

City

Map

Editor

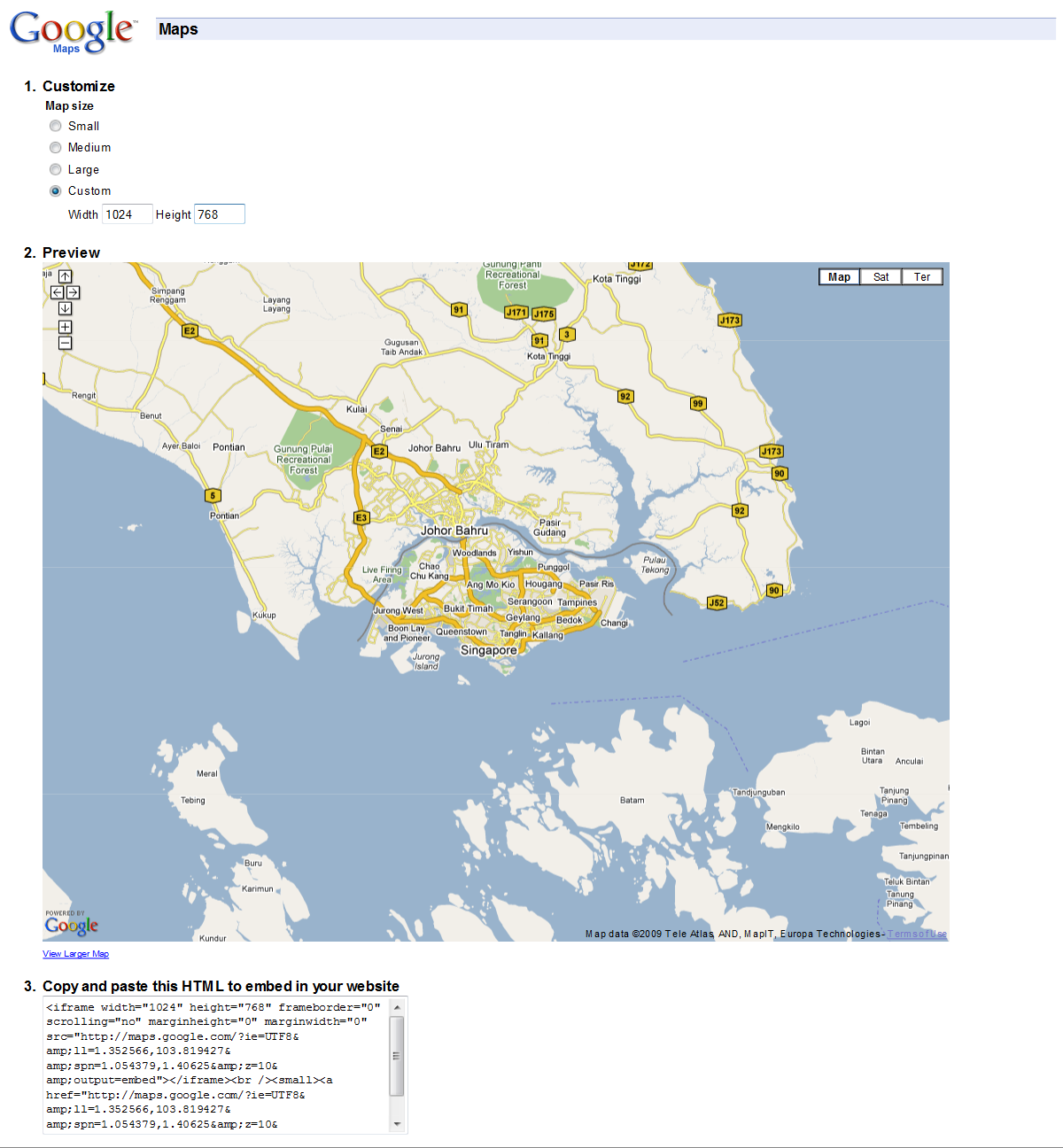
Algorithm Design Description

**Chapter 1**

Getting Maps

Initial Methodology

Initially the maps were actually saved screenshots of the web browser running the page on Google custom embedded maps. They are then cropped using programs like paint and Adobe Photoshop to obtain map images.



