159.333

Project Implementation

Traffic Simulation

Brad Heap

04231708

Semester Two 2007

Abstract

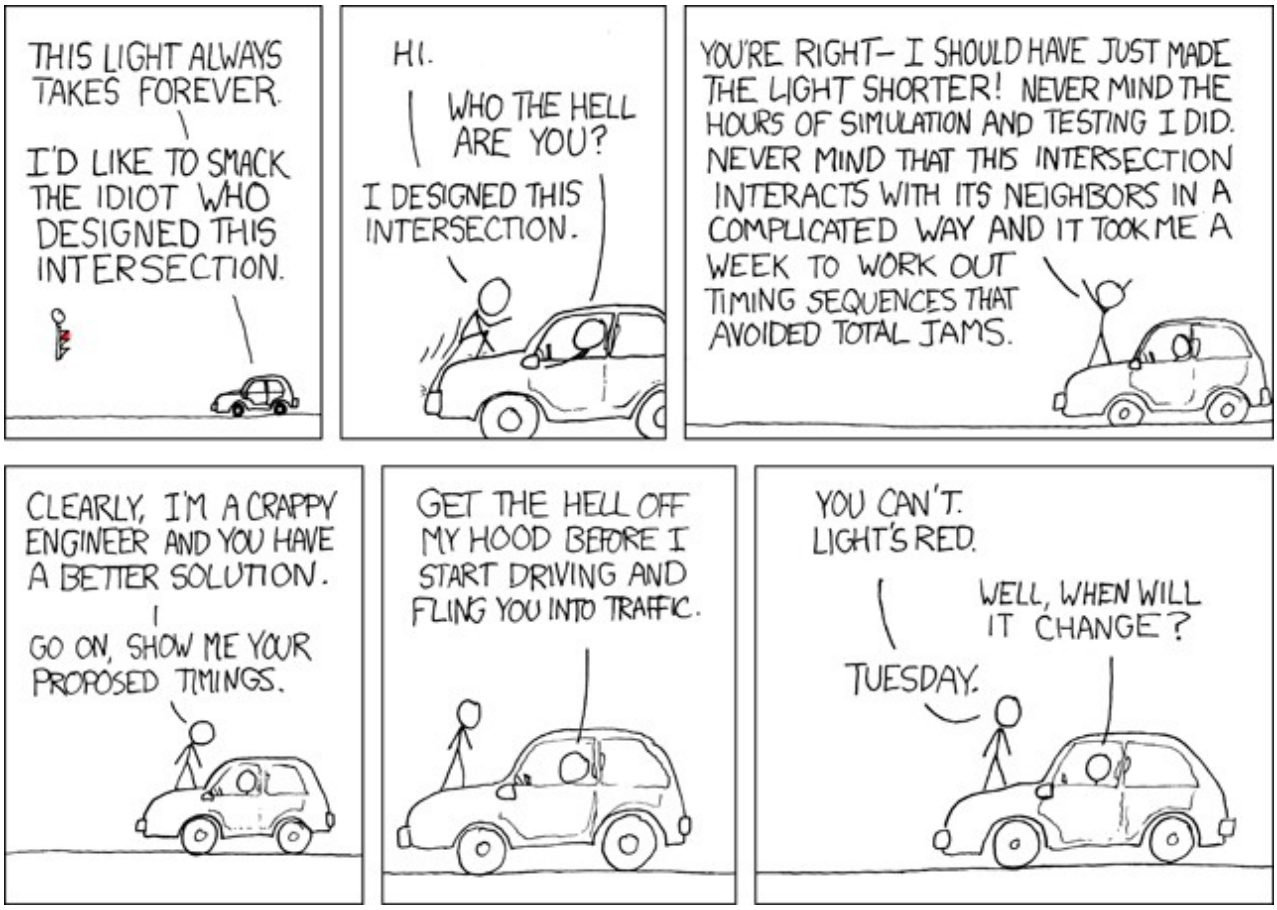
The aim of this thesis is to outline and describe mine and Dean Jerkovich's project into Traffic Simulation.

Traffic is one of those things that everyone hates, and yet no one can manage to avoid. The idea for this project came from a number of different areas. The first area was Auckland's traffic congestion. The second was computer simulation and modeling the real world on computers. The third area was getting experience in working in a team on a large scale project, as due to time constraints none of the assignments from taught papers allow for this.

This thesis will outline our initial background research into traffic and current traffic simulators, look at the planning and design of our traffic simulator, review the construction of the traffic simulator, and suggest improvements for future systems.

At the end of this project and thesis we aim to have developed key experience in working as a team in constructing a large scale software product, as well as general experience programming in Java.

(http://img.dailymail.co.uk/i/pix/2007/01/china_468x312.jpg)



Cartoon 1: Long Light (http://xkcd.com/277/)

Table of Contents

Abstract....2
Acknowledgments....8
Background Research....9
Auckland Traffic....9
Current Traffic Simulators....13
Microsimulation of road traffic....13
Traffic Simulation....14
Multi-Agent Traffic Scenario....15
A Traffic Simulator Applet....16
Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism....17
Cellular Automation Traffic Simulators....18
Project Planning and Initial Design....19
Initial Project Outline....19
Development of Outline Into Brief....24
Problem to Solve....24
Base Specifications....24
Further Specifications....25
Options for Expansion....25
Initial Design....26
Initial Program Design....26
Class Design....27
Prototyping....36
Road Network Data Structures In Java....40
GUI Design and Implementation....41
GUI Design and Layout....41
Road Network Design Pane....44
Simulator Control Pane....45
User Interaction with GUI....45
Common to both....45

Road Network Design Pane....46
Simulator Control Pane 48
Error detection and checking....51
Menu Bar Design....52
File Menu....52
Edit Menu....52
Insert Menu....53
View Menu....53
Tools Menu....54
Help Menu....54
Simulator Design and Implementation....55
Initial Simulation Loop....55
Issues with initial design....55
Improved Simulation Loop....59
Car Controller Design and Implementation....61
Car Class Design....61
Car Controller V1: Initial Implementation....62
Car Controller V2: Artificial Intelligence....63
Car Controller V3: Collision Detection and Awareness 63
Car Controller V4: Polished Version....63
Limitations of Artificial Intelligence System....64
Intersection Controller and Implementation....65
Background....65
Initial Implementation....65
Simple Give Way....65
Traffic Lights....66
Summary, Conclusions and Recommendations....67
Major issues faced/Missing and Incomplete Features....67
Object Orientated Programming Approach 67
Documentation....67
Testing....67
Comparison of final version to requirements laid out in brief....68

Recommendations for a future system....69
Appendices....70
Car Controller Code V2....71
Car Controller Code V3....74
Car Controller Code V4....76
References....81
Brief Glossary....81

Illustration Index

Traffic Chaos in China....2
Long Light (cartoon)....3
Microsimulation of road traffic....14
Traffic Simulation....15
Multi-Agent Traffic Scenario....16
A Traffic Simulator Applet....17
Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism....18
Cellular Automation Traffic Simulators....19
Original Road Network Class Diagram....28
Class Diagram With Car Class....29
Class Diagram With Intersection Extension....33
Final Design Class Diagram....34
Road Network Data Structure Diagram....39
159.235 Assignment One GUI....40
Traffic Simulator Main Window GUI....41
159.235 Assignment Two Viewer Control Pane....42
Manage Roads GUI....42
Intersection Properties Dialog....44
Road Properties Dialog....45
Road Information Dialog....45
Car Class Core Structure....59

Acknowledgments

Firstly to my project partner Dean Jerkovich thanks for always having those wild ideas and ways to look at the project from a different angle and perspective. And for constantly saying “Refactor!”

Ken Hawick thanks for supervising us, and providing those ideas and that vision when we couldn't see through the darkness.

Chris Scogings and Peter Kay for listening to our original idea in late 2006.

The other Massey University Computer Science Staff for helping us out when we needed help and advice especially with debugging specific and often unusual/bizarre issues.

Background Research

Auckland Traffic

Auckland has the worse traffic congestion in New Zealand. Naturally being the largest city in New Zealand this is to be expected. However, the severity of traffic congestion in Auckland is caused by a lot more factors then just its size. One of the key issues impacting Auckland's traffic congestion is the design of the Auckland Road Network. Auckland is a city whose road network was never thoroughly designed and instead grew quite erratically over time as the city expanded and demand grew.

The backbone of Auckland's Road Network is Auckland Motorway Network, which consists of the Southern, Northern, North-Western and South-Eastern motorways. The design or lack of design, of these motorways forms some of the key issues with traffic congestion in Auckland. One of the most important and unfortunate contributing factors to the congestion on the Auckland motorway system is the fact that it was never completed to its original plans, and what we have now is a network that is a lot smaller then the original intent. Because of this the current Auckland road network consists of a small series of motorway links and many smaller arterial roads. Within the motorway system in particular there are three key areas that severely affect traffic congestion in Auckland.

1. The Auckland Harbour Bridge

   The Auckland Harbour Bridge is the only direct road link between Auckland City and North Shore City. Because of this all traffic that needs to travel between these two cities has to travel through this bottleneck the Auckland Harbour Bridge has become a major factor in traffic congestion in Auckland.

   Traffic Congestion on the Auckland Harbour Bridge has become such an issue that a recent study has shown that many people are delaying their trips around peak periods and instead crossing the bridge when it is less busy. However, in the last 15 years the total traffic crossing the bridge daily has increased by 38%[1].

There have been many suggested plans for a second harbour crossing but a number of factors have prevented this from going ahead. The first factor is the cost of building a second harbour crossing. Many councils have what is called 90 day vision, and with the cost of a long term project such as the construction of a second harbour crossing being so high anything serious on the matter never really gets past a few comments to the media.

A more important factor in my view on the issue of a second harbour crossing is a new bridge causing more traffic congestion then the current single bridge. The issue here is if they construct a new crossing to a different area of the North Shore say Devonport or Birkenhead they could face issues with it as the surrounding local road network is not designed to cope with the sudden increase in traffic.

2. Spaghetti Junction (or the Central Motorway Junction)

   Spaghetti Junction in the middle of central Auckland is another area of high traffic congestion. This is caused by the merger of the Northern, Southern, and North-Western motorways.

   Spaghetti Junction is one of those necessary evils in road network design. Without it traffic would be even worse as cars would need to traverse suburban roads to connect between the different motorways. However, because of all the connections in Spaghetti Junction the traffic congestion contained in it is a result of its own success, with many cars needing to merge in many different places in a small amount of space.

3. South-Eastern motorway terminal at Mt Roskill

   The biggest issue with the end point of the South-Eastern motorway is that it ends in the middle of a residential area. Work is being slowly made to improve this, however until it is complete there will continue to be traffic problems.

Not all of Auckland's traffic issues can be blamed on poor motorway design. There are a number of suburban areas that have issues with traffic congestion. A few case examples of these areas are:

- Downtown Auckland

  Downtown Auckland suffers severe traffic congestion for a number of reasons. The first reason is the high density of commercial and residential areas in the central city and this results in a large number of people being in the area. Downtown Auckland is also one of the oldest areas of Auckland and the road network was not designed for the use that it receives today. There are a lot of narrow roads and one way lanes.

- Greenlane

  Greenlane is one of the central suburban hubs of Auckland. The Greenlane intersection and motorway onramp system in particular suffers from severe traffic congestion. This is purely a result of the number of key arterial roads that meet at this key point. Work is currently underway to improve the roads and intersections in this area.

- Takapuna

  Takapuna suffers traffic congestion because it is the CBD of the North Shore, connects the Devonport peninsula to the rest of Auckland, is away from the motorway, and has two major roads arterial roads connect into it (Hurstmere Road, and Taharoto Road)

- Onewa Road

  Onewa Road suffers from severe traffic congestion in the morning heading towards the motorway and the opposite at night heading away from the motorway, this is because it is the only major road serving a large number of residents on the North Shore.

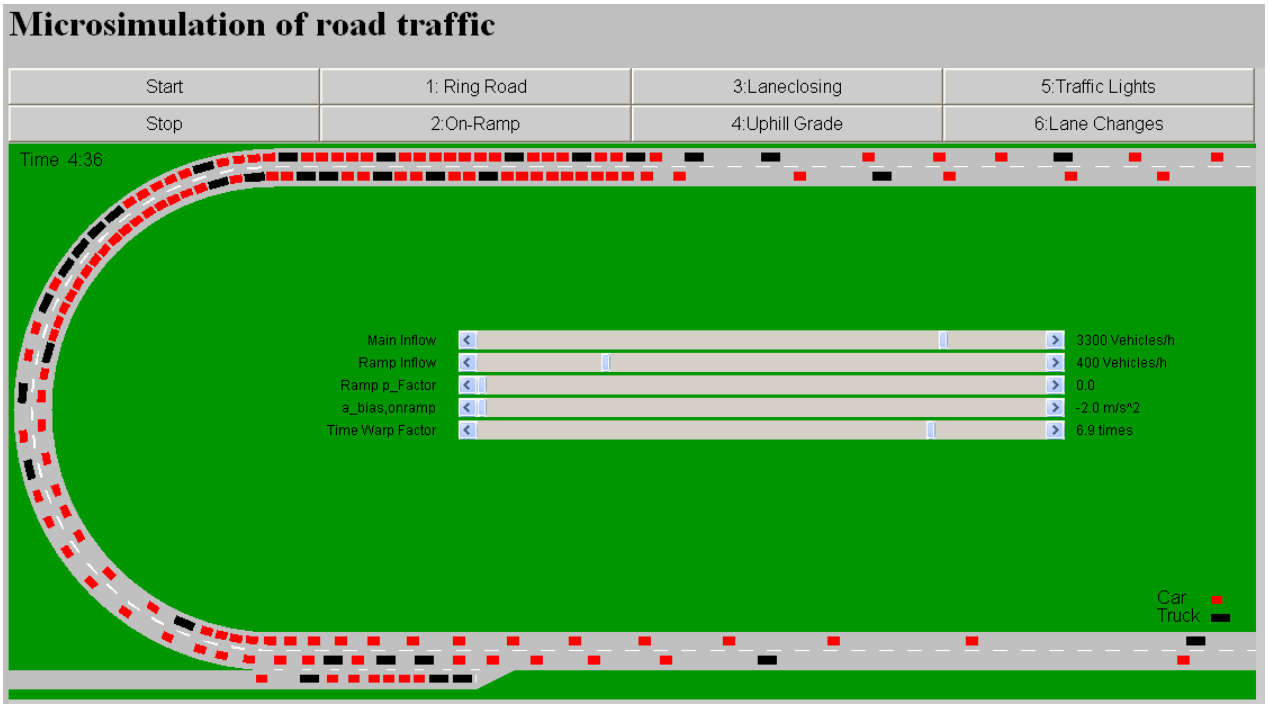
There is one common theme runs amongst all of the above examples, a central focal point of multiple roads that in most cases were not designed for the amount of traffic that they no carry. How traffic is affected by these sorts of roads is one key area that we wish to investigate in our project.

Current Traffic Simulators

In deciding to do a project on traffic simulation we first looked at currently available traffic simulators. One of the key issues that we found with them was they all only analysed either a single or only a few intersections and not a complex network of interconnected roads and intersections and this is what drove us to develop a broader traffic simulator.

