To add: - eclipse uml diagram
- testing document
- instagantt charts

----- = -----

After the build, the “Parameters” and “Run Options” tabs have user defined options. Initialize; GIS display will appear, run.

## 1. Introduction Describe the context for the work and the problem you are addressing. Briefly summarise what you achieved in the project.

This work was first and foremost an opportunity for working in a group. Employers want engineers who are not only skilled in their craft, but also persons who are able to collaborate, work well as part of a team, and invest in project aims and those working to achieve them. Our team was able to develop a simulation that involved each member in the process. In spite of differing abilities and communication styles, each person was able to contribute in important ways.

The software engineering part of this project focused on creating a traffic simulation software where we can test various traffic management strategies. We wanted to create a simulation that would be visually interesting with user modifiable variables, and that would have potential for further expansion and use.

We were able to create a simulation that is interesting, flexible, and allows different layouts and road policies to be tested and visually represented. Any geographic information system (GIS) map shapefile representing road networks can be loaded into our simulator and could easily be extended to take into account any other relevant attributes.

Our team further gained a more practical understanding of agent based modelling (ABM) through using Repast Simphony, and gained experience with version control through using GitHub.

## 2. Review Describe related work. Background and Impetus.

### Traffic flow forecasts

Analysis of complex urban networks and traffic flow models is the groundwork for reliable traffic flow forecasts, which are widely used to avoid traffic congestion and maximise road network capability in metropolitan areas.[11k] Predictions of traffic growth published by the UK Department of Transport in 2013 show that despite a slowdown in the past decade mainly due to economic recession and high oil prices, traffic on all roads in England are expected to grow by about 45% by 2040 [10j]. [jpg: England traffic, 10j]

Consequently, causes and effects of traffic congestion have been studied extensively in order to face the increasing road network saturation. Several factors that have significant impact on traffic flow have been identified, including the following:

- Road/Traffic policy approaches
- Timing/Cost of journeys
- Types of vehicles and speed
- Junctions and bottlenecks
- Lane splitting and joining
- High occupancy vehicle lanes
- Driver's behaviour
- Rush hours
- Occasional factors such as car accidents, road works or bad weather

The model we implemented has been designed with acceptable trade-off between accuracy and computational complexity. We have therefore selected the following set of urban traffic factors:

- Different types of vehicles, e.g. trucks are slower than cars and are less likely to overtake
- Driver's behaviour, e.g. reckless drivers do not stop at amber signal
- Journey of vehicles is based on best route according to roads' length and speed limit
- Roads can have multiple lanes, fast vehicles try to overtake slow ones
- Different traffic management policies, including traffic lights and/or give way signs

Data extracted from simulations will be evaluated at the end of this report, in order to analyse different traffic policies highlighting their pros and cons.

----

Our team began by researching various traffic simulation models and code available online. Some of the more significant included.

[1a] gave us a picture of what an interesting simulation and its code might look like. [2b] offered a good description of what coding with cellular automata involves for those on the team unfamiliar with the concept.

[3c] show one way to conceptualize moving cars along a road from origin to destination.

[4d] Detailed example of an agent-based traffic simulation; referenced for approach and features often considered when creating an agent based model. Well written and fairly comprehensive.

[5e] argues that "only agent-based models can explicitly incorporate the complexity arising from individual behaviors and interactions that exist in the real-world."[5e 1458] Shows the wide uses of Repast in various fields [5e 1457 Table 1]. The article notes that agent modelling using desktop environments like Repast Simphony is a good way to explore the potential of ABM in a brief space of time and with minimal training investment [5e 1464] and this is a benefit our team hoped to gain through this project.

[6f] provided many ideas regarding behaviour rules that was useful, as was the brief but descriptive discussion of modelling.

[7g] The entire team went through a very good tutorial to learn Simphony basics. This tutorial enabled us to dive into Simphony with code that has practical

[8h] used by Yoann as reference when coding lane changing and gap acceptance logic.