Methodology Review

However, we feel that this process should be automated and we should not let the user go through such a tedious process to obtain a map image. We then researched on the feasibility of loading the Google maps in our application.

After reviewing several methodologies to achieve our objective, we realize the fastest and easiest way would be to embed a browser in our application and load a html page which contains the Google map application customize code.

Embedded browser in Java Application

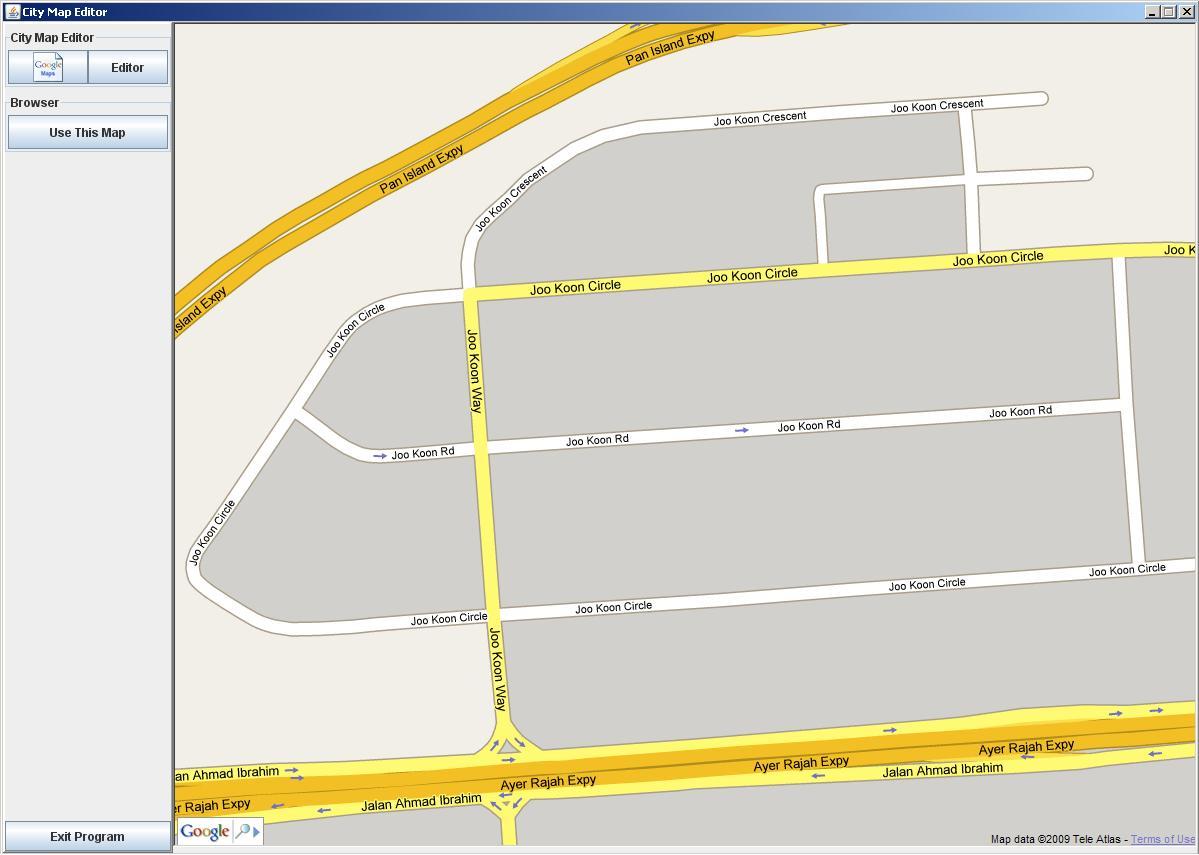
https://jdic.dev.java.net/

In order to run a web browser within a Java application, we search online and found an API called JDIC that allows us to run the web browser application inside java. It uses the settings and the browser engine installed in your computer and run an instance of the browser in your application.