Microsimulation of road traffic

http://www.traffic-simulation.de/

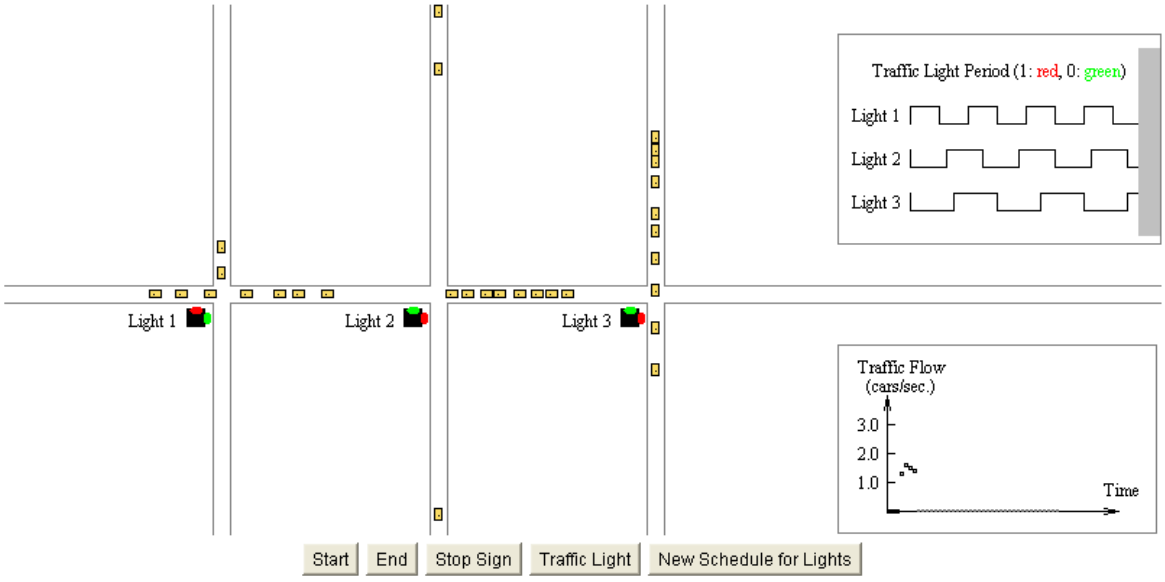


This traffic simulator models six different traffic situations: a simple ring road, a motorway on-ramp, the effect of closing a lane on a multi-laned road, the effect of climbing a hill, the effect of traffic lights, and the effect of lane changes.

This is one of the best traffic simulators that we found. However as already noted in the general comments it only deals with a simple single situation road network and does not look at the effect of traffic from a broader perspective. The ability to dynamically change road and driver factors make this simulation a interesting tool to play with but without any real data output all it is pixels moving on the screen.

Traffic Simulation

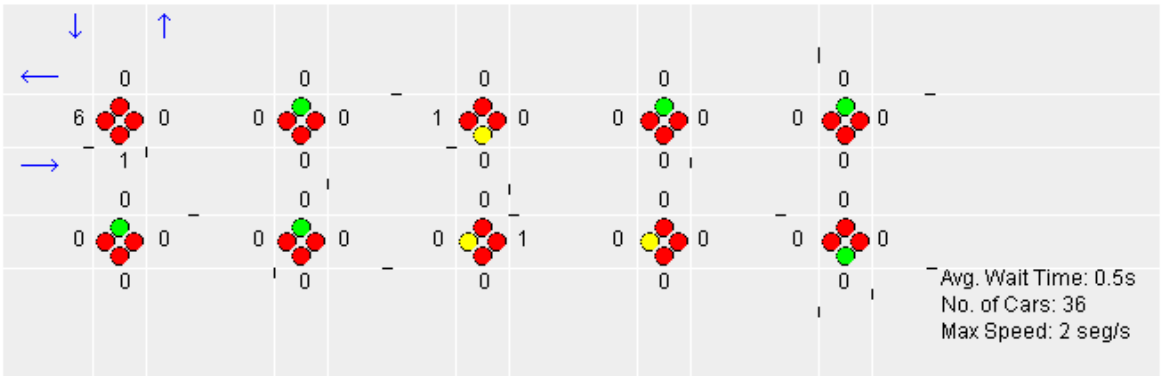
http://www.phy.ntnu.edu.tw/oldjava/Others/trafficSimulation/applet.html



This traffic simulator analyses three interconnected intersections all controlled either traffic lights with a simple square pulse model light controller or a stop sign. Like the Microsimulation of road traffic this simulator looks nice on the screen but gives the user no control over the traffic conditions and does not give any real data as output.

Multi-Agent Traffic Scenario

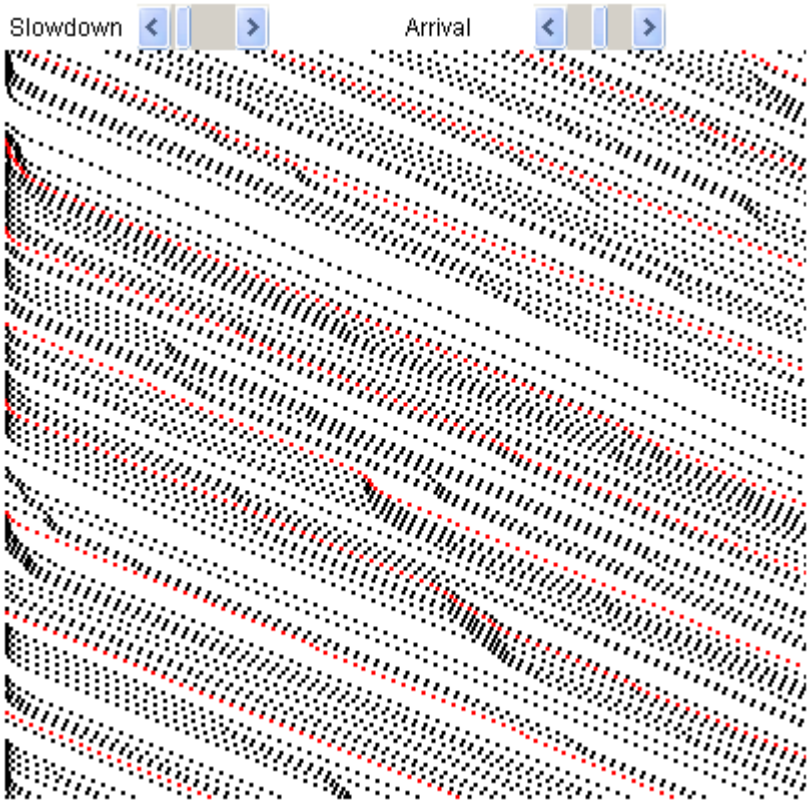
http://www.ece.umd.edu/~adityak/trafficagent.html



This simulator analyses nine interconnected intersections and is the largest in terms of scale of the simulators that we looked at. Like most of the other simulators this simulator does not allow the user to edit the parameters of the simulator and only provides very basic output data.

A Traffic Simulator Applet

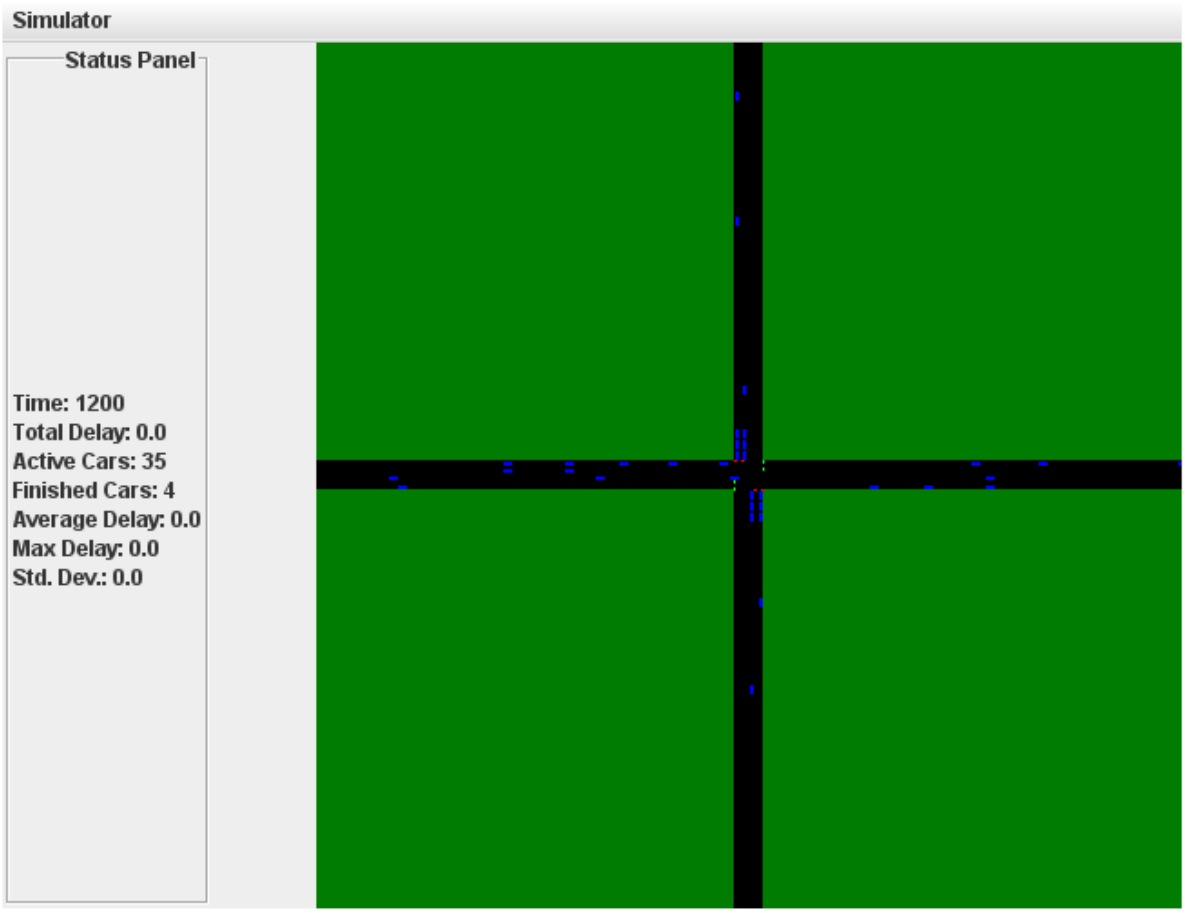
http://horstmann.com/applets/RoadApplet/RoadApplet.html



It takes a little bit of time and reading to understand how this traffic simulator works. The key thing that it demonstrates is traffic flows on a freeway and how traffic is slowed down by the behavior of drivers. As well as being hard to understand, like the other simulators there is no hard data outputted from the simulator. As a result of this, despite its best intentions this simulator was one of the worst they we researched.

Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism

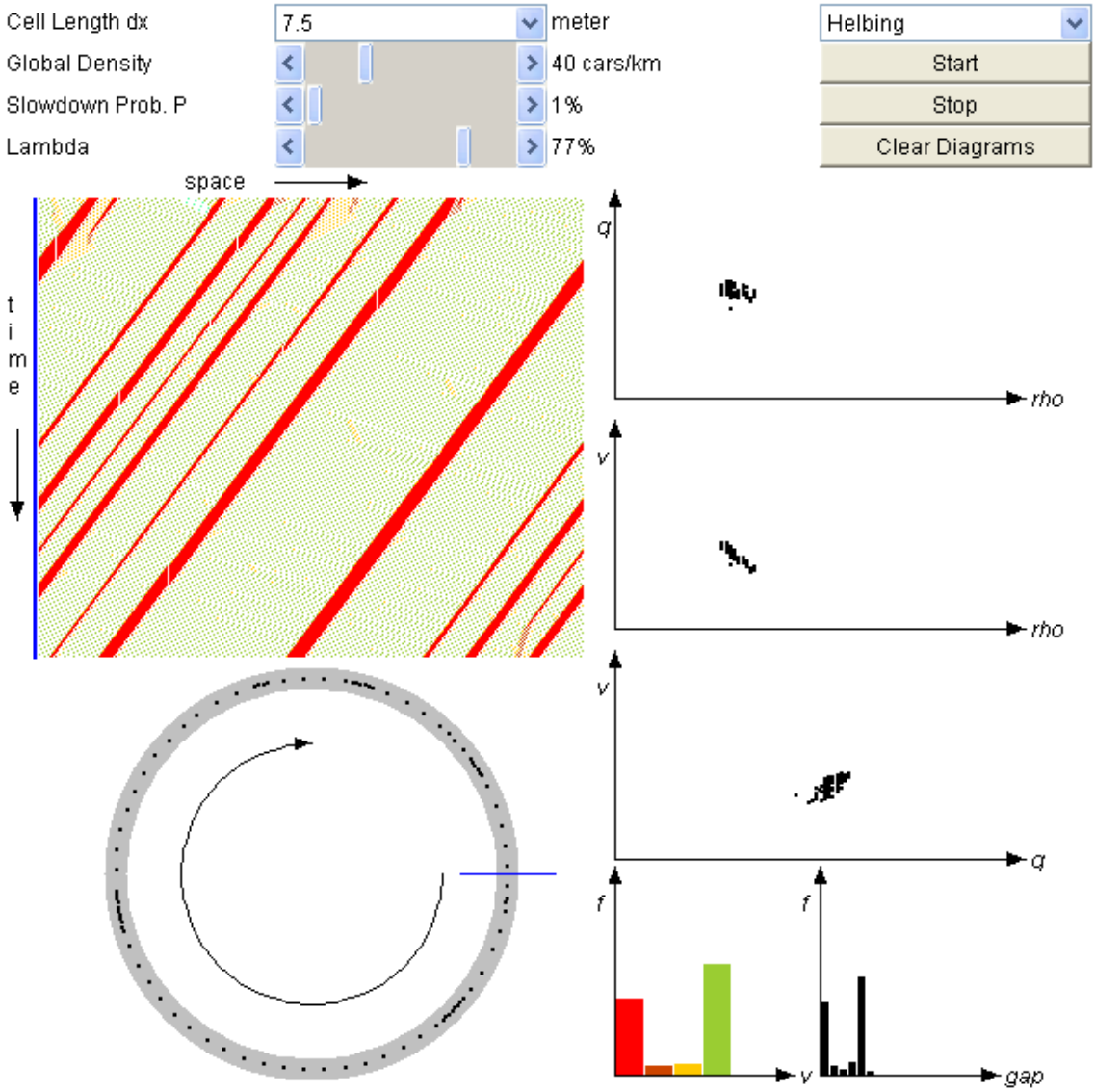
http://www.cs.utexas.edu/users/kdresner/2004aamas/index.html



This simulator is one of the best that we investigated. It simulates three different approaches to designing an intersection: traffic lights, a pseudo give way system, and an overpass system. The simulator is customisable and produces some basic output however it still only simulates one intersection which limits the use of this simulator.

Cellular Automation Traffic Simulators

http://rcswww.urz.tu-dresden.de/~helbing/RoadApplet/



This simulator uses the ideas of Cellular Automata to simulate traffic. This simulator is one of the more difficult to understand and while it allows for customised parameters it does not produce any solid data as output.

Project Planning and Initial Design

Initial Project Outline

Our initial project outline consisted of three areas: foundation requirements, graphical interface requirements, and expansion ideas. Foundation Requirements listed a number of what we considered key requirements for the project. I will briefly highlight our reasoning for each.

- Road network and intersections implemented using graphs

  A graph is:

  > *"In computer science, a graph is a kind of data structure, specifically an abstract data type (ADT), that consists of a set of nodes and a set of edges that establish relationships (connections) between the nodes. The graph ADT follows directly from the graph concept from mathematics.*
  >
  > *A graph G is defined as follows: G=(V,E), where V is a finite, non-empty set of vertices (singular: vertex) and E is a set of edges (links between pairs of vertices). When the edges in a graph have no direction, the graph is called undirected, otherwise called directed. In practice, some information is associated with each node and edge."* [2]

  We felt that a graph was a perfect data structure for holding the road network and intersections as it allowed us to represent the roads (and lanes) as edges, and intersections as vertices connecting the edges.

- Project written in Java

  The decision to code the project in Java was made from the very first brainstorming session. This decision was made for a number of reasons. Firstly Java is object orientated which allows us, for this project, to program the code a lot more naturally and logically. Secondly Java is currently one of the most popular languages and we felt that it was important for both of us to develop some key experience in the language as both of us have had little experience to date. Thirdly Java has a large API which allows us to focus on the core of the project and not the little complex parts of programming unimportant areas.