Selected Bibliography [do not renumber, these are throughout the paper]:

[1a] https://optalk2011.wordpress.com/traffic-simulator1/

[2b] http://natureofcode.com/book/chapter-7-cellular-automata/

[3c] http://crimesim.blogspot.it/2008/05/using-Repast-to-move-agents-along-road.html

[4d] http://www.myhomezone.co.uk/project/Report.htm

[5e] http://www.informs-sim.org/wsc11papers/130.pdf

[6f] http://www.umc.edu.dz/vf/images/misc/session4A/34-4A-paper2-Benhamza-SERIDI%20Hamid.pdf

[7g] http://repast.sourceforge.net/docs/RepastJavaGettingStarted.pdf

[8h] https://dspace.mit.edu/bitstream/handle/1721.1/34607/71301166.pdf?sequence=1

[9i] https://github.com/geotools/geotools/blob/master/modules/unsupported/process-raster/src/main/java/org/geotools/process/raster/GridConvergenceAngleCalc.java

[10j] https://www.gov.uk/government/publications/road-transport-forecasts-2013

[11k] https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/230433/an-introduction-to-dft-congestion-statistics.pdf

[13n] https://dspace.mit.edu/bitstream/handle/1721.1/34607/71301166.pdf

## 3. Requirements and design Describe the requirements you set for your project at the beginning and the design you have taken for your project. Focus on why you decided to tackle the problem in the way you did, and what effects that had on the design. You may also wish to mention the impact of team-working on your requirements and design.

The traffic simulator object of this report – the SignalGreen Traffic Simulator – is an agent-based, non-deterministic, discrete-time simulator for microscopic traffic modelling which allows to test different traffic policies on user-defined Geographic Information System (GIS) maps.

An Agent-Based modelling (ABM) simulation seemed most appropriate for a traffic simulation[5e, see 1465]. ABM uses entities (agents) to embed certain behaviours and they exist in a certain environment. Agents are autonomous, have objectives and goals, and can interact with other agents. ABM is particularly suitable for traffic simulation because agents, such as vehicles and traffic lights, can adapt to the evolving environment to accommodate changes, e.g. a vehicle needs to accelerate when a traffic light turns green.

The system is non-deterministic, with future states of traffic flow depending on unpredictable events, just as real-life traffic phenomena always have a degree of unpredictability. Randomness is added by the use of a random seed which changes during each run: with same input, output is always different.

A discrete time model, given its iterative nature, is convenient for traffic simulations. Every tick, the current state of the agents is evaluated, algorithms are recomputed and the overall state of the system is updated. For example, traffic lights change light every n ticks, and vehicle's velocity can be updated accordingly. Our simulation updates state every tick of the clock.

Microscopic traffic models typically simulate the behaviour of single vehicles using microscopic properties such as the vehicle's position and velocity. Our simulation uses the following common decision models[13n]: Car-following model, Lane-selection model, and Gap-acceptance model.

Lastly, GIS integration allows real maps to be loaded into the system. GIS is the de-facto standard for most professional traffic simulators, and governments regularly publish GIS maps about road networks, which are freely available for download. Virtually any GIS map can be loaded into our system.

<u>Milestone 1:</u>
- deadline: 23 February
- code compiles and runs simple simulation
- vehicles run on a map
- variable number of vehicles
- vehicles make decisions to reach a goal

<u>Milestone 2:</u>
- deadline: 8 March (changed to 16 March)
- implement basic vehicle type: car
- traffic flows bi-directionally
- junctions: lights, give way, roundabouts
- multiple maps in GIS standard
- cars have different behaviours: timid, aggressive, patient
- multiple lanes on some roads

<u>Milestone 3:</u>
- deadline: 16 March
- vehicles appear visually different by some criteria (type/behaviour, speed, congestion)
- speed limits on roads
- implement vehicle types: lorry, emergency, motorcycles
- vehicles exhibit passing behaviours