Modifications

As we do not need to show any interface at all, we modified the library codes and customized it. Thus showing only the fixed view of the Google map html page we created.



Google map servlet

As Google mapplet application in web browser is fairly common, it is not hard to find a guide to do so.

Guide: <http://www.searchengineknowledge.com/hosting/embedgooglemaps.php>

<http://today.java.net/pub/a/today/2006/10/25/integrating-google-maps-into-web-application.html>

By providing the correct parameters using the Google Maps API, we customized the map for our purposes.

API: <http://mapki.com/wiki/Google_Map_Parameters>

<http://code.google.com/apis/maps/documentation/>

Settings include:

* Setting map display size (Resolution)
* Disable zoom
* Add mini search bar
* Remove navigation button

Saving Maps

By using Java API called Robot, we saved a screen shot of only the map. This map images will then be used with the rest of our program and the rest of the sub-systems.

API: <http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Robot.html>

**Chapter 2**

Map Validation

Finding roads

A Google map uses several colors for roads. The different color denotes the type of road it is, example highway, main roads, etc. These colors are not used for buildings or anywhere else. We use this to our advantage.

Color filtration

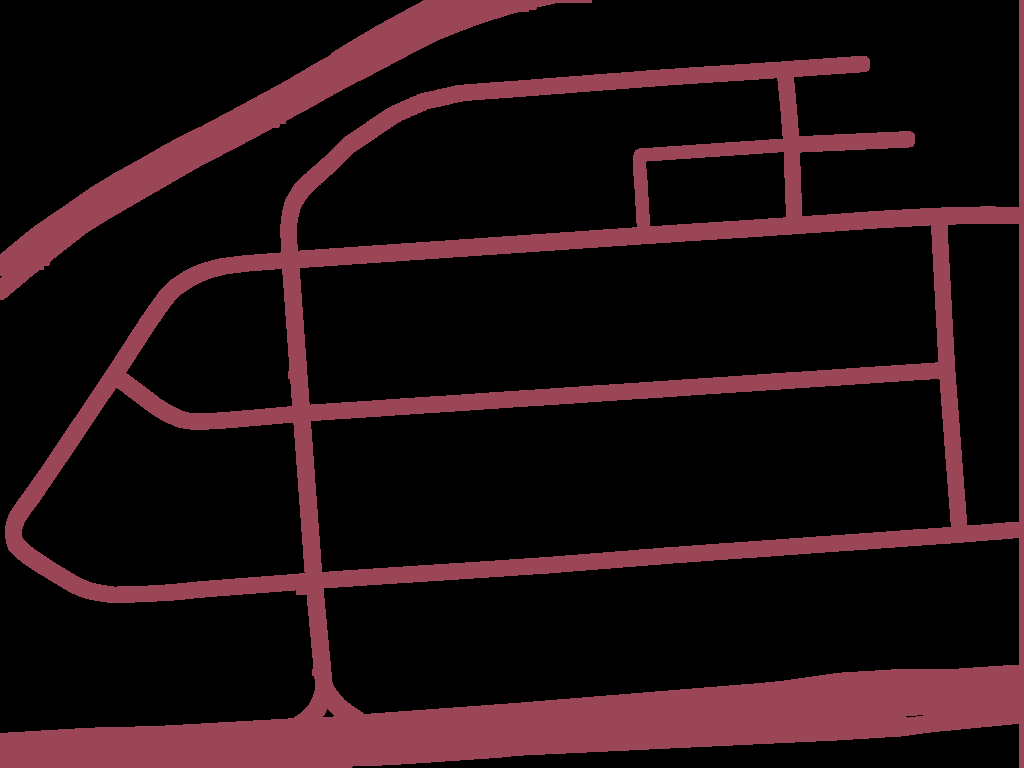
Raster scan: http://en.wikipedia.org/wiki/Raster\_scan

By finding out the colors of the road, we used raster scanning method on the map image saved. Repainting the image with only 2 colors (black for non roads, pink for roads) we are able to differentiate which part of the map are roads and which aren’t.

