roads as i didn't have time to implement a drawOnclick() method.



Important Data Structures Used

1. Trie Structure
    - Two classes were created to set up a trie structure “Trie.java” and “TrieNode.java”. These would be used to sort the road data to enable a fast search on the names of roads.
    - The Trie class creates the structure and initialises the head and nodes. Also sets up an array to store returned strings.
    - Trie node class does most of the work, assigning characters to nodes. Steps down the trie checking for the next characters in the children nodes. Adds returned strings to an array list etc etc, returns the word if its the search word on the screen. This can be linked to a draw method to display the searched word with a red stroke.

2. Buffer Reader
    - This was used to sort through all the data and store it all in an appropriate format so i could access what ever part i needed.
    - Buffer Reader was a good option as it is an efficient way to scan through a lot of data.
    - I was able to implement the segments/nodes/roads with a little planning and drawing out a plan of how to scan though it all.
    - I found it very difficult read though the polygon data - this took me the longest to implement. I struggled to deal with how the different parts of the data were separated. The debugger was very useful here as i was easily able to see what values were being assigned to what variables.



How i tested my program:
I didn't do a lot of testing on my program. The smaller data folder made it easy to quickly load in a small part of the map so i could run it many times to see how my program was going.