<u>Possible Traffic Policy Implementations:</u>
- all light junctions
- all give way junctions
- mix lights/give way junctions
- variable speed limits
- lights vs signals with varying levels of congestion

Language and Framework:

The team began by choosing a programming language for development. We quickly settled on Java as our language, as everyone on the team was familiar with it and Java is commonly used in industry. Java is fast enough to simulate hundreds of agents, is entirely object oriented which makes it ideal for an agent paradigm, and currently the most popular Agent Based Modelling framework is in Java. C++ was considered, but the team preferred Java for its ease of coding. A scripting language like python would have enabled fast development, but team members were unfamiliar with it, it is computationally slow, and so not a good choice for modelling with many objects.

For a framework we considered several options. We could code an entire agent based traffic simulation ourselves; this was not selected due to the enormous coding task this would involve, and we were concerned we would not meet all the requirements for the assignment in the time allocated. We considered NetLogo, but decided it did too much work for us; we would do almost no coding ourselves, and very little logic to create. We wanted to show our own ability to code and create our own logic.

Repast Simphony:

We settled on Repast Simphony, a general application programming interface (API) providing us tools, libraries, and plug-ins to create a simulation while allowing us to code and implement appropriate logic for the desired simulation. Repast comes with a BSD software license and is widely used in many fields of academia. [5e, table 1, 1457] Repast Simphony provided the following benefits that made it a good fit for our group project:

- As a framework for Java, it is fully object oriented and everything is created as a Plain Old Java Object. It includes Java libraries and .jar utilities, and all objects are rewritable.

- It provides an ABM environment with a practical skeleton of agents and their contexts, and includes features such as behaviour activation, a discrete event scheduler and Watcher component, and space and grid management.

- It provides a flexible plug-in framework, allowing the developers to use, modify, or write plug-ins. It provides the ability to integrate advanced elements into our code such as importing GIS maps, multi-threading, XML configuration files, and other advanced features.

- It includes charting and data collection functions, so we can focus on deciding on traffic policies and changing these parameters rather than spending time finding ways to display and record the results. The probe function is particularly useful for ABM testing.

- It provides an attractive display graphical user interface (GUI) we can modify to suit our needs and make a more interesting simulation, including user defined options at runtime and 2D and 3D visualisation and GIS plugins. One can create new or rewrite existing plug-ins if one wishes.

The benefits of Simphony come at the flipside cost of there being more opportunities for our team to display skill in modelling concepts and logic rather than a mastery of Java programming skill. When our team chose ABM as our design platform this meant behaviour and goals became more central than designing passive functions [4d], and we trust our ability to create an interesting system shows in our project and does credit to our ability. The team spent some time learning the framework, but this was considered a good investment for the benefits.

## 4. Implementation Describe the most significant implementation details, focussing on those where unusual or detailed solutions were required. Quote code fragments where necessary, but remember that the full source code will be included as an appendix.

### Simphony and Our Code:

The integrated development environment (IDE) for Simphony is a preconfigured Eclipse IDE. Eclipse Simphony varies from the normal Eclipse IDE in that it contains custom views, run configurations, and it adds three .jar files to the buildpath: JOGL (Java bindings for openGL), Java3D, and JAI (Java Advanced Imaging). These .jar files are used to create graphical representation of objects. Simphony also requires a ContextBuilder to initialize and drive the simulation, replacing the Java main run method.

While Simphony provided a very useful API, our team determined how each agent and behaviour should be modelled and the logic used to describe each object. For example, Simphony had no effect on how we modelled traffic signals. There is no agent superclass, no inheritance interfaces, no exposed code provided. The actual creation of agents and the method of that creation is the work of the programmers.