Original Map:



Filtered Road Version:



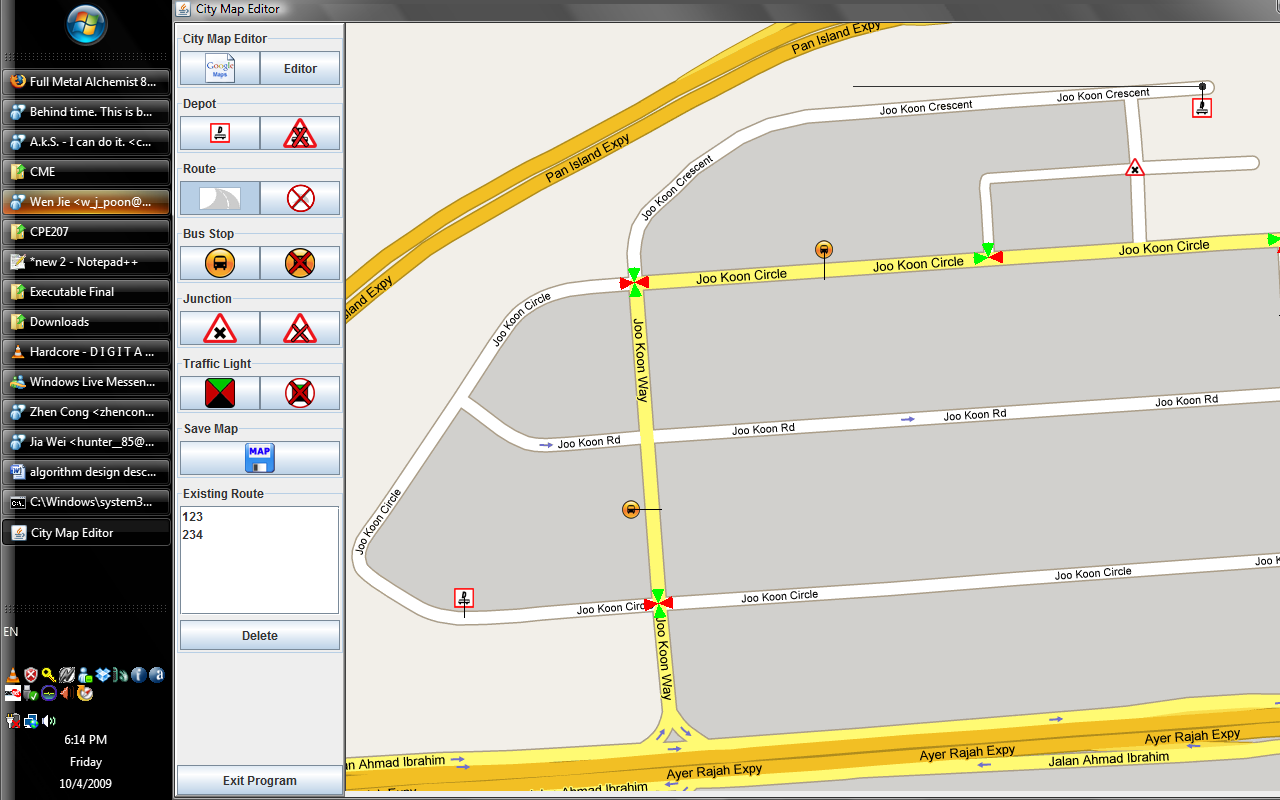
The problem with this method, however, is that it includes the text as part of the road (many roads has names written on it). Because of this text written on the maps are also considered roads and thus is not very accurate as sometimes the names are across the road or even outside the road. Despite this minor issue, the validation works perfectly for our purposes.