- Simple road layout and simple intersection system

  In order to get the project off the ground we decided to ensure that we had only a basic road network to start with and simple give way rules implemented at intersections. The simplest road network is a triangle, with three roads, and three intersections. We knew that the give way rules at intersections would lead to issues later on (e.g. if there are cars at the end of all the roads and they are all giving way to each other), however we decided to program the code in such a way that we could address this issue when we came to it.

- Single car on network, and then multiple cars

  Here we decided the key issue would be just to get a single car moving, once we had achieved that we would expand to more cars, and cars spawning from multiple points.

- Static road network

  In order to keep things simple at the start we decided that the road network should be fixed at run time, and once we had some basic functionality then we would expand it to a dynamic system.

- Cars not to crash

  One of the key issues that we must face when we get cars moving is to ensure that they do not crash or overlap each other. This issue will particularly arise at intersections if cars are already queued up.

When it came to prototyping and making the initial programming some of these foundation requirements were changed. But the generalised ideas remained the same.

From the start of the project we decided that we needed a graphical interface, this decision was made for a number of reasons. Firstly, the most important reason for a GUI was that we needed to see that our simulator was actually working. We could of used a text console output but this would have been very hard if not impossible to interpret the results of and would not have made for a very useful simulator. Secondly, a GUI allows us to design the system so that it can be dynamically altered which makes it both more user friendly and more useful. When we initially outlined the requirements for the simulator we made a number of requirements for the GUI which I will explain the reasoning behind.

- Display roads, intersections and cars on roads

  This requirement is self explanatory, if you cannot see what the simulator is simulating there is not much point in even trying to use it.

- Simulates in relative real time, ie cars move every 0.1s

  We felt that having a simulator being able to run and display in real time was vitally important in developing a simulated system.

- Interaction with user, ability to add roads, intersections etc, change number of cars and number of lanes (double road capacity)

  As already mentioned being able to dynamically update the system was one of the key reasons for designing and using a GUI.

- Ability to pause simulation, and step simulation

  Having the control of the simulator built into the simulator was also a key factor for us.

When we initially planned the outline for the simulator we also decided on some brief expansion ideas that we could look at implementing if we had time. The initial ideas were:

- Different types of intersections, lights, give way, round-a-bouts

  Any road network has differing types of intersections and being able to simulate these and see how this affected traffic flows would be a good way of being able to successfully and realistically model traffic.

- Traffic light phasing

  Traffic light phasing plays a very important part in traffic congestion and getting it right is a complicated business. Not only does a local traffic light system need to manage traffic on each of its incoming roads, it also phase with other traffic lights in the surrounding area so that traffic flows smoothly across the network.

- Effect of weather, ie slow downs due to rain

  Bad weather plays an important part in traffic congestion. Generally if the weather is wet there are more cars on the road as less people are willing to walk or even catch public transport. Also severe weather can cause crashes or even close roads and this can have massive amounts of impact on a roading network.

- Time of day, ie rush hour

  It is well known that traffic flows are not linear each road and even each lane has different peaks during the course of the day and this in turn affects the rest of the road network.

- Accidents, if a road is closed or capacity is reduced how does this affect traffic flows on surrounding roads

  This happens in real life, so a good traffic model should be able to cater for this situation as well.

- Cars have basic AI system – decide what road to go based on traffic density, have a location to drive to rather than just randomly going through city

  To make it more realistic. The more realistic the simulator is the more valid its results will be.

Development of Outline Into Brief

Using the project outline as the base of our ideas we then developed the project brief. The project brief is similar to the project outline however it just provides more focus to specific areas of the project and what we are actually going to achieve.

Problem to Solve

As part of the brief we decided that we should have a few sentences that describes the basic intent of the project and our aims. After a few different drafts we settled for the following.

"It has been stated that if you build more roads and add more intersections to the road network rather then improving traffic flows you actually decrease it because of more time spend waiting at intersections. In order to try and prove this we are going to build a simulator in Java to simulate first traffic flows, and secondly the effects of changes on the traffic network."

After the problem to solve we lay out the three sections to the project, the base specifications, further specifications, and options for expansion. As already mentioned most of these were developed from the project outline. When we starting construction on the project these specifications were used as a pathway for the project development however they were not always stuck to as some of the further specifications and options for expansion became easier to implement then first thought. I have listed the original specifications below, I have chosen not to provide commentary as most of the reasoning has already been explained above, and other sections are self explanatory.

Base Specifications

- Simple road network (fixed at compile time, or read from file) implemented in Java using graphs
- Cars get generated from one area of road network, drive around and exit at another area

- Cars are not to crash into each other
- GUI
- Static Road network, fixed at compile time
- Ability to control simulator start, stop, pause, step, change driver response speed

Further Specifications

- Multiple-lane roads
- Cars changing lanes
- Cars having basic AI system, behavioral traits, start and finish locations
- Dynamic Road network

Options for Expansion

- Roads having differencing speed limits
- Weather effects on road conditions + driver behavior
- Different types of intersections
- Traffic light phasing
- Time of day – rush hour etc
- Accidents + road works
- Working in threads
- Graph File Format

Initial Design

Initial Program Design

The initial program design began with a small brainstorm of ides for functions and classes that would be required. The initial program design in this format was never completed however because the design focus was shifted and put on designing the classes first and then working from there.

The original brainstormed classes were:

- Vehicle – i.e. a car, bus, truck
- Road – This class immediately raised questions of if it was contained within the graph structure, or if this class was a part of the graph structure. Ie was a Road contained within an Edge or was a Road an actual Edge itself.
- Intersection
- Traffic Lights – The question attached to this class was if it was actually needed or could its functionality be included easier inside the Intersection class.
- Driver – The idea here was for AI capabilities.

As mentioned above the initial program design was never completed and as such the only brainstorm for any function was the main function.

Main Function

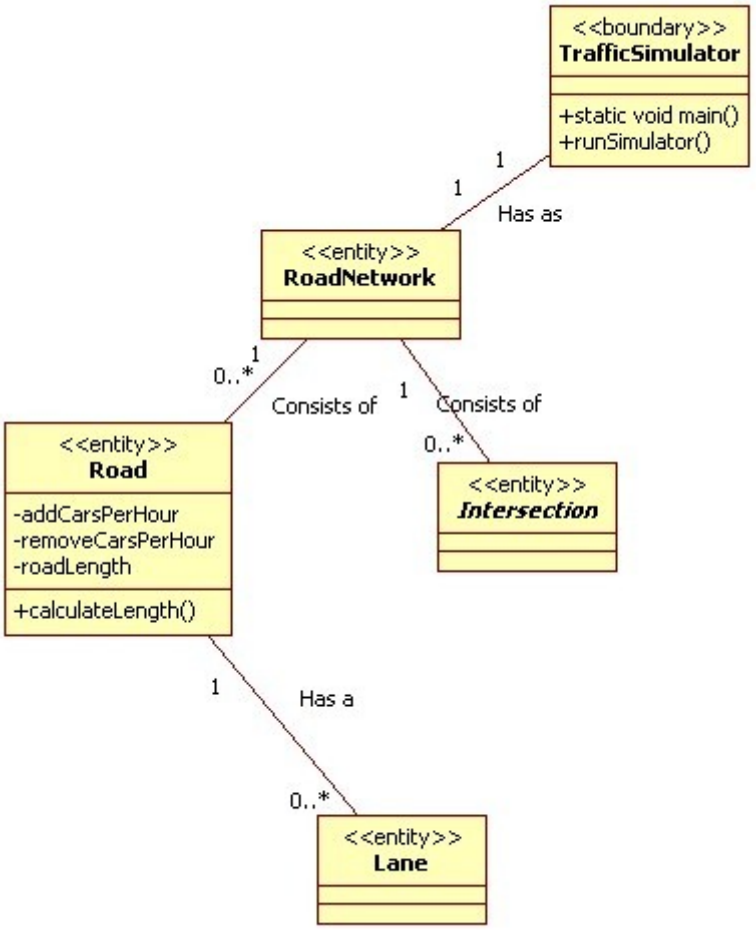
- Set up starting road network
- Set up initial cars – do we need to do this
- While true
  - generate cars, if time is right
  - drive cars – does this need to be done in threads?
  - changing traffic lights?

Text 1: Original brainstorm for Main() (26/04/2007)

Of course once we started programming the final version of the main function was significantly different from this, however, without this original idea for the function the project would not have got off the ground.

Class Design

Leading on from the original program design the class design became vital to ensuring that the design of the program would meet its requirements. Starting from our original brainstorm we used an open source UML application, Star UML, to layout and plan our classes. Using a class diagram in UML the road network became very easy to design. However, figuring out how a car worked with the road network became an issue of much discussion.



Class Diagram 1: Original Road Network Class Design

The above diagram is our original design for the classes in the road layout. This design was kept and used in the actual traffic simulator. Here the RoadNetwork class made up the holder container for the graph data structure. The Road class took on the role of an edge, and the Intersection took the role of a vertex.

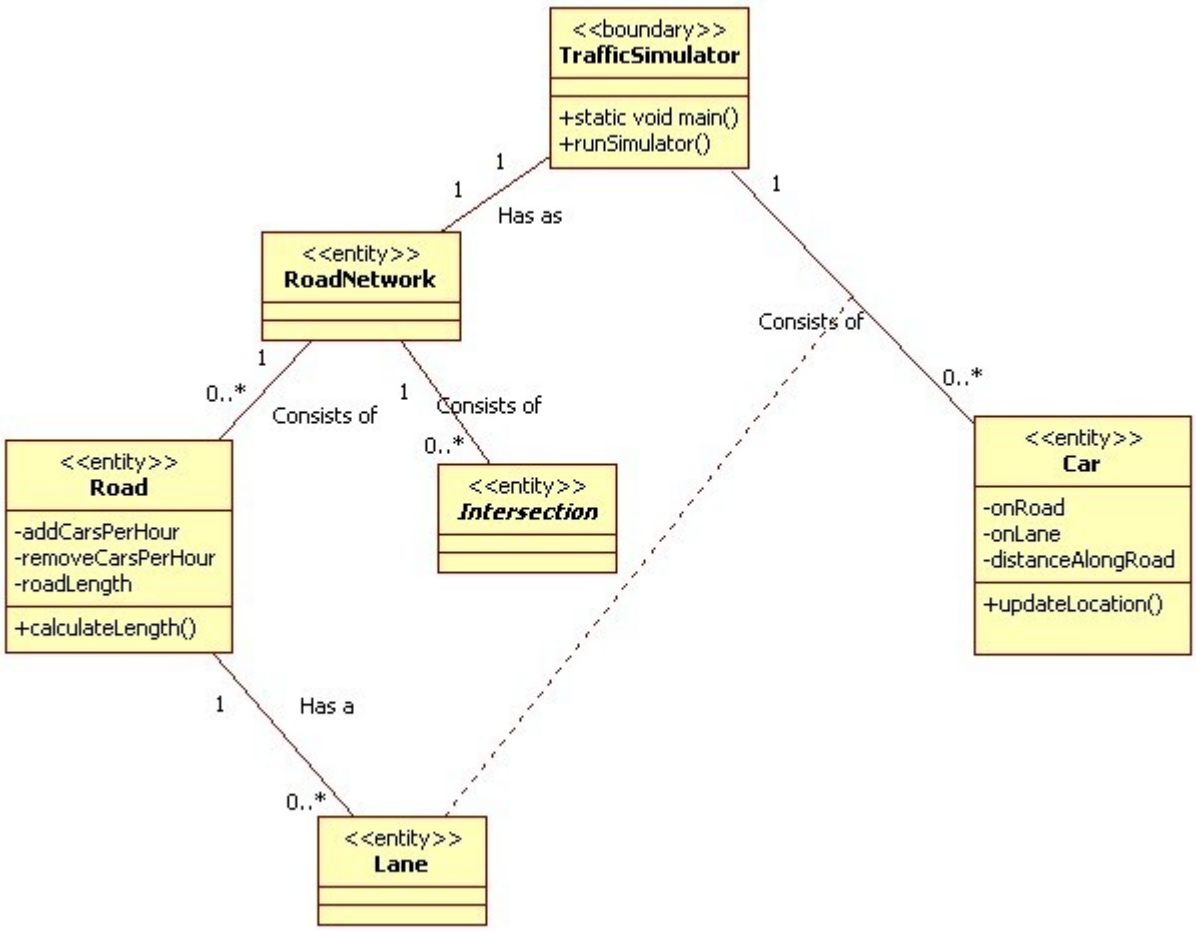


Illustration 2: Class layout with Car class

Deciding where to place the Car class became a major design headache and discussion for us. It made sense to place the Car under a Lane because after all in real life a car sits on a lane on a road on a road network. However, at the same time unlike a lane which is a part of a Road Network a car is not it is it's own separate entity to a road network.

Another major point of discussion surrounding the placement of the Car class also surrounded around how the Car would access data from the Road Network in particular the Lane. As discussed above it made the most amount of logic sense to have both the lane and the car talk directly too each other however this would be fundamentally wrong. One of the major issues that we were concerned about was code that would look similar to the following.

```
TrafficSimulator.getRoadNetwork().getRoad().getLane().getNextCarOnLane()
```

Firstly this is an example of train wreck code, which is dangerous because it is firstly very confusing and secondly is slow because of all the amount of data that needs to be moved about on the stack due to all the function calls.

In order to try and solve issues surrounding this we decided that it was important for a car to know which lane and road it was on so a call like the above would only need to be used once. This was implemented in our final product by using a data variable to provide a reference to each of these related classes.

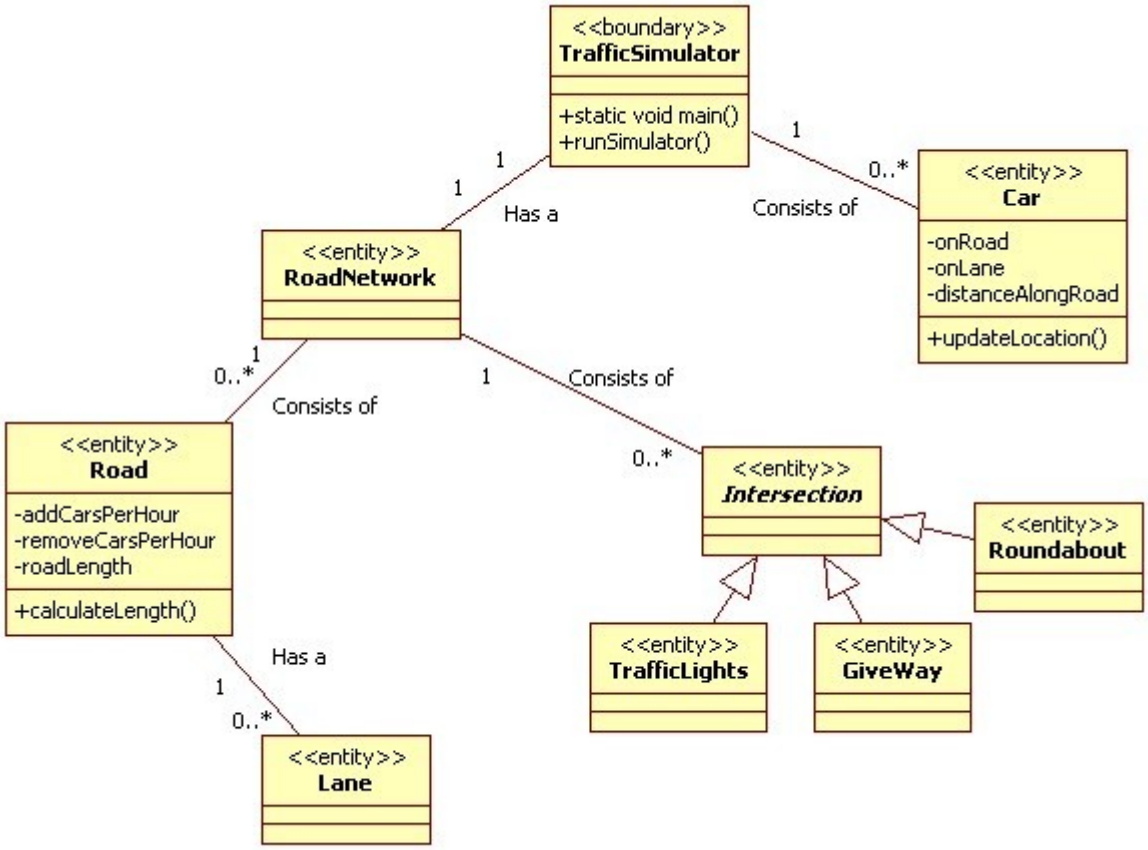


Illustration 3: Suggested Intersection Extension

The next extension to the base UML design that we added was three different ideas for subclasses of Intersection. In a real life road network there are different types of intersections with the three most common ones being Traffic Light Controlled, Give Way Controlled and a Round-a-bout. Because there are three different types of intersections it would be interesting to investigate how each of these affects traffic flows.