Our team wanted to especially take advantage of Simphony's visualisation benefits, but our core requirements are based around the logic of the agents, not their appearance. Simphony did provide nice helper features and many tools to increase the speed at which we could code. Our code uses Simphony's background foundation required for ABM, including multi-threading, the Clock/Watcher, having objects automatically drawn on the screen, and agents being aware of their environment.

### Agents and Environment:

The architecture for our simulation engine is a hierarchy of Java classes, where each class is an agent or a context. Agents are objects that exhibit behaviours, while a context is a collection of agents. Behaviour can be embedded through the agents reacting to the environment themselves, by using the watcher component which listens for events and triggers actions, or by globally scheduling events to happen.

Repast offers a ContextBuilder component containing a framework for creating agents and environments. Our agents included objects such as vehicles, junctions, and traffic signals. The environment is coded as a continuous space grid with an NxN matrix, in continuous space with (x,y) coordinates, on a directed graph network. It allows agents to be aware of their neighbourhood, including other agents.

The matrix allows the agents to know their real position and determine the distance between objects. Having a continuous space coordinates allows the user interfaced to display the agents in detail during the simulation. The network represents roads as edges connected by nodes as the intersections.

The team modelled entities and features including (but not limited to): vehicles, roads, junctions, road network topology, and GIS geography. We provided behaviour logic (direction, volition, velocity, etc) for the agents.

### GIS Maps and Direction of Motion:

Geographic Information system (GIS) maps were simplified by removing the road coordinates between the Junctions to make it more efficient. We have only one context which builds a road network and the GIS geography. The Road Network is a directed graph representing the road topology, and the GIS geography is used to display the simulation.

Any GIS map shapefile (.shp) representing road networks can be loaded into our simulator. The simulator uses some GIS attributes for modelling vehicle's behaviour (ex. road type, max speed, etc.) and could easily be extended to take into account any other relevant attribute.

The vehicle.java class determines the behaviour of vehicles so they follow roads. This required special coding, as GIS images point true north and use azimuth coordinates for its 3D mapping, while the UI points to grid north and uses simple X,Y coordinates for the grid. Azimuth coordinates allow one to measure distances and plot locations in a spherical coordinate system, and our code utilized this form of measurement in moving the vehicles along the maps. We referenced [9i] to see how one could compute the correct angle for displaying graphics on the UI. This further required the use of kinematics equations for plotting the direction of vehicles in several methods. Kinematics is the study of objects moving through space. Because we used GIS maps, all calculations along the map involve real distances.

## Context Builder

In Simphony no Main() method is required; instead a ContextBuilder is used to initialize the simulation. Class SignalGreenBuilder.java implements the ContextBuilder for the program.

## Path Abstraction

At the highest level of abstraction, the program contains a graph of vertices connected by paths. Agents are initialised with random source and destination vertices, the shortest path is calculated, and the agent travels there along the paths. When the destination is reached, new random values are chosen and the process repeats.

For performance reasons we did not find it feasible to recalculate the best path at each vertex; instead the entire path is chosen at initialisation using the highly optimised shortest path algorithm provided by Repast Simphony. The path is shortened as each junction is reached on the way to the destination.

## Vehicle Behaviour

A vehicle's vision is the distance around which a vehicle is aware of its environment and is calculated based on constant values DIST_VEHICLES and DIST_VEHICLES_STOPPED.

If a vehicle observes a slower moving vehicle in its path, it makes the decision to change lanes. The vehicle then checks the speed and distance of any vehicles in its path, slowing if needed to an optimal velocity. If no other vehicles are in its vision it will accelerate to its maxVelocity. The vehicle then checks for approaching junctions. If it views a red light then the vehicle decelerates to a stopped position before the light.

When a vehicle reaches a junction, the reference of the junction is removed from its path and the vehicle moves along the next road segment towards the next junction on its path. Once a vehicle reaches its destination, a new random destination and shortest route are calculated and the process repeats.