**Chapter 3**

Drawing Routes

Methodology

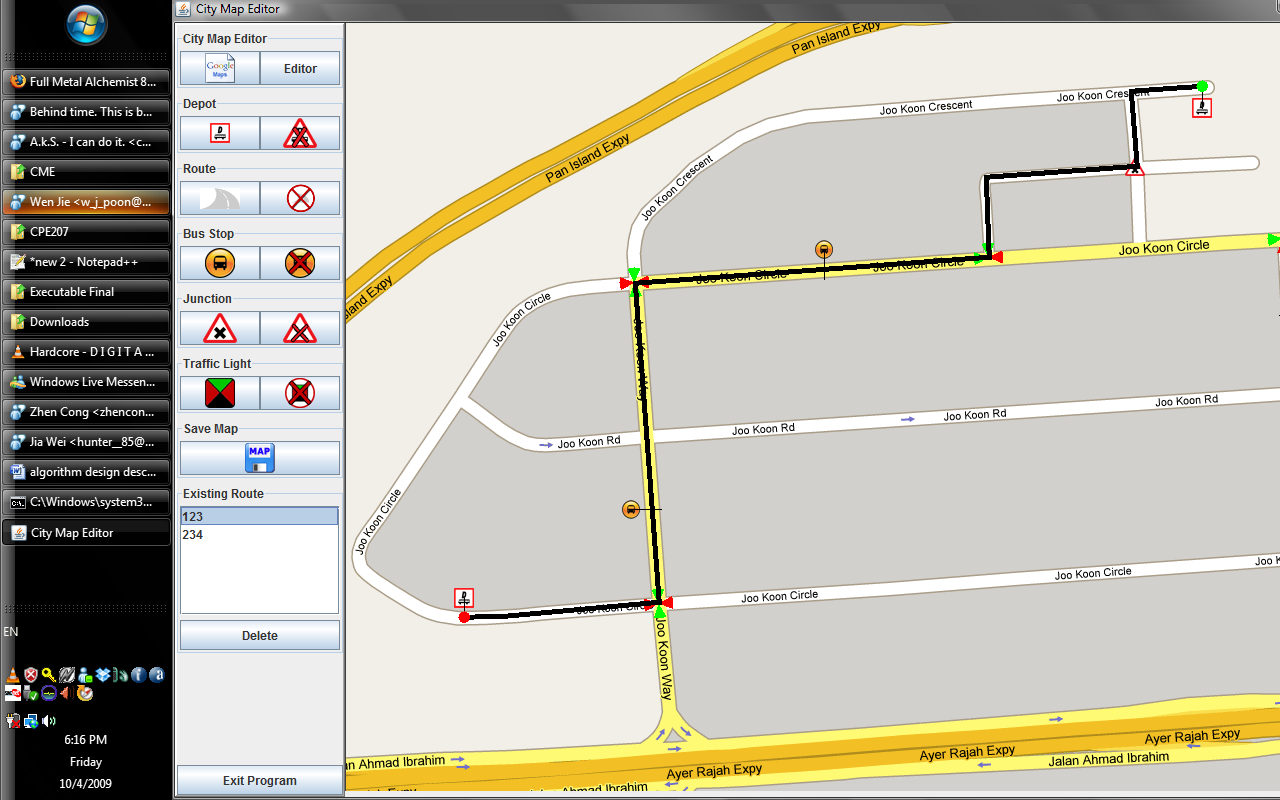
The methodology of drawing the bus routes involving clicking on the map. To simplify the calculation involved, we draw routes based on 2 points. The straight line between the 2 points is the route the bus will travel on. Using this method, we will reduce the confusion when there is more than 1 way to travel between the 2 points. However it will require the using to do multiple clicks when the road is a curve.



Validation of routes

By using the color filtered map, we check if the line between the 2 points is on the road. If it is, the points are accepted and become part of the route when saved. Else, the user will not be able to proceed until the point which the user clicks in correct.

When the user clicks and confirms the end point, the route is highlighted to indicate the confirmation of the route.