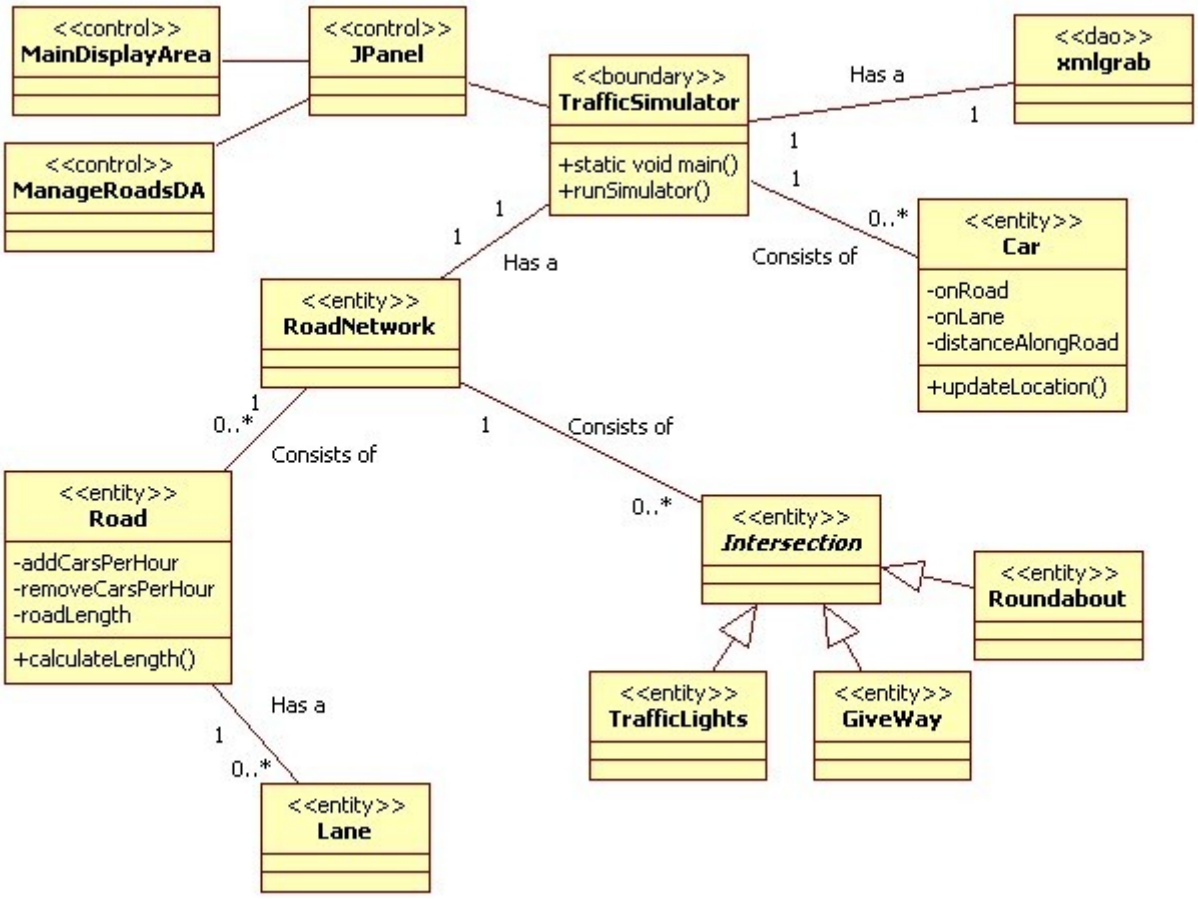


Illustration 4: Final planned class layout

This is the final version of our UML design, once we created the actual product the final class layout was a lot different to this, however the majority of the project remained structure around and based on this design.

Prototyping

Moving on from our UML Class Design we decided to do some basic prototypes to test our ideas and in particular to try and help solve our issues with the Car class location. Initially we started our prototypes using C++ rather then Java, because we have direct access to the memory (pointers) so that we can be completely sure of the interactions that are going on, and secondly both of us had more experience in C++ rather then Java so any mistakes could be solved easily.

```
#include <iostream>

using namespace std;

struct intersection {
	int id;
	int xpos;
	int ypos;
};

struct road {
	int id;
	intersection* startpos;
	intersection* endpos;
};

int main() {
	intersection a, b;
	road c;
	a.id = 1;
	a.xpos = 10;
	a.ypos = 20;
	b.id = 2;
	b.xpos = 15;
	b.ypos = 30;
	c.id = 1;
	c.startpos = &a;
	c.endpos = &b;
}
```

Code 2: First C++ Prototype

This first C++ prototype was designed to test out our ideas on using the Road for a graph edge and an Intersection for a graph vertex. It worked well with the intersections storing in memory their locations on the screen and the road storing what intersection was its start and end position.

```
#include <iostream>


using namespace std;


struct intersection {
      int id;
      int xpos;
      int ypos;
};


struct road {
      int id;
      intersection* startpos;
      intersection* endpos;
      int roadlength;         // how do we work this out? triangulation between
                                    → start and end?
      // does the road need to know what lanes it has?
      // what about other properties, surface type, speed limit etc.  these
come        → later
      car *traffic[];
};


struct lane {
      int id;
      road* parentRoad;
      bool AtoB;  // if a to b then lane goes from startpos to endpos, else
                        → endpos to startpos.
      // also do we need to know about other lanes that are going in the same
            → direction?
};
```

Code 3: Second C++ Prototype

```
struct car {
      int id;
      //int xpos;       // or should this be current road? or position on road;
                        // direction along road;
      //int ypos;       // or maybe this should be lane on road
      lane* onLane;
      int lanepos;      // position away from start point of lane, we need to
                              → get this from the road info
      road* onRoad;     // do we need this for road info?
};

int main() {
      intersection a, b;
      road c;
      car d;
      a.id = 1;
      a.xpos = 10;
      a.ypos = 20;
      b.id = 2;
      b.xpos = 15;
      b.ypos = 30;
      c.id = 1;
      c.startpos = &a;
      c.endpos = &b;
      d.id = 1;
}
```

Code 4: Second C++ Prototype cont.

This second C++ prototype is a lot more complex and more detailed then the first prototype, it was primarily set up to test ideas around the placement of the car class. As you can see in the example the code also had a lot of ideas put into it in comments. From this code we moved to Java using these ideas as the foundation for our Java classes.

Road Network Data Structures In Java

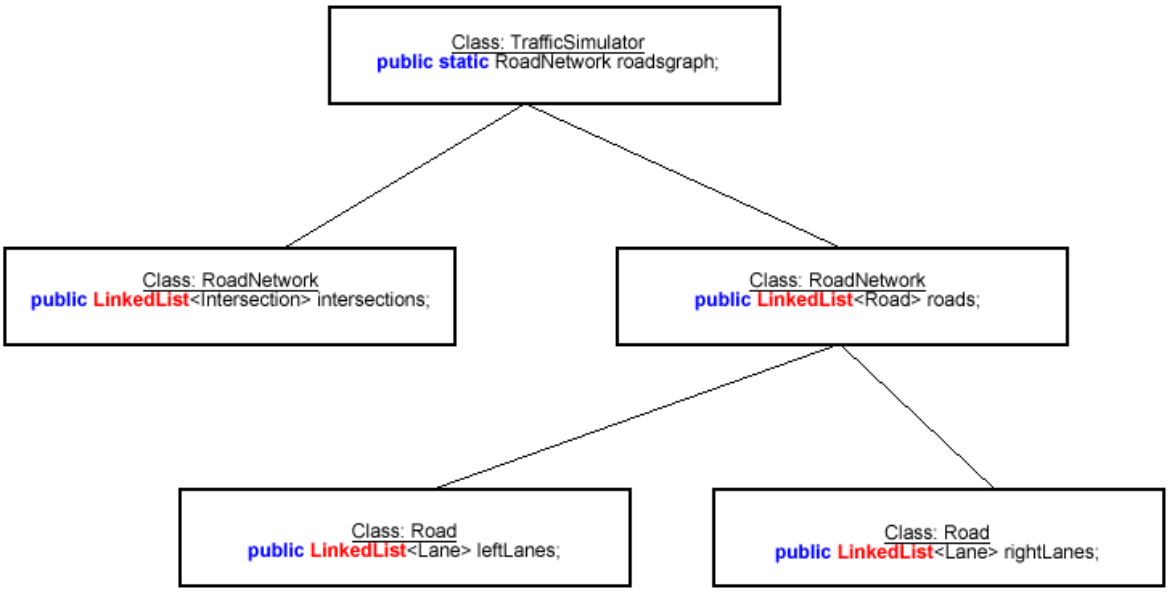


Illustration 5: Road Network Data Structure Layout In Java

The Road Network data structure is managed over five different classes and Java files. The top level main class TrafficSimulator creates the first part of the data structure, RoadNetwork. The RoadNetwork's key data is a linked list of Intersection to hold all the intersections and a linked list of Road to hold all the roads. The Road class then has two linked lists of Lane one for left lanes and the other for right lanes.

GUI Design and Implementation

GUI Design and Layout

One of the main goals of the Traffic Simulator project was for it to have an interactive GUI. Because neither Dean nor myself have had much if any experience in GUI design we decided to base the GUI for the project off the two assignments that I had done in 159.235 (Computer Graphics). There were a few important reasons for doing the GUI this way. Firstly the assignments were already based in Java so you could take a look at the code and understand most of how it worked. Secondly from an aesthetics point of view the large viewer area of the assignments allowed the user to see a lot of the core of what was going on and that is something that we desired in our project as well.

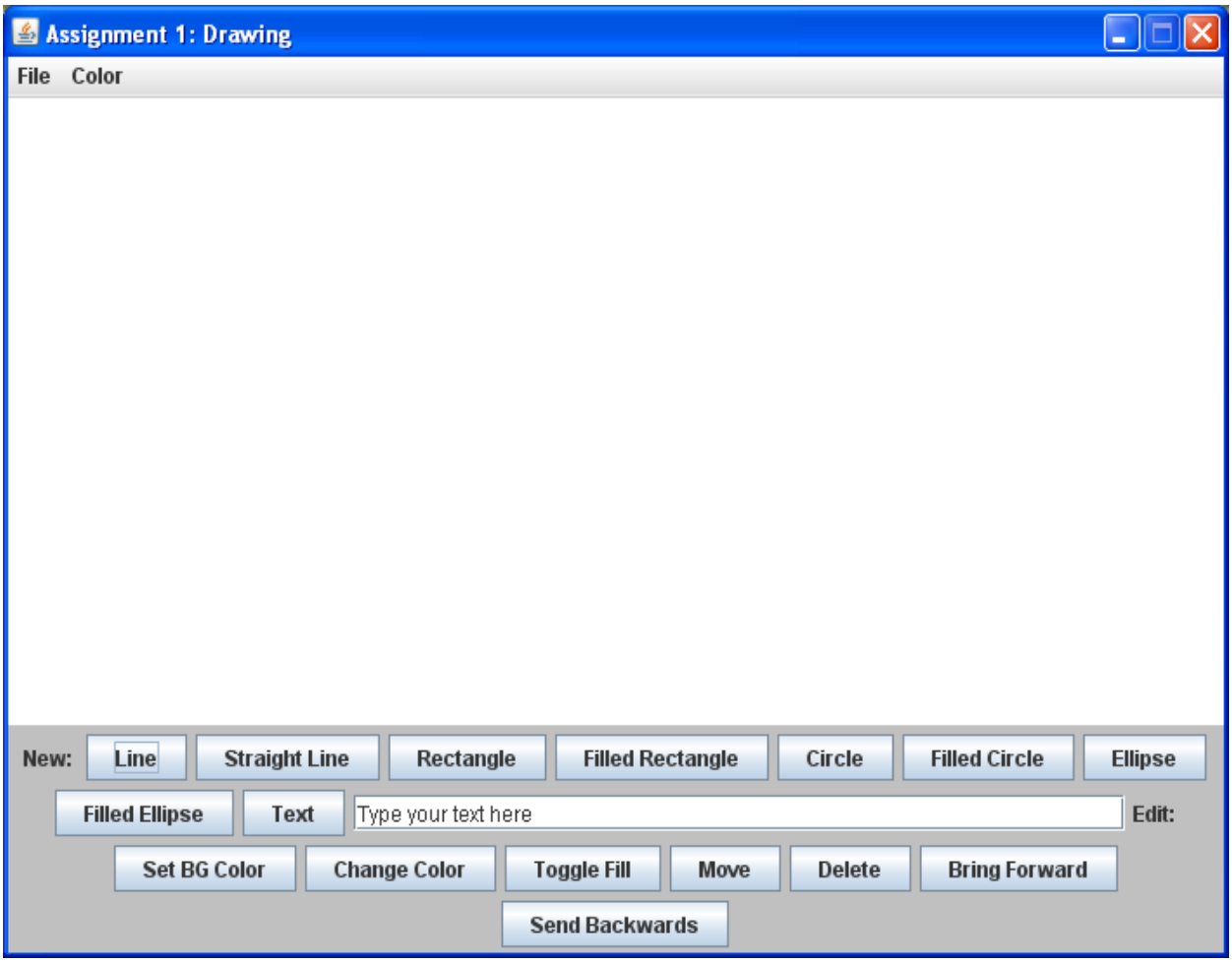


Illustration 6: 159.235 (2006) Assignment One GUI

Assignment One (159.235) is where we based most of our initial GUI code off. The key difference was that we decided that we would remove the menu from the bottom and create our menu as a separate frame.