## Step Method: Diagram

[figure step_diagram]

Assume the current vehicle V is heading towards a junction J[i]. J[i] is the next node on the road network topology J[n], where n is the number of nodes, and i is the ith iteration of the simulation.

V needs to first decide which lane it wants to go to (lane selection decision model, step 1.2) and check that the road is clear (gap acceptance decision model, step 2).

In steps 3 and 4, V tries to either accelerate or slow down, depending on the outcome of the previous steps.

Next, if V is close enough to J[i], traffic management policies are evaluated (step 8): for example, if J[i] uses traffic jams, then V asks J[i] if it has to wait (Light.RED signal detected, step 8.1.1). Finally, displacement is executed, meaning V's position is updated both in the road network topology and in the GIS projection (step 9).

V checks what is the J[i] on the route J[n], because it knows which vehicles are approaching. This is done by holding a priority queue of incoming Vehicles for each road segment, where weights are the distance from V to J[i]. This is an efficient solution as insertion is done in O(log n), while removal is done in constant time.

From Repast's point of view, Step() method is executed every iteration of the simulation, using the Java annotation: @ScheduledMethod(start = 1, interval = 1) meaning that it starts from tick one, and gets executed every tick.

## The Vehicle Class

Class Vehicle.java is a very important class which initializes vehicles, assigns vehicles their individual behaviour, type, source, and interactions, and contains vehicle behaviours. This class contains the generic vehicle constructor. The class contains a list of junctions that create paths along the road graph.

Classes CarVehicle.java and TruckVehicle.java extend the Vehicle.java class and override the constructor to create three new types of vehicles: slow cars, fast cars, and trucks. Each type has its own adjusted maxVelocity and visual icon.

Method initVehicle() creates a type of vehicle and assigns it a random source and destination junction that includes the vehicle's route to that destination.

Method step() determines individual vehicle behaviours, including interaction with junctions and other vehicles, increasing/decreasing speed, changing lanes, and moving the vehicle along the map.

Method computeDisplacement() is used to determine exactly how far in metres the vehicle should travel based on the vehicle's attributes.

## Junction Classes

Class Junction.java creates junction objects that are nodes on the road network. These objects contain references to road segments (including adjacent junctions and roads) and queues (containing vehicles) associated with it. It is used throughout the program to determine where vehicles are, and so is fundamental to vehicle behaviour.

Class TrafficLight.java is an agent that extends Junction.java and schedules traffic light management. It has its own step method to iterate through lights, using class Light.java which encapsulates and toggles between the three light states. It also keeps junction states using a HashMap of junction and light objects.

Other Work

The only class which is not our team's work is class PriorityBlockingDequeue.java. This class is copyrighted by Aviad Ben Dov, and it implements a rather complicated data structure holding vehicles in a queue at junctions. We used his work because the code was complex enough that it would take a great deal of time to write ourselves, and we thought it prudent to use his code with attribution instead of risking not completing the junction code satisfactorily in time or relying too heavily on his work.

## 5. Testing Explain how you tested your software (e.g. unit testing) and the extent to which you tested it. If relevant to your project, explain performance issues and how you tackled them.

Testing will be done to confirm that our system conforms to functional requirements and expected behaviours. Exhaustive testing of the system is naturally not realistically possible in the time given, but we will perform basic unit, functional, error, cross-platform, and system testing.

ABM testing will be done by running the simulation, observing the agents' behaviour and comparing it with expected behaviour. We will be test our code at three different levels of functionality:

1) Agent behaviour: Verified by checking the changing environment variables around agents. This tests functional requirements of the system as well.
2) Runtime Parameters: Elements and data which load at runtime will be tested using valid and invalid inputs and checking for correct results.
3) Overall system behaviour: Checks that all agents of the system produce expected results for a given scenario.

This will include testing of agents interaction with other components of environment (roads, traffic lights) when programme will be executed at run time.