**Chapter 4**

Validation of Traffic Lights and Junction

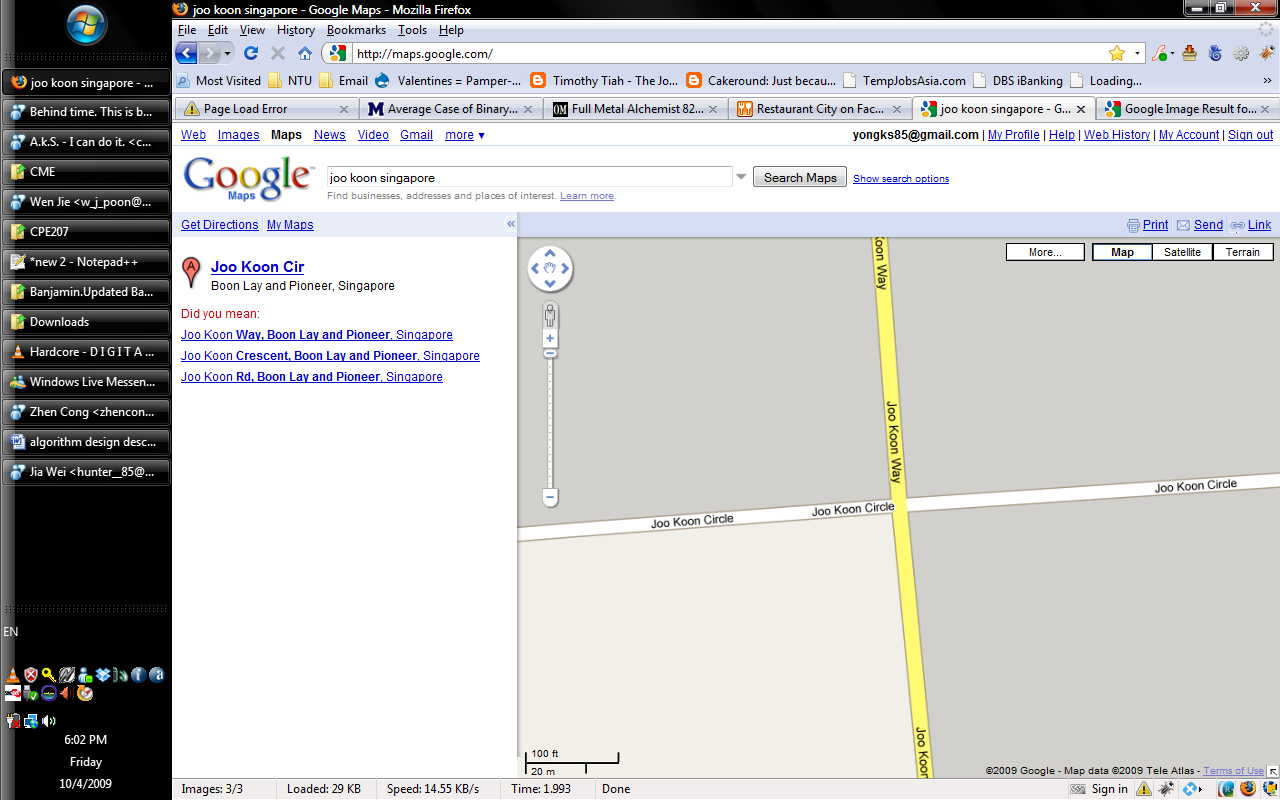
Introduction

In this program we assume the traffic lights to be placed only at junctions. This is to reduce complexity involved for the whole system. A lot of brain storming has been done on how to effectively validate the position on which user clicks for the placement of traffic light as well as how to automatically detect what type of junction it is (3-way, 4-way, etc).