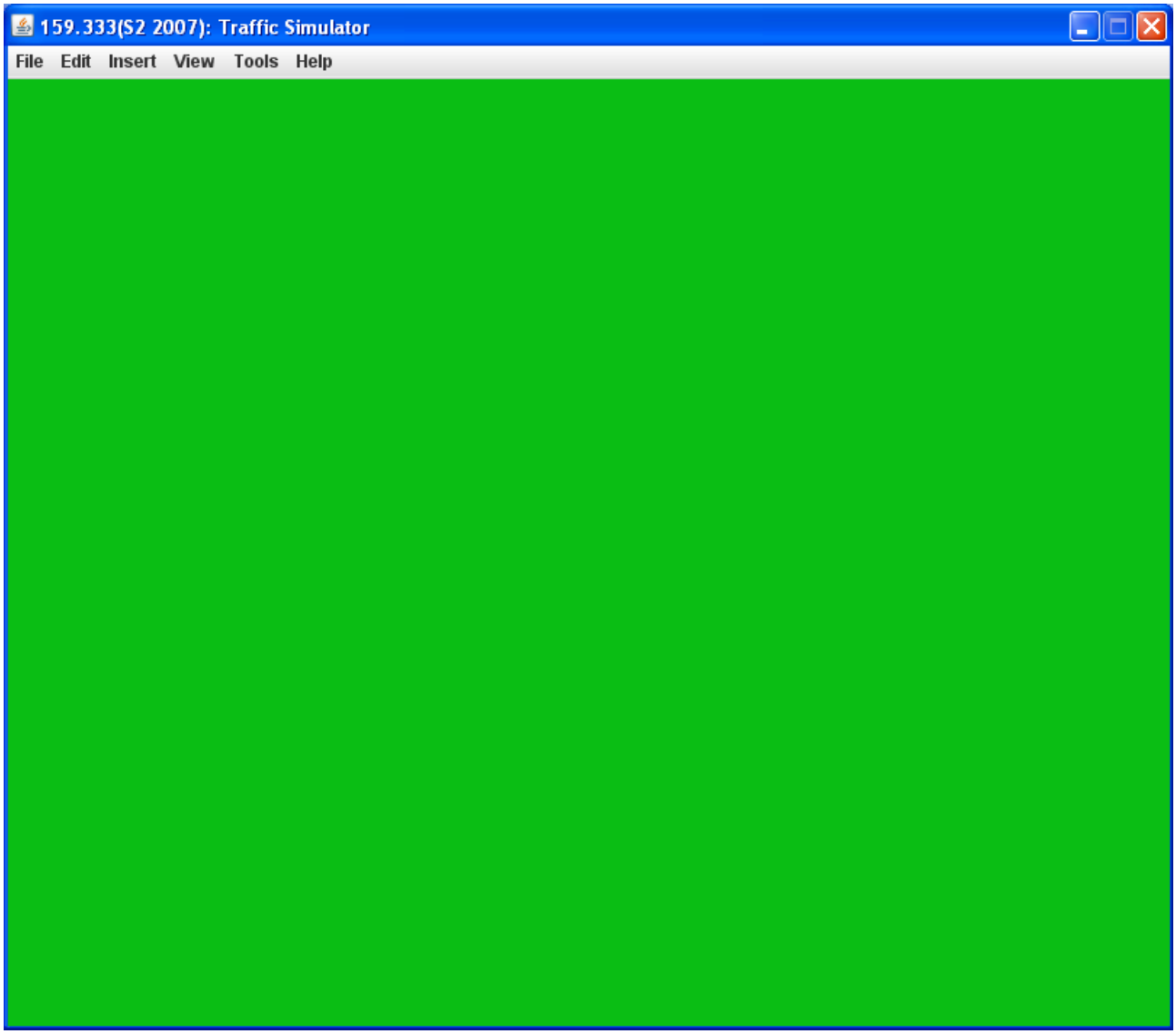


Illustration 7: GUI for the Traffic Simulator

On the coding side of the two GUI's there were some major differences. In the assignment that we based the GUI off the main class of the assignment had extended the JFrame and implemented the ActionListener. We felt that this was a fundamentally bad idea for our traffic simulator as the traffic simulator was more then just an extension of a JFrame it was a much more complex object. Therefore our main TrafficSimulator class did not extend the JFrame but rather inside it had a JFrame as a variable.

Road Network Design Pane

Both the Road Network Design Pane and the Simulator Control Pane designs were based off the Viewer Control Panel of Assignment Two 159.235. The Road Network Design Pane consists of tools that allow you to create and alter Roads, Intersections and Lanes.

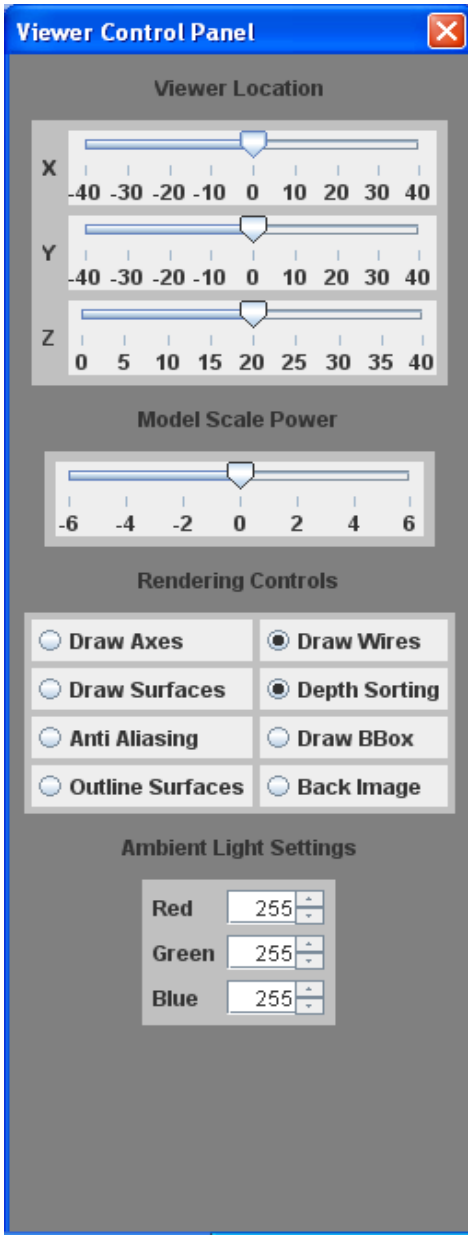


Illustration 8: 159.235 Assignment Two Viewer Control Pane

Illustration 9: Manage Roads Dialog

Simulator Control Pane

The Simulator Control Pane took a similar form to the Road Network Design Pane this is primarily for consistency across the application and also for ease of programming because if they are programmed in the same manner it is easier to understand.

User Interaction with GUI

The interaction between the control panes and the main window of the GUI is one of the most important aspects to the successful functionality of the simulator. Dependant on what the user wishes to do there are a lot of different interactions that occur during the running of the application.

I will briefly go over the functionality of each of the options in the two control panes.

Common to both

- Zoom Factor

  The Zoom Factor allows the user to zoom/scale in and out from the Road Network Model. The original design for the application did not have any plans for a zoom function and instead set the model up with a rule of 1px is equal to 2m. After a while this became unworkable because in some situations we wished to be able to zoom in and see particular cars moving across the screen and in other circumstances you wanted the model to be able to be larger then the fixed size. When implementing the zoom function because of our original non 1 to 1 scale we faced major issues over getting the zoom correct. This was solved with a few constants and division factors.

- Viewer Location

  This idea came from both the idea of being able to zoom in and out from the model and from the functionality used on Google Maps. These buttons allow the user to move view point for the model across different areas of the model/screen.

Road Network Design Pane

- New Intersection

  The New Intersection button is the key to all the functionality of the Traffic Simulator without it we would not be able to build a road network to simulate. Once pressed the next time the user clicks somewhere on the main GUI window a new intersection will appear. By default this intersection will also be highlighted as the B intersection.

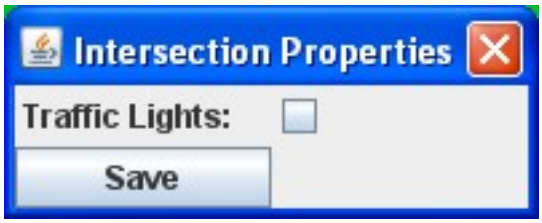
- Select Intersection A/B

  The Select Intersection buttons allow the user to highlight intersections. The role of these two buttons is to let the user select the two end points for a road and then act on this road (create new, remove, add lanes, etc). Once pressed the select intersection button acts in almost the same way as the new intersection button as the next click on the main window GUI that is contained within an intersection that intersection will become the selected intersection.

- Move Intersection

  The functionality of this intersection is derived from the select intersection buttons. Once pressed it allows the user to click down on an intersection contained on the main GUI window and drag it around the screen and placing it in a new location.

- Intersection Properties

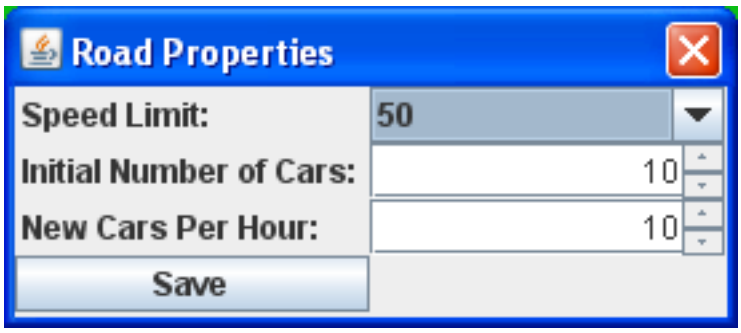
  The intersection properties relies on an intersection being highlighted as intersection A. If this intersection is highlighted the button will pop up a dialog box prompting the user to state if the intersection is controlled by traffic lights (if the box is unchecked it is controlled by give way).

- Delete Intersection

  The delete intersection button again borrows its functionality from the select intersections buttons. Once clicked the next click in the GUI on an intersection will cause it to be removed from the linked list of Intersection.

- Make Road

  The make road button as one can automatically assume makes a road. It relies on two intersections to be selected, as A and B. Once clicked the listener for this button will create a road between these two selected intersections. It will also pop up a dialog box asking about the Road Properties such as the speed limit, number of initial cars to generate and the number of new cars that spawn on this road every hour.

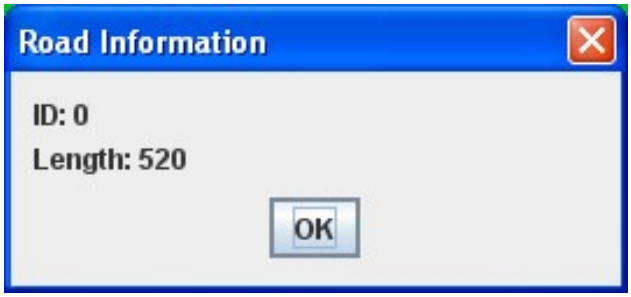


- Road Properties

  This button allows for the properties of a road to be altered after it has been created, such as changing the speed limit. It relies on intersections A and B to be highlighted and once clicked will pop up the same dialog box as what appears when you initially create a road.

- Road Info

  This button also relies on two intersections A and B to be selected and a road to exist between the two points. When the button is pressed it displays a dialog box that shows some fixed information about the road (ID in the linked list, length, etc).



- Delete Road

  The delete road button deletes a road from the Road Network. This relies on a road existing between two selected intersections A and B.

- Add Lane A-B

  The add lane A to B button adds a left lane along the road from intersection A to intersection B.

- Add Lane B-A

  The add lane B to A button adds a left lane along the road from intersection B to intersection A, in other words it does a lane in the opposite direction to add lane A-B.

- Remove Lane A-B

  Remove Lane A to B removes a left lane heading from A to B from the road.

- Remove Lane B-A

  Remove Lane B to A removes a left lane heading from A to B from the road.

- Simulator Controls

  The Simulator control button removes the Road Network Design Pane from the screen and displays the Simulator Control Pane in its place.

Simulator Control Pane

- Start Simulator

  The Start Simulator button does a lot more then just starting the simulator. Firstly it has to go through a lot of preparation code to set up all the variables etc to make the simulator run. The following is some simplified pseudo code of how it works

```
De-select Intersections A and B

Set simulation running time to 0

Create an int containing the total number of roads in the road network

loop through all the roads in the network

      for each road in the network:

      calculate the length of the road

      for each intersection at the end of the road add this road to its
            → connected list

      create an int containing the number of cars to generate for this
            → road

      get the furthest left and right lanes for the road, cars should
            → always be generated in these spots

      create a for loop up to the number of cars to generate

            randomly select if to add a new car to the lane to a left
                  → or right lane

            add a new car to the linked list of Car

            give the car information on the current road and lane that
                  → it is on, also the length of the road that is on,
                  → and its random current position on this road

repaint the screen with all these new cars

start a screen refresh thread

set the simulator running
```

Text 5: Simulator Start Pseudo Code

- Pause Simulator

  The Pause Simulator button simply sets the simulator running boolean to false.

- Step Simulator

  The Step Simulator button allows you, once the simulator is paused, to move the simulator forward one time step at a time. In the final version for this thesis, however, the full functionality of this button was never complete.

- Stop Simulator

  The Stop Simulator button stops the simulator from running. Once stopped it clears out all of the cars from the Car linked list and also removes the list of connected roads from each of the intersections to enable to user to edit the road network again if they desire.

- Simulation Rate

  The Simulation Rate Spinner controls how quick the simulator runs, starting from 1x (real time) it can allow the simulator to run at up to 100x real time.

- Manage Roads

  The Manage Roads button hides the Simulator Controls control panel and replaces it with the Manage Roads control panel.

Error detection and checking

A user interacting with the GUI can cause a lot of errors and issues they make a mistake while using it. While programming the assignment we decided that if any common error occurred we would just detect it and ignore it, the common errors that we did this on are:

- Trying to select an Intersection where there is no intersection
- Move Intersection where there is no intersection
- Intersection Properties with no intersection selected
- Delete Intersection where there is no intersection selected.
- Make Road where there are no intersections selected.
- Road properties where there are no intersections selected
- Road properties where there are no road
- Road info where there are no intersections selected
- Road info where there is no road
- Delete road where there is no intersection selected
- Delete Road where there is no road
- Add lane where there is no road
- Add lane where there are no intersections selected
- Remove lane where there are no intersections selected
- Remove lane where there is no road
- Start simulator after simulator is started
- Pause simulator after simulator is already paused
- Stop simulator when simulator is not running.

Menu Bar Design

The menu bar is designed to both compliment the Road Network Design and Simulator Control panes and add more functionality.

File Menu

The file menu for the Traffic Simulator has similar functions to those that you would expect on any program with a GUI.

- New
  The New button on the menu clears all existing data about the Road Network from the memory and allows the user to start from scratch.
- Open
  The Open button displays a Chooser box and then loads an XML file that describes a Road Network into the Simulator.
- Save
  As you would expect Save saves a Road Network. If a file has already been opened or saved it will save over the top of this file, if no file has been opened or saved it will display a Chooser box allowing the user to choose a filename and file location.
- Save As
  Save As works the same as Save with the exception that always allows the user to chose a filename and location.
- Exit
  As you would expect Exit quits the program.

Edit Menu

All the functions in the Edit Menu perform the same task as their counterparts in the Manage Roads dialog. Those functions are:

- Select Intersection A
- Select Intersection B

- Move Intersection
- Delete Intersection
- Delete Road
- Remove Lane A-B
- Remove Lane B-A

Insert Menu

Like the Edit Menu the Insert Menu just provides redundancy for those functions that are available through the Manage Roads dialog,

- New Intersection
- New Road
- New Lane A-B
- New Lane B-A

View Menu

The View Menu controls the display of various components of the simulator.

- Manage Roads
  This button displays or hides the Manage Roads dialog. If the simulator controls dialog is visible it is hidden.

- Simulator Controls
  This button displays or hides the Simulator Controls dialog. If the Manage Roads dialog is visible it is hidden.

- Choose Background
  The Choose Background button allows the user to select a JPEG image as a background for the simulator, this allows you to load in a Satellite image or road map to trace a Road Network off.

- Show Buttons

  The various show buttons allow the user to suppress various components of the simulator. This can also be used to speed up display when there are lots of cars.

Tools Menu

This menu was never completed. It was designed to potentially hold an individual car analyser, a data logger, and other statistical tools.

Help Menu

The Help menu only contains an about dialog. If we were developing a commercial product it would contain other documentation.

Simulator Design and Implementation

Initial Simulation Loop

```
while( true ) {
	startTime = System.currentTimeMillis();
	if(simulatorRunning) {
		Collections.sort(cars);		// put the cars into the order from
			→ furtherest to end of road, to closest to start of road
		ListIterator<Car> carsptr = cars.listIterator();
		while(carsptr.hasNext()) {
			Car currentCar = carsptr.next();
			currentCar.moveOneStep();
		}
		if(simulatedTime%50==0) {	// refresh the screen every 50
			→ milisecs or 20fps
			da.repaint();
		}
		simulatedTime += 10;
	}
	// then sleep until right time to continue
	elapsedTime = System.currentTimeMillis() - startTime;
	waitTime = Math.min( 10 - elapsedTime, 10 );
	if(waitTime < 1) waitTime = 1;
	try {
		Thread.sleep( waitTime );	// this will sleep for 10 milisecs
	} catch ( InterruptedException ie ) {}
}
```

Issues with initial design

This initial simulator loop was simple and worked well to get the cars moving. This section of code along with the moveOneStep() code of the Car class was the most time consuming and complicated part of the whole project. As we developed more functionality into this code section and the car controller function (moveOneStep()) we discovered a number of

issues with this initial loop, the main issues are discussed below:

- Fixed Simulation Rate

  In order to initially make the cars move the simulation rate was fixed. However we made a number of mistakes in the initial implementation of this. Firstly we got ourselves confused on the initial code. In our brief we had stated that we wanted the cars to move at 10 times real time, and have a response time of 0.1 sec (ie they move in 0.1sec time gaps). However when we coded this (and the car controller function) we managed to get ourselves confused between real time (ie the actual time in which the simulator runs) and the simulated time of 10x, as a result we thought we had everything working well, but in fact we had a mixture of things running at 1x and 10x rates, when they should of all been at the same rate.

  This issue took us a number of weeks to realise and it was not until we decided to allow the user to increase and/or decrease the simulation rate that we realised that there was a major issue here. This resulted in the rewriting of both this code section and the car controller function.

- Slow Refresh Rate

  After we had resolved the issue surrounding the simulation rate a new problem arose. If you increased the simulation rate the refresh rates of the screen would also increase and you would hit a point where everything would start going slower because the screen would try to refresh while the cars were trying to also move.

  This issue took a number of different approaches and a number of days to resolve. Our first approach was to try and sort out the timing within the loop itself so that the screen refreshed every 20ms, rather then every time around the loop. However, trying to get this to work actually became a major issue as once you increased the simulation rate you often ended up in a situation where you would be looking for a result such as refreshtime % 20 == 0, and in fact because the loop was running every 7ms the refreshtime would be equal to 21 and this wouldn't work.

The second major approach that we tried was to make the time working if the refreshtime was > 20. However, again this didn't work. The issue that arose here was with resetting the timing and also the associated calculations involved in the if statements slowed dramatically slowed down the simulator.

In the end we resolved to putting the refresh screen function inside it's own separate thread and this allowed it to refresh every 20ms independent of the main loop of the simulator. However this lead to concurrency issues when both the main simulator loop and the refresh display thread tried to read and write to data at the same time.

- Cars never leave simulator

  All the initial simulator loop did was move the cars. This was initially great because it achieved our primary goal of simulation. However, cars in real life do not drive around aimlessly forever they will normally have a destination to get to and drive with some intelligence (maybe not much if you are an Aucklander) to get there. In order to make the simulator more realistic the simulator randomly selects a goal location for each of the cars that it generates, when the car reaches its goal it is removed from the list of cars.

  The application of this caused concurrency issues because cars would be removed as they were trying to be written to the screen.

- Concurrency Issues

  As mentioned errors arose with concurrency with the screen refresh thread conflicting with the main simulation loop. This was caused because cars were being removed from the simulator at the same time as the refresh screen thread was trying to display the cars on the screen. This problem took a number of days and various different approaches to eventually solve.

I Initially tried to solve the problem with putting the two sets of code into a synchronised block based on the variable Cars (LinkedList<Car>) however the synchonisation of the two sections of code dramatically affected the smooth running of the simulator so we were not able to use it as a successful solution.

Another idea was to make a copy of all the cars, however this would take up a lot of memory and time in doing the copy. However, this lead to our solution of the issue, doing a shallow copy of the linkedlist of Car which only holds references to each of the cars in memory and not the cars themselves.

Improved Simulation Loop

```
while( true ) {
      startTime = System.currentTimeMillis();
      if(simulatorRunning) {
            LinkedList<Car> carsToRemove = new LinkedList<Car>();
            Collections.sort(cars);       // put the cars into the order from
                  → closest to end of road, to closest to start of road
            ListIterator<Car> carsptr = cars.listIterator();
            while(carsptr.hasNext()) {    // move all the cars
                  Car currentCar = carsptr.next();    // get the next car to move
                  // determine if car is at its goal
                  if(currentCar.onRoad == currentCar.endLocation &&
                        → currentCar.distanceAlongRoad >= currentCar.endSpot -
                        → 15 && currentCar.distanceAlongRoad <=
                        → currentCar.endSpot + 15) {
                        carsToRemove.add(currentCar);
                        continue;
                  }
                  Car inFrontCar = findCarInFront(currentCar);    // find if
                        → there is a car in front
                  currentCar.moveOneStep(inFrontCar);       // move the car
            }
            ListIterator<Car> carsToRemoveptr = carsToRemove.listIterator();
            while(carsToRemoveptr.hasNext()) {
                  cars.remove(carsToRemoveptr.next());
            }
            carsToRemove.clear();
            if(simulatedTime%30000==0) {  // change traffic lights every 20 secs
                  // go through each intersection, and move next road to green.
                        → only if that road has an incoming lane
                  ListIterator<Intersection> intersectionsptr =
                        → roadsgraph.intersections.listIterator();
                  while(intersectionsptr.hasNext()) {
                        Intersection currentI = intersectionsptr.next();
                        currentI.changeGreen();
                  }
            }
```

```
            simulatedTime += 1000 * CAR_RESPONSE_TIME;
                        // keep track of the time
            if(cars.size()==0) {
                  simulatorRunning = false;
                  JOptionPane.showMessageDialog(mainWindow, "Simulation Complete
                        → \nRunning Time (s): "+simulatedTime/1000, "Finished:
                        → Traffic Simulator",JOptionPane.INFORMATION_MESSAGE);
                  // go through each intersection and remove all the connected
                        → roads
                  ListIterator<Intersection> intersectionsptr =
                        → roadsgraph.intersections.listIterator();
                  while(intersectionsptr.hasNext()) {
                        Intersection currentInt = intersectionsptr.next();
                        currentInt.connectedRoads.clear();
                  }
                  da.repaint();
            }
      }
      // sleep until right time to continue
      elapsedTime = System.currentTimeMillis() - startTime;
      waitTime = Math.min( (long)( (1000 / SIMULATION_RATE) *
CAR_RESPONSE_TIME)            → - elapsedTime, (long)( (1000 / SIMULATION_RATE)
*
            → CAR_RESPONSE_TIME) );
      if(waitTime < 1) waitTime = 1;
      try {
            Thread.sleep( waitTime );
      } catch ( InterruptedException ie ) {}


}
```

This simulator loop is a lot more complex them the original design. However, it address all of the issues faced by the first loop, it also allows for removing cars if they get to their goal and changing traffic lights.

Car Controller Design and Implementation

Car Class Design

```
1  public class Car implements Comparable<Car> {
2      private int currentSpeed;
3      public Road onRoad;
4      public Lane onLane;
5      public float distanceAlongRoad;
6      private Road startLocation;
7      public Road endLocation;
8      public float endSpot;
9      private double driverSpeedFactor;
10     public Color carColor;
11
12     public Car( Road whichRoad, Lane whichLane, int length, Road end) {}
13
14     public int compareTo(Car c) {
15         return (int)c.distanceAlongRoad - (int)distanceAlongRoad;
16     }
17
18     public void moveOneStep(Car inFront) {}
19
20 }
```

Illustration 10: Core of Car Class

Each car needs a lot of information about itself and its surroundings, it also needs a car controller function which can easily become very complex. The code sample above shows each of the variables, most are self explanatory.

Two of the most important variables are the endSpot and the driverSpeedFactor. The endSpot is the location along the endLocation road that the car finishes at. The driverSpeedFactor is a random normally distributed variable that describes how much the driver sticks to the speed limit. If it is above 1 the driver will break the speed limit, if it is below 1 the driver will drive below the speed limit. This variable allows each of the cars in the simulator to have an individual behavior.

The moveOneStep function is the controller for the car. It will move the car along the road, change lanes, move through an intersection onto a new road, pass other cars, accelerate, brake, and anything else that a car needs to do. The development of this function took up a lot of the time spent on the entire project, and the four main versions of the function are described below.

Car Controller V1: Initial Implementation

```
public void moveOneStep() {
      int currentSpeedLimit = onRoad.getSpeedLimit();
      if(currentSpeed == currentSpeedLimit) {
            if(currentSpeedLimit==30) distanceAlongRoad += 0.0833333333;
            else if(currentSpeedLimit==50) distanceAlongRoad += 0.138888889;
            else if(currentSpeedLimit==60) distanceAlongRoad += 0.166666667;
            else if(currentSpeedLimit==70) distanceAlongRoad += 0.194444444;
            else if(currentSpeedLimit==80) distanceAlongRoad += 0.222222222;
            else if(currentSpeedLimit==100) distanceAlongRoad += 0.277777778;
      } else {
            currentSpeed += 1;      // acelerate the car by gaining 1 kmh per
                  → 10 milisecs so it will go 0 to 50kmh in 5 secs
            distanceAlongRoad += currentSpeed *
                  → 0.00277777777777777777777777777778;  // move the car
                  → along the road.


      }
      return;
}
```

This is the very first successful version of the car controller function. It allowed the car to start at 0 kmh and accelerate up to the speed limit of the road. However, this function had many shortcomings; the main thing being the cars had no intelligence whatsoever. The cars were not aware of what road they were on, where on the road they were, if they were even on a road, if there was any other cars around them (they could drive right through another car and not realise it), they could drive off the end of the road because they would never know where the road ended, and they would always drive at the speed limit (they couldn't and never would brake). Because of all of this we of course had to develop and artificial intelligence system for the cars and this was implemented in the second version of the code.

Car Controller V2: Artificial Intelligence

The second version of the Car Controller was a lot more complex then the initial loop. The major addition was the inclusion of basic AI functions. The key features of it are:

- Cars know how far they will travel in two and six seconds.
- The cars will stop at the end of a road
- Cars have a speed factor which means they all travel at slightly different speeds
- Cars can pass other cars

(see appendix for code)

Car Controller V3: Collision Detection and Awareness

The third version of the Car Controller Code saw a significant rewrite of the code. It ensured that the code was both quick and understandable. The only major change was that cars could change roads. No other major additional features were added; instead it functioned as a major clean up of version 2 of the code.

(see appendix for code)

Car Controller V4: Polished Version

The final version of the Car Controller code was another rewrite. It was based off the lessons learned in the first and third implementations of the car controller. In this final version the cars had the greatest amount of awareness and realism of any of the previous versions. However, because of this added awareness the amount of code and processing time is greater then past versions, but I believe that this added realism will outweigh the performance lost through the added processing.

(see appendix for code)

Limitations of Artificial Intelligence System

The final Car Controller is far from perfect and not something that would probably be contained in a commercial product. The most important missing feature is a decent navigation system for the cars. Because the cars travel through a graph some form of graph traversal would make the most sense. We never had the time available to try and implement a graph searching method. One of our biggest theoretical problems that was never tested was the processing time of doing a graph traversal. A system such as and A* search would, in our opinion, be too costly in terms of performance, while something like an uniform-cost search during initialisation would also slow the cars down too much, and on the same token in real life how many people take the absolutely perfect route to a destination. And what makes that the perfect route? It may be the shortest but in terms of traffic load it make in fact be the slowest.

One of the few experiments that we did do was to use a system where at the end of each of road the car would select which road to go onto next by working out the two dimensional distance to the goal from each of the connected roads and then head down the one with the smallest distance to travel. However, this did not work because quite often you would have a road that is the closet to the goal but its connecting roads do not connect to the road that the goal is on.

Intersection Controller and Implementation

Background

In real life when you are driving on the road there are generally three types of intersections that connect roads: round-a-bout, give way, and traffic lights. In order for a traffic simulator to be realistic one must try and simulate all three of these situations.

Initial Implementation

In some ways it is slightly ironic to name this section Initial Implementation as there really was no real initial implementation instead crossing intersections was controlled by the car controller and it would just randomly select the next road to go to and cross it without any awareness of any other cars on the intersection.

Simple Give Way

A development of the original implementation made by the car controller was a simple give way system. On the roads in New Zealand you at a give way sign you are required to always give way to the left. In a simulated computer environment it is a lot harder to work out what is left and right of you. One approach to solving this issue would be to use a system where if there is a car at any of the other connected roads to wait and give way to it. However, this approach causes issues if there are cars at the ends of all of the connected roads because they would all be giving way to each other and you would end up in a deadlock. In the simulator we over came this by allowing the cars to only take into account if there was room for them to cross the intersection onto another road. If there was room the cars would cross, if there was no room the cars would not cross. Using this system there is a chance that two cars would cross the intersection and end up on different roads, (and in real life this would probably result in a crash), however the chances of this occurring are reasonably low.

Traffic Lights

Traffic lights were the next development from the simple give way system. Traffic lights allowed large blocks of cars on a road to all move onto other roads in quick succession. It also prevented the issue two or more cars potentially crossing the intersection at the same time and causing an accident. However, the traffic light system that we implemented was extremely basic and lacked a lot of functionality that most modern traffic light systems have.

The simple traffic lights worked by changing the road that was green every 30 seconds. We chose the time gap of 30 seconds for a number of reasons; firstly it closely matches the time that most intersections operate on in real life. We also did some basic experiments with different times, ranging from 10 – 40 seconds and from our basic analysis we found that 30 seconds ensure the best traffic flow.

As mentioned there are lots of limitations with this simplified system. Firstly the time that the light is green is fixed, and even if there are no cars on the road it will remain green for the fixed time. Secondly there is no orange light; it is either green or red. Thirdly a major limitation of our system is all roads are controlled by traffic lights even if they only connect two roads directly, in a better system you would have an option of either traffic lights or give way depending on the requirements of the intersection. Fourthly, cars cannot trigger the intersection they have to wait for the control to switch to them and as already mentioned if roads don't have cars on them they will remain green for their time period so this can severely affect the simulation and is not a true representation of what would happen in real life.

Summary, Conclusions and Recommendations

Major issues faced/Missing and Incomplete Features

Object Orientated Programming Approach

Using object orientated programming both aided and hindered this project. The majority of ideas related to OOP helped us to make a successful program. However, some aspects of the OOP approach also hindered the performance of the application. Private variables within data structures was our biggest issue and in the end we had to make a lot of variables public (and in the process broke a taboo of programming).

Making the many of the variables contained with the data structures public was done for reasons of performance. The key issue was too many function calls affecting memory on the program stack. Being able to directly access a variable rather then make a function call for a reference to the variable reduced the load on the stack, used less CPU time and also resulted in lower memory usage. For our simulator to be quick when simulating a large road network the decision to make variables public over private was vital.

Documentation

Working in a team presented a number of challenges. Documentation of almost everything was important in minimising confusion and other issues. We tried to make a lot of our code Java Doc compatible but again, given time constraints; we were not able to do this too all our code. However, in a commercial product this would be vital.

Documentation of code was also incredibly important in this project because the two of us were often working on different classes and needed to understand each others code.

Testing

One of the biggest things that we were unable to complete, due to time limits, was to do multiple tests of the simulator in different conditions. Doing thorough tests of the simulator

would be a complex and time consuming exercise.

There are a few different approaches that could be made to testing. One approach would be to compare the results of the simulator with that of real life. This would allow you to see prove that the simulator did simulate the real world well. However, implementing this approach to testing would have some major hurdles. Firstly, getting the simulator to a point where it could be considered that it simulates the world well, this cannot be said of the current simulator. Secondly, getting real world results to compare to. For a few cars this would be easy, however, the simulator is designed for a large scale simulation and setting up a real world test of this would be near impossible.

A second approach would be to set up a series of tests such as one major connector roads vs. two and the like. This would allow you to investigate if commonly cited "facts" such as having less connectors to a motorway network improves traffic flows are true. We did have time to try one of these situations. We set up a network where there were two sections of the road network connected with either a single or double road. We tested both network layouts 50 times each timing the amount of time that it took for the network to clear of all cars. Our results suggested that having only one connecting road was better then two. However, we believe that this was caused by limitations with the simulator rather then being able to be used as a true result. We believe that the limitations that caused this were the fact that the cars do not have any form of navigation system so with the two connecting roads there were more roads for the cars to randomly decide to drive on so it took longer because of this, if the cars had a navigation system the results may be different.