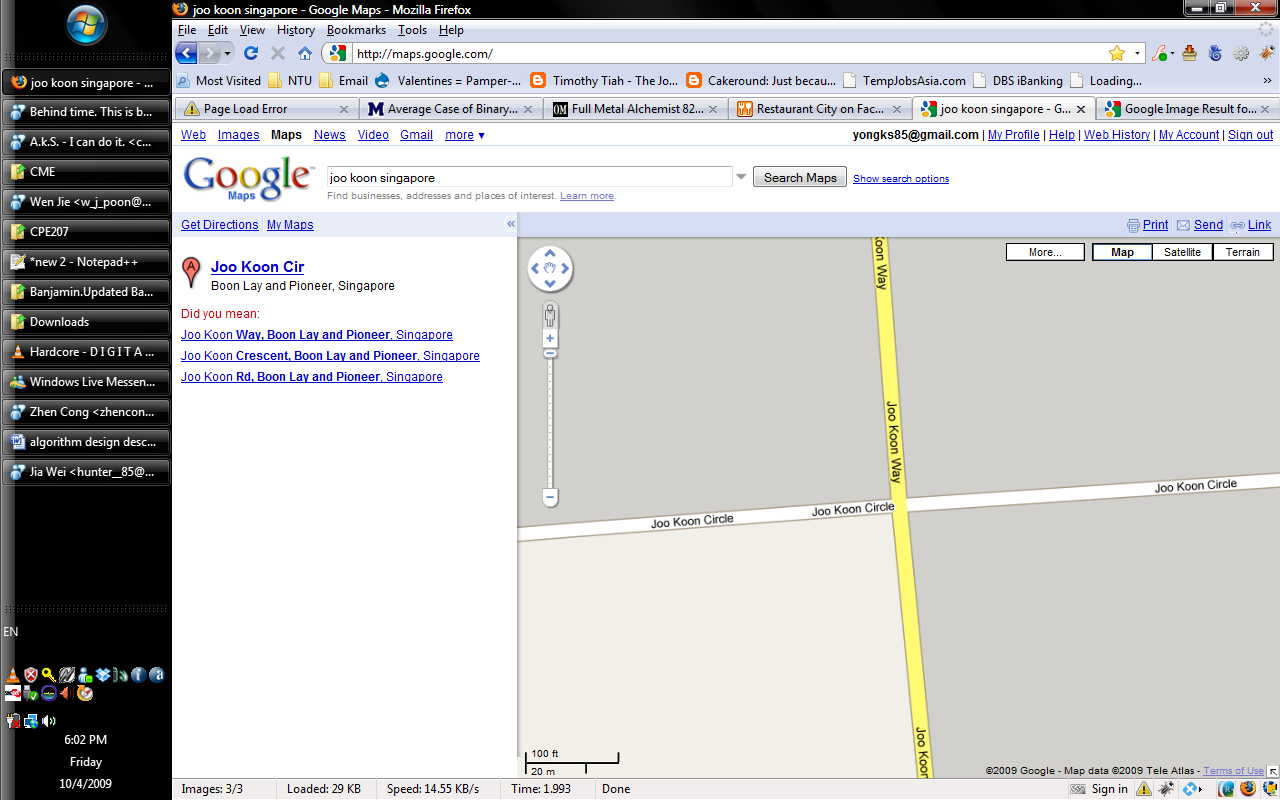
Validation

The picture below shows a 4 way junction. The black cross is where the user clicks to place a traffic light. Upon clicking, a hidden square is drawn, and points where the roads and non-roads meet are marked (blue crosses).

By counting the number of points (blue crosses) we can tell whether if this is a junction or not as well as what type of junction this is.



The middle points that are on the road are then computed (green crosses, image below). Depending on the type of junction, different ways of drawing lines needs to be done. The image below shows how hidden lines are drawn for a 4 way junction. The centre point which is where the junction is located is the intersection of the 2 lines.



Conclusion

The above method is very good as it allows automated detection of junctions based on a point. It also allows detection of the different angles of the road at the junction as well as identifying how many ways the junction is.

However, this method is not fool proof. We have tested with about 80-85% success rate. It is dependent on the road width as well as the square width for validation and as there are many different widths of road on the earth, the results may differ. As we are using a fixed square width, roads that are thicker than the square at the junction will not be validated correctly and produce poor results.