Because the simulator that we built does not successfully simulate everything about a real road network and the cars lack a successful navigation system I do not believe that you can get any realistic or scientific results from the system.

Comparison of final version to requirements laid out in brief

I believe that our end project resembles what we laid out in the brief well. We managed to achieve the goal of simulating traffic and we also implemented a lot of additional functionality. We were not able to achieve a few pieces of functionality one of these was the affect of different times of the day and also the affect of weather. However, until the car controller and artificial intelligence was

stronger I would not put this high on a priority list. Overall I believe that the final simulator successfully meets the requirements of the brief.

Recommendations for a future system

If I was to repeat this project again I would keep with the same approach that we did here. The logical ideas of object orientated programming that allowed us to model the real world environment in code was a great asset and help to the project.

For the road network I would retain the graph structure and build in additional functionality such as different types of intersections and a stronger traffic light controller.

For the cars, firstly I would get a copy of the road code and ensure that the cars obeyed as many road rules as possible. The key to making the simulator a success is how well it resembles real life. I would also invest a lot of time in making a good navigation system for the cars.

As far as testing goes I believe that it was one of the most important features that we did not implement. A series of fair tests needs to be compiled and then implemented with the results compared to those that you would expect from real life situations.

Appendices

Car Controller Code V2

```
 1: public void moveOneStep(Car inFront) {
 2:     int currentSpeedLimit = onRoad.getSpeedLimit();
 3:     double maxDistanceToMove = 0;
 4:     // convert the speed to meters per second, and then amount we are moving by
 5:     double distanceInTwoSec;
 6:     double distanceInSixSec;
 7:     if( (currentSpeed + driverSpeedFactor*15) > 15) {
 8:         distanceInTwoSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME
                + driverSpeedFactor * (15 / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME);
 9:         distanceInSixSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME
                + driverSpeedFactor * (15 / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME);
10:     } else {
11:         distanceInTwoSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME;
12:         distanceInSixSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME;
13:     }
14:     int roadLength = onRoad.getLength();
15:     // work out the maximum distance you can move at the current speed
16:     if(currentSpeed == currentSpeedLimit) {
17:         if(currentSpeedLimit==30) maxDistanceToMove = 30 / 3.6 * TrafficSimulator.
                CAR_RESPONSE_TIME;
18:         else if(currentSpeedLimit==50) maxDistanceToMove = 50 / 3.6 *
                TrafficSimulator.CAR_RESPONSE_TIME;
19:         else if(currentSpeedLimit==60) maxDistanceToMove = 60 / 3.6 *
                TrafficSimulator.CAR_RESPONSE_TIME;
20:         else if(currentSpeedLimit==70) maxDistanceToMove = 70 / 3.6 *
                TrafficSimulator.CAR_RESPONSE_TIME;
21:         else if(currentSpeedLimit==80) maxDistanceToMove = 80 / 3.6 *
                TrafficSimulator.CAR_RESPONSE_TIME;
22:         else if(currentSpeedLimit==100) maxDistanceToMove = 100 / 3.6 *
                TrafficSimulator.CAR_RESPONSE_TIME;
23:     } else if( distanceAlongRoad + distanceInSixSec < roadLength || (inFront!=null
            && distanceAlongRoad + distanceInTwoSec < inFront.distanceAlongRoad - 6) )
            {
24:     // make sure car is not meant to be slowing down, before you accelerate it
25:         currentSpeed += 1;  // accelerate the car by gaining 1 kmh per 10 milisecs
                so it will go 0 to 50kmh in 5 secs
26:         maxDistanceToMove = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME
                ;
27:     }
28:     if( (currentSpeed + driverSpeedFactor*15) > 15) {
29:         maxDistanceToMove += driverSpeedFactor * (15 / 3.6 * TrafficSimulator.
                CAR_RESPONSE_TIME);
30:         distanceInTwoSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME
                + driverSpeedFactor * (15 / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME);
31:         distanceInSixSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME
                + driverSpeedFactor * (15 / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME);
32:     } else {
33:         distanceInTwoSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME;
34:         distanceInSixSec = currentSpeed / 3.6 * TrafficSimulator.CAR_RESPONSE_TIME;
35:     }
36:     if(maxDistanceToMove < 0) maxDistanceToMove = 0;
37:     // look ahead
38:     if( (distanceAlongRoad + maxDistanceToMove) > roadLength - 8) { // at end of
            the road
39:         currentSpeed = 0;
40:         distanceAlongRoad = roadLength - 8; // stop car from coming onto
                intersection
41:     } else if(inFront!=null) {      // there is a car infront of you, take it into
            account
42:         // there are three situations here
43:         if(inFront.currentSpeed == 0) {     // car in front of you is stopped
44:             if( distanceAlongRoad + 2 >= inFront.distanceAlongRoad - 6 ) {  // car
                    is stopped and you should be too, give 2m stopping room
45:                 currentSpeed = 0;
46:             } else if( distanceAlongRoad + distanceInTwoSec >= inFront.
                    distanceAlongRoad - 6 ) {   // almost there brake a little harder
47:                 currentSpeed -= 2;
48:                 if(currentSpeed < 0) currentSpeed = 0;
49:                 distanceAlongRoad += currentSpeed / 3.6 * TrafficSimulator.
                        CAR_RESPONSE_TIME;
50:             } else if( distanceAlongRoad + distanceInSixSec >= inFront.
                    distanceAlongRoad - 6 ) {   // here you need to start braking
51:                 currentSpeed -= 1;
52:                 if(currentSpeed < 0) currentSpeed = 0;
```

```
53:                 distanceAlongRoad += currentSpeed  / 3.6 * TrafficSimulator.
                        CAR_RESPONSE_TIME;
54:             } else {
                    // driver is not affecting you
55:                 distanceAlongRoad += maxDistanceToMove;
56:             }
57:         }
58:         else if( distanceAlongRoad + distanceInTwoSec > inFront.distanceAlongRoad -
                6) {
59:             // car in front of you is travelling slower then you, and you are
                    within two seconds of it
60:             // see find out if there is room for you to pass. Only allow passing
                    every simulated three seconds
61:             if(TrafficSimulator.getSimulatedTime()%1000==0) {
62:                 if(onLane.isRight()) {
63:                     ListIterator<Lane> lanesptr = onRoad.rightLanes.listIterator();
64:                     while(lanesptr.hasNext()) {
65:                         Lane currentLane = lanesptr.next();
66:                         if(currentLane == onLane) continue;
67:                         // need to make sure lane is next to current one
68:                         if(onRoad.rightLanes.indexOf(onLane) - 1 == onRoad.
                                rightLanes.indexOf(currentLane) || onRoad.rightLanes.
                                indexOf(onLane) + 1 == onRoad.rightLanes.indexOf(
                                currentLane)) {
69:                             // The above if stops cars from jumping across something
                                    like three lanes at once
70:                             if(currentLane.isRoom(currentSpeed,distanceAlongRoad))
                                    {
71:                                 onLane = currentLane;
72:                                 break;
73:                             }
74:                         }
75:                     }
76:                 } else {
77:                     ListIterator<Lane> lanesptr = onRoad.leftLanes.listIterator();
78:                     while(lanesptr.hasNext()) {
79:                         Lane currentLane = lanesptr.next();
80:                         if(currentLane == onLane) continue;
81:                         if(onRoad.leftLanes.indexOf(onLane) - 1 == onRoad.leftLanes
                                .indexOf(currentLane) || onRoad.leftLanes.indexOf(
                                onLane) + 1 == onRoad.leftLanes.indexOf(currentLane)) {
82:                             if(currentLane.isRoom(currentSpeed,distanceAlongRoad))
                                    {
83:                                 onLane = currentLane;
84:                                 break;
85:                             }
86:                         }
87:                     }
88:                 }
89:             }
90:             if(onLane == inFront.onLane) {
91:                 if(currentSpeed > inFront.currentSpeed) {   // brake, lose two 2kmh
92:                     currentSpeed -=2;
93:                 }
94:                 if(currentSpeed < 0) currentSpeed = 0;
95:                 distanceAlongRoad += currentSpeed  / 3.6 * TrafficSimulator.
                        CAR_RESPONSE_TIME + driverSpeedFactor * (15 / 3.6 *
                        TrafficSimulator.CAR_RESPONSE_TIME);
96:             } else {
97:                 currentSpeed += 3;
98:                 distanceAlongRoad += currentSpeed  / 3.6 * TrafficSimulator.
                        CAR_RESPONSE_TIME + driverSpeedFactor * (15 / 3.6 *
                        TrafficSimulator.CAR_RESPONSE_TIME);
99:                 currentSpeed -= 3;
100:            }
101:
102:        } else { // car in front of you is not affecting you
103:            distanceAlongRoad += maxDistanceToMove;
104:        }
105:    } else if(distanceAlongRoad + distanceInSixSec > roadLength) {  // you are
            nearing the end of the road
106:        currentSpeed -= 1;
107:        if(currentSpeed < 0) currentSpeed = 0;
108:        distanceAlongRoad += currentSpeed  / 3.6 * TrafficSimulator.
```

```
                CAR_RESPONSE_TIME;
109:      } else {
110:          // always look to the left lanes ie keep left at all time
111:          if(TrafficSimulator.getSimulatedTime()%1000==0) {
112:              if(onLane.isRight()) {
113:                  ListIterator<Lane> lanesptr = onRoad.rightLanes.listIterator();
114:                  while(lanesptr.hasNext()) {
115:                      Lane currentLane = lanesptr.next();
116:                      if(currentLane == onLane) continue;
117:                      if(onRoad.rightLanes.indexOf(onLane) + 1 == onRoad.rightLanes.
                             indexOf(currentLane)) {
118:                      // The above if stops cars from jumping across something like
                             three lanes at once
119:                          if(currentLane.isRoom(currentSpeed,distanceAlongRoad)) {
120:                              onLane = currentLane;
121:                              break;
122:                          }
123:                      }
124:                  }
125:              } else {
126:                  ListIterator<Lane> lanesptr = onRoad.leftLanes.listIterator();
127:                  while(lanesptr.hasNext()) {
128:                      Lane currentLane = lanesptr.next();
129:                      if(currentLane == onLane) continue;
130:                      if(onRoad.leftLanes.indexOf(onLane) + 1 == onRoad.leftLanes.
                             indexOf(currentLane)) {
131:                          if(currentLane.isRoom(currentSpeed,distanceAlongRoad)) {
132:                              onLane = currentLane;
133:                              break;
134:                          }
135:                      }
136:                  }
137:              }
138:          }
139:          distanceAlongRoad += maxDistanceToMove;
140:      }
141:      if(currentSpeed<0) currentSpeed = 0;
142:      return;
143:  }
```

Car Controller Code V3

```
 1: public void moveOneStep(Car inFront) {
 2:     double timeConversionFactor = 3.6 / TrafficSimulator.CAR_RESPONSE_TIME;
 3:     int currentRoadLength = onRoad.getLength();
 4:     int currentSpeedLimit = onRoad.getSpeedLimit();
 5:     double distanceInOneSec = currentSpeed / timeConversionFactor * (1/
            TrafficSimulator.CAR_RESPONSE_TIME);
 6:     double distanceInTwoSec = distanceInOneSec * 2;
 7:     double distanceInSixSec = distanceInOneSec * 6;
 8:     // accelerate the car
 9:     if( (distanceAlongRoad + distanceInSixSec < currentRoadLength - 8) &&
10:         ( inFront==null || ( inFront!=null && distanceAlongRoad + distanceInTwoSec
                < inFront.distanceAlongRoad - 6) ) &&
11:         (currentSpeed < ( currentSpeedLimit + (int)(0.2 * currentSpeedLimit *
                driverSpeedFactor) ) ) ) {
12:         currentSpeed += 1 + (int)(0.2 * currentSpeed * driverSpeedFactor);
13:         distanceInOneSec = distanceInOneSec = currentSpeed / timeConversionFactor *
                (1/TrafficSimulator.CAR_RESPONSE_TIME);
14:         distanceInTwoSec = distanceInOneSec * 2;
15:         distanceInSixSec = distanceInOneSec * 6;
16:     }
17:     // handle any cars in front of you
18:     if(inFront!=null) {
19:         // is there a car one sec ahead
20:         boolean carOneSecAhead = false;
21:         if( distanceInOneSec + distanceAlongRoad > inFront.distanceAlongRoad )
                carOneSecAhead = true;
22:         // is there a car two secs a head
23:         boolean carTwoSecsAhead = false;
24:         if( distanceInTwoSec + distanceAlongRoad > inFront.distanceAlongRoad - 6 )
                carTwoSecsAhead = true;
25:         // is there a car six secs a head
26:         boolean carSixSecsAhead = false;
27:         if( distanceInSixSec + distanceAlongRoad > inFront.distanceAlongRoad - 6 )
                carSixSecsAhead = true;
28:         // if there is a car six seconds a head is it stopped or going slowly
29:         // if it is then you should start slowing to a stop
30:         if(inFront.currentSpeed < 15) {
31:             if( distanceAlongRoad + 2 >= inFront.distanceAlongRoad - 6 ) {
32:                 currentSpeed = 0;
33:             } else if(carTwoSecsAhead) {
34:                 currentSpeed -= 2;
35:             } else if(carSixSecsAhead) {
36:                 currentSpeed -= 1;
37:             }
38:         }
39:         // if there is a car two seconds ahead
40:         // first look to pass it
41:         // check for room on the right lanes, and then the left
42:         // if you can pass it do so, else slow down and give it room
43:         if(carTwoSecsAhead) {
44:             if(TrafficSimulator.getSimulatedTime()%1000==0 && currentSpeed > 20) {
45:                 if(onLane.isRight()) {
46:                     int laneNum = onRoad.rightLanes.indexOf(onLane);
47:                     if(laneNum > 0) {
48:                         Lane temp = onRoad.rightLanes.get(laneNum - 1);
49:                         if(temp.isRoom(currentSpeed,distanceAlongRoad)) onLane =
                                temp;
50:                     }
51:                     if(onLane == inFront.onLane) {
52:                         if(laneNum < onRoad.rightLanes.size() - 1 ) {
53:                             Lane temp = onRoad.rightLanes.get(laneNum + 1);
54:                             if(temp.isRoom(currentSpeed,distanceAlongRoad)) onLane
                                    = temp;
55:                         }
56:                     }
57:                 } else {
58:                     int laneNum = onRoad.leftLanes.indexOf(onLane);
59:                     if(laneNum > 0) {
60:                         Lane temp = onRoad.leftLanes.get(laneNum - 1);
61:                         if(temp.isRoom(currentSpeed,distanceAlongRoad)) onLane =
                                temp;
62:                     }
63:                     if(onLane == inFront.onLane) {
64:                         if(laneNum < onRoad.leftLanes.size() - 1 ) {
```

```
65:                             Lane temp = onRoad.leftLanes.get(laneNum + 1);
66:                             if(temp.isRoom(currentSpeed,distanceAlongRoad)) onLane
                                    = temp;
67:                         }
68:                     }
69:                 }
70:             }
71:             if(onLane==inFront.onLane) {
72:                 currentSpeed -= 1;
73:                 while(carOneSecAhead) {
74:                     if( (currentSpeed / timeConversionFactor * (1/TrafficSimulator.
                            CAR_RESPONSE_TIME)) + distanceAlongRoad > inFront.
                            distanceAlongRoad ) carOneSecAhead = true;
75:                     else {
76:                         carOneSecAhead = false;
77:                         break;
78:                     }
79:                     currentSpeed -= 1;
80:                 }
81:             } else {
82:                 currentSpeed += 3;
83:             }
84:         }
85:     }
86:     // if you are nearing the end of the road slow down
87:     if( distanceAlongRoad + distanceInSixSec > currentRoadLength - 8 ) {
88:         if( distanceAlongRoad >= currentRoadLength - 8 ) {
89:             currentSpeed = 0;
90:         } else if( distanceAlongRoad + distanceInOneSec > currentRoadLength ) {
91:             currentSpeed -= 3;
92:         } else if( distanceAlongRoad + distanceInTwoSec > currentRoadLength ) {
93:             currentSpeed -= 2;
94:         } else {
95:             currentSpeed -= 1;
96:         }
97:     }
98:     if( inFront==null || (inFront!=null && onLane == inFront.onLane) ) {
99:         if(TrafficSimulator.getSimulatedTime()%5000==0 && currentSpeed > 20) {
100:             if(onLane.isRight()) {
101:                 int laneNum = onRoad.rightLanes.indexOf(onLane);
102:                 if(laneNum < onRoad.rightLanes.size() - 1 ) {
103:                     Lane temp = onRoad.rightLanes.get(laneNum + 1);
104:                     if(temp.isRoomLeft(currentSpeed,distanceAlongRoad)) {
105:                         onLane = temp;
106:                         currentSpeed -= 3;
107:                     }
108:                 }
109:             } else {
110:                 int laneNum = onRoad.leftLanes.indexOf(onLane);
111:                 if(laneNum < onRoad.leftLanes.size() - 1 ) {
112:                     Lane temp = onRoad.leftLanes.get(laneNum + 1);
113:                     if(temp.isRoomLeft(currentSpeed,distanceAlongRoad)) {
114:                         onLane = temp;
115:                         currentSpeed -= 3;
116:                     }
117:                 }
118:             }
119:         }
120:
121:     }
122:     // move the car
123:     if(currentSpeed <0 ) currentSpeed = 0;
124:     distanceAlongRoad += currentSpeed / timeConversionFactor;
125:     return;
126: }
```