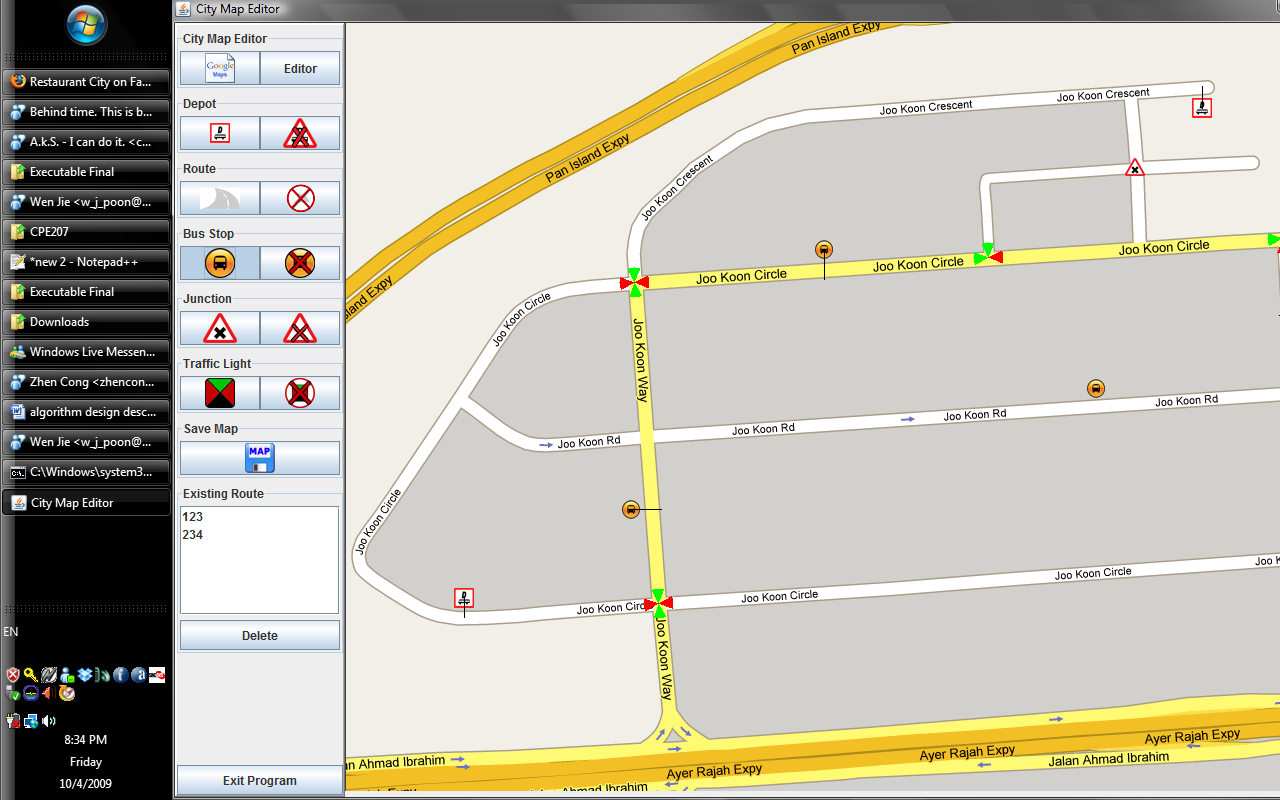
**Chapter 5**

Validation of Bus stop and Depots

Introduction

The bus stops and Depots are an important part in this program. The depots are the points where buses are “created” as well as “destroyed”. The bus stops are where the buses stop to pick up passengers. Both the bus stops and depots uses a similar method of validation and generate a similar set of data.

Methodology

The user can only click on the side of the road for placing bus stops and depots. The point where user clicks would be the point where the actual bus stop and depots are created.

The user needs to click near the road but not on the road itself. The program validates if the user clicks too far or too near to the road. This is to allow space for creating the actual image for bus stops and depots.

In the image below shows an example. The blue cross shows where the user clicks. Similar to the traffic light validation, a hidden square is drawn. The programs then scan and try to find the nearest point where the road is.

The program then finds the point where the road ends based on that line. For depots, it returns the middle of the road instead.