Car Controller Code V4

```
 1: public void moveOneStep(Car inFront) {
 2:     int currentRoadLength = onRoad.getLength();
 3:     int currentRoadStoppingPoint = currentRoadLength - 10;  // always stop 10
            meters short of the actual end of the road
 4:     // 10 meters = 2 * 4meters for the lanes + 2 meters for safe distance back
 5:     boolean carInFront = true;
 6:     if(inFront==null) {
 7:         carInFront = false;
 8:     }
 9:     boolean onRightLane = onLane.isRight();
10:     double timeConversionFactor = 3.6 / TrafficSimulator.CAR_RESPONSE_TIME;
11:     Intersection endOfRoadInt;
12:     if(onRightLane) {
13:         endOfRoadInt = onRoad.intersectionA;
14:     } else {
15:         endOfRoadInt = onRoad.intersectionB;
16:     }
17:     boolean useLights = true;
18:     if(endOfRoadInt.connectedRoads.size()<=2) useLights = false;
19:     // if (at end of road)
20:     // no car in front and within 5 meters of the stopping point
21:     if(distanceAlongRoad >= currentRoadStoppingPoint - 5 && !carInFront && (
            endOfRoadInt.greenRoad == endOfRoadInt.connectedRoads.indexOf(onRoad))) {
22:         Road newRoad;
23:         Lane newLane;
24:         if(onRightLane) {
25:             int numConnectedRoads = endOfRoadInt.connectedRoads.size();
26:             if(numConnectedRoads > 1) {
27:                 newRoad = endOfRoadInt.connectedRoads.get( randomNumbers.nextInt(
                        endOfRoadInt.connectedRoads.size() ) );
28:                 while(newRoad == onRoad) newRoad = endOfRoadInt.connectedRoads.get(
                        randomNumbers.nextInt( endOfRoadInt.connectedRoads.size() ) );
29:                 // if going from right lane to right lane
30:                 if( newRoad.intersectionB == endOfRoadInt ) {
31:                     // try and stay in the same lane num that you are currently in
32:                     try {
33:                         newLane = newRoad.rightLanes.get( onRoad.rightLanes.indexOf
                                (onLane) );
34:                     } catch (Exception e) {
35:                         // if that lane does not exist move into the left most lane
36:                         if(newRoad.rightLanes.size()!=0) {
37:                             newLane = newRoad.rightLanes.getLast();
38:                         // there are no lanes that you can access on this road
39:                         } else {
40:                             // look through other roads for possible lanes,
                                    otherwise uturn
41:                             // try and do a uturn
42:                             uturn();
43:                             return;
44:                         }
45:                     }
46:
47:                 // going from right lane to left lane
48:                 } else {
49:                     // try and stay in the same lane num that you are currently in
50:                     try {
51:                         newLane = newRoad.leftLanes.get( onRoad.rightLanes.indexOf(
                                onLane) );
52:                     } catch (Exception e) {
53:                         // if that lane does not exist move into the left most lane
54:                         if(newRoad.leftLanes.size()!=0) {
55:                             newLane = newRoad.leftLanes.getLast();
56:                         // there are no lanes that you can access on this road
57:                         } else {
58:                             // look through other roads for possible lanes,
                                    otherwise uturn
59:                             // try and do a uturn
60:                             uturn();
61:                             return;
62:                         }
63:                     }
64:
65:                 }
66:                 if(newLane.isRoom(8)) {
```

```
67:                     onLane = newLane;
68:                     onRoad = newRoad;
69:                     distanceAlongRoad = 8;
70:                     Collections.sort(TrafficSimulator.cars);
71:                 } else {
72:                     // there is no room on the new road
73:                     // just return without moving
74:                     return;
75:                 }
76:             } else {
77:                 // try and do a you turn
78:                 uturn();
79:                 return;
80:             }
81:         // left lanes
82:         } else {
83:             int numConnectedRoads = endOfRoadInt.connectedRoads.size();
84:             // there are other roads that you can go onto
85:             if(numConnectedRoads > 1) {
86:                 newRoad = endOfRoadInt.connectedRoads.get( randomNumbers.nextInt(
                        endOfRoadInt.connectedRoads.size() ) );
87:                 while(newRoad == onRoad) newRoad = endOfRoadInt.connectedRoads.get(
                        randomNumbers.nextInt( endOfRoadInt.connectedRoads.size() ) );
88:                 // if going from left lane to right lane
89:                 if( newRoad.intersectionB == endOfRoadInt ) {
90:                     // try and stay in the same lane num that you are currently in
91:                     try {
92:                         newLane = newRoad.rightLanes.get( onRoad.leftLanes.indexOf(
                                onLane) );
93:                     } catch (Exception e) {
94:                         // if that lane does not exist move into the left most lane
95:                         if(newRoad.rightLanes.size()!=0) {
96:                             newLane = newRoad.rightLanes.getLast();
97:                         // there are no lanes that you can access on this road
98:                         } else {
99:                             // look through other roads for possible lanes,
                                    otherwise uturn
100:                            // try and do a uturn
101:                            uturn();
102:                            return;
103:                        }
104:                    }
105:                // going from left lane to left lane
106:                } else {
107:                    // try and stay in the same lane num that you are currently in
108:                    try {
109:                        newLane = newRoad.leftLanes.get( onRoad.leftLanes.indexOf(
                                onLane) );
110:                    } catch (Exception e) {
111:                        // if that lane does not exist move into the left most lane
112:                        if(newRoad.leftLanes.size()!=0) {
113:                            newLane = newRoad.leftLanes.getLast();
114:                        // there are no lanes that you can access on this road
115:                        } else {
116:                            // look through other roads for possible lanes,
                                    otherwise uturn
117:                            // try and do a uturn
118:                            uturn();
119:                            return;
120:                        }
121:                    }
122:                }
123:                if(newLane.isRoom(8)) {
124:                    onLane = newLane;
125:                    onRoad = newRoad;
126:                    distanceAlongRoad = 8;
127:                    Collections.sort(TrafficSimulator.cars);
128:                } else {
129:                    // there is no room on the new road
130:                    // just return without moving
131:                    return;
132:                }
133:            } else {
134:                // try and do a you turn
```

```
135:                uturn();
136:                return;
137:            }
138:        }
139:    // you are still driving along the road
140:    } else {
141:        int currentSpeedLimit = onRoad.getSpeedLimit();
142:        double distanceInOneSec = currentSpeed / timeConversionFactor * (1/
                TrafficSimulator.CAR_RESPONSE_TIME);
143:        double distanceInTwoSec = distanceInOneSec * 2;
144:        double distanceInSixSec = distanceInOneSec * 6;
145:        // handle any cars in front of you
146:        if(carInFront) {
147:            // is there a car one sec ahead
148:            boolean carOneSecAhead = false;
149:            if( distanceInOneSec + distanceAlongRoad > inFront.distanceAlongRoad )
                    carOneSecAhead = true;
150:            // is there a car two secs a head
151:            boolean carTwoSecsAhead = false;
152:            if( distanceInTwoSec + distanceAlongRoad > inFront.distanceAlongRoad -
                    6 ) carTwoSecsAhead = true;
153:            // is there a car six secs a head
154:            boolean carSixSecsAhead = false;
155:            if( distanceInSixSec + distanceAlongRoad > inFront.distanceAlongRoad -
                    6 ) carSixSecsAhead = true;
156:            // if there is a car six seconds a head is it stopped or going slowly
157:            // if it is then you should start slowing to a stop
158:            if(inFront.currentSpeed < 15) {
159:                // if you are within two meters of it ensure that your car is
                        stopped
160:                if( distanceAlongRoad + 2 >= inFront.distanceAlongRoad - 6 ) {
161:                    currentSpeed = 0;
162:                // if the car is two seconds ahead brake quite hard
163:                } else if(carTwoSecsAhead) {
164:                    currentSpeed -= 2;
165:                // if the car is six seconds ahead break softly
166:                } else if(carSixSecsAhead) {
167:                    currentSpeed -= 1;
168:                }
169:            }
170:            // if there is a car two seconds ahead
171:            // first look to pass it
172:            // check for room on the right lanes, and then the left
173:            // if you can pass it do so, else slow down and give it room
174:            else if(carTwoSecsAhead) {
175:                if(TrafficSimulator.getSimulatedTime()%1000==0 && currentSpeed > 15
                        ) {
176:                    if(onLane.isRight()) {
177:                        int laneNum = onRoad.rightLanes.indexOf(onLane);
178:                        if(laneNum > 0) {
179:                            Lane temp = onRoad.rightLanes.get(laneNum - 1);
180:                            if(temp.isRoom(currentSpeed,distanceAlongRoad)) onLane
                                    = temp;
181:                        }
182:                        if(onLane == inFront.onLane) {
183:                            if(laneNum < onRoad.rightLanes.size() - 1 ) {
184:                                Lane temp = onRoad.rightLanes.get(laneNum + 1);
185:                                if(temp.isRoom(currentSpeed,distanceAlongRoad))
                                        onLane = temp;
186:                            }
187:                        }
188:                    } else {
189:                        int laneNum = onRoad.leftLanes.indexOf(onLane);
190:                        if(laneNum > 0) {
191:                            Lane temp = onRoad.leftLanes.get(laneNum - 1);
192:                            if(temp.isRoom(currentSpeed,distanceAlongRoad)) onLane
                                    = temp;
193:                        }
194:                        if(onLane == inFront.onLane) {
195:                            if(laneNum < onRoad.leftLanes.size() - 1 ) {
196:                                Lane temp = onRoad.leftLanes.get(laneNum + 1);
197:                                if(temp.isRoom(currentSpeed,distanceAlongRoad))
                                        onLane = temp;
198:                            }
```

```
199:                         }
200:                     }
201:                 }
202:                 if(onLane==inFront.onLane) {
203:                     currentSpeed -= 1;
204:                     while(carOneSecAhead) {
205:                         if( (currentSpeed / timeConversionFactor * (1/
                                TrafficSimulator.CAR_RESPONSE_TIME)) +
                                distanceAlongRoad > inFront.distanceAlongRoad )
                                carOneSecAhead = true;
206:                         else {
207:                             carOneSecAhead = false;
208:                             break;
209:                         }
210:                         currentSpeed -= 1;
211:                     }
212:                 } else {
213:                     currentSpeed += 3;
214:                 }
215:             }
216:         }
217:         // slow down if approaching the end of the road
218:         // if you are nearing the end of the road slow down, when you are
              approaching the end only stop if light
219:         // is red
220:         else if( distanceAlongRoad + distanceInSixSec > currentRoadStoppingPoint )
                 {
221:             if( distanceAlongRoad >= currentRoadStoppingPoint && endOfRoadInt.
                    greenRoad != endOfRoadInt.connectedRoads.indexOf(onRoad)) {
222:                 currentSpeed = 0;
223:                 distanceAlongRoad = currentRoadStoppingPoint;
224:             } else if( distanceAlongRoad + distanceInOneSec > currentRoadLength &&
                    endOfRoadInt.greenRoad != endOfRoadInt.connectedRoads.indexOf(
                    onRoad)) {
225:                 currentSpeed -= 3;
226:             } else if( distanceAlongRoad + distanceInTwoSec > currentRoadLength &&
                    endOfRoadInt.greenRoad != endOfRoadInt.connectedRoads.indexOf(
                    onRoad) ) {
227:                 currentSpeed -= 2;
228:             } else if(distanceAlongRoad + distanceInTwoSec > currentRoadLength) {
229:                 // do nothing
230:             } else {
231:                 currentSpeed -= 1;
232:             }
233:         }
234:         // otherwise accelerate up to the speed limit
235:         // accelerate the car
236:         if( (distanceAlongRoad + distanceInSixSec < currentRoadStoppingPoint) &&
237:             ( inFront==null || ( inFront!=null && distanceAlongRoad +
                    distanceInTwoSec < inFront.distanceAlongRoad - 6) ) &&
238:             (currentSpeed < ( currentSpeedLimit + (int)(0.15 * currentSpeedLimit *
                    driverSpeedFactor) ) ) ) {
239:             currentSpeed += 1 + (int)(0.15 * driverSpeedFactor);
240:             distanceInOneSec = distanceInOneSec = currentSpeed /
                    timeConversionFactor * (1/TrafficSimulator.CAR_RESPONSE_TIME);
241:             distanceInTwoSec = distanceInOneSec * 2;
242:             distanceInSixSec = distanceInOneSec * 6;
243:         }
244:     }
245:     if(currentSpeed <0 ) currentSpeed = 0;
246:     distanceAlongRoad += currentSpeed / timeConversionFactor;
247:     return;
248: }
249:
250: public void uturn() {
251:     Lane newLane = null;
252:     if(onLane.isRight()) {  // look to u turn to a left lane
253:         if(onRoad.leftLanes.size()!=0) {
254:             try {
255:                 newLane = onRoad.leftLanes.get( onRoad.rightLanes.indexOf(onLane) )
                        ;
256:             } catch ( Exception e) {
257:                 newLane = onRoad.leftLanes.getFirst();
258:             }
```

```
259:         }
260:     } else {                // look to u turn to a right lane
261:         if(onRoad.rightLanes.size()!=0) {
262:             try {
263:                 newLane = onRoad.rightLanes.get( onRoad.leftLanes.indexOf(onLane) )
                        ;
264:             } catch ( Exception e) {
265:                 newLane = onRoad.rightLanes.getFirst();
266:             }
267:         }
268:     }
269:     if(newLane==null) return;
270:     if(newLane.isRoom(8)) {
271:         onLane = newLane;
272:         distanceAlongRoad = 8;
273:         Collections.sort(TrafficSimulator.cars); // because you have changed the
                order of the cars
274:                                                 // by moving lanes, resort
                                                        everything.
275:     }
276:     return;
277: }
```

References

Note: some references and links are inline and are not indexed here.

1. http://www.nzherald.co.nz/topic/story.cfm?c_id=348&objectid=10450836
2. http://en.wikipedia.org/wiki/Graph_%28data_structure%29

Brief Glossary

AI – Artificial Intelligence

The branch of computer science that deal with writing computer programs that can solve problems creatively.

GUI – Graphical User Interface

A user interface based on graphics (icons and pictures and menus) instead of text; uses a mouse as well as a keyboard as an input device

OOP – Object Orientated Programming