Automation testing will not be possible due to the nature of agent-based modelling simulations. Agents are non-deterministic and there is no set predictability of where objects will be at a given time. There is not a set result that can be expected for each vehicle, so automated testing for agents is not generally possible.

## 6. Team work Describe how you worked together, including the tools and processes you used to facilitate group work.

Our approach to the project was to set down ideas, goals, and tools in the first two weeks, then jump into the coding and get as far along as we could until the initial report came due. This worked well in many ways. Important decisions about implementation were decided very quickly so coding was able to start in January. The milestones were itemized early in the project, with Milestone 1 set early and items assigned to Milestones 2 and 3 differentiated closer to the deadline of the Initial Report.

Roles and Task Division:

| | |
|---|---|
| Yoann Strigini: | Design Architect, Developer #1 |
| | Design SignalGreen model, integration with GIS, develop Vehicle class, presentation layer, Reporting (architecture and technical details) |
| Adeela Saalim: | Testing Specialist |
| | Develop test cases, run test cases, document testing |
| James Kerr: | Documentation Specialist #2, LaTeX Specialist |
| | LaTeX for documentation, writing technical documentation for report |
| Waqar Aziz: | Developer #2, Policy Comparison and Reporting |
| | develop code; primary traffic policy testing and reporting |
| Andrea Senf: | Team Leader, Tech Doc Specialist |
| | Coordinate team activities, Scrum Master, Reporting (Project vision, goals and activities, soft. req., PM details etc.) |

Tools:

- GitHub. GitHub was used for creating the code, providing version control and easy sharing of code. Initial and final reports were also kept there. Andrea held the main code repository, and the team branched off that code.

- Communication. For communication outside coding, the most used tool by far was WhatsApp; this was practical for team chat, answering basic questions, and coordinating meetings. Longer reports for the group were written on asana or sent by email. Asana was used consistently by team members and was helpful for the documentation of the final report and for automatically generated Gantt charts through Instagantt.

- Meetings. When one or more group members believed a whole group meeting was needed to further the project, this was communicated on WhatsApp and a time and place of meeting was arranged using input from all members. These meetings were in person. Other times two or three members met together using Skype or in person as needed to work on parts of the project pertaining only to them.

Development and Reporting:

The team set out to utilize an Agile method of development with SCRUM reports once a week, either in person (if we had a meeting) or written if we did not gather. Group SCRUM did not pan out exactly as anticipated, as early meetings had absences which made SCRUM impractical as we waited for other members arrived (which they occasionally did not). We then decided to transition completely to written reports.

Approximately every week the group coordinator would remind the team to post WhatsApp/asana updates so everyone was aware of what the others were doing and to help the team to continually progress. Team members were to request help as needed and there was always a visible “next task” waiting when one’s current task had been completed, so while SCRUM was not overtly performed the goals behind the method did remain.

Team Challenges:

Our team faced several challenges in working together and with our chosen tools. [Note: all members are here referred to as 'he'.]

- Our team struggled to coordinate regarding GitHub the first part of the project, as several members focused on coding on their own machines rather than branching from GitHub in the weeks prior to the initial report date. Initial code was uploaded to GitHub the day before the initial report was due. The members leading in coding later created a full GitHub project for everyone to access.

- A member missed two meetings where decisions were made, so his preferences were not implemented. As he had strong feelings about these decisions it was difficult for the person to accept and caused some friction.

- A member found he did not have the ability to run Simphony and so did not have a way to code his part of the project; the group was notified of this after the initial report presentation. He found a way to partially resolve this so he could code from the last week of February, although still without full Eclipse functionality.

- A member had difficulty prioritizing his code at the end, putting off resolving a key issue for several days. This resulted in our final Milestone deadline being pushed back almost a week and added stress to other team members.

- After difficulty getting team members to follow through with attending agreed meetings, we voted to modify our practice so that absence to agreed full group meetings would result in a point for that person being deducted.

## 7. Evaluation Critically evaluate your project: what worked well, and what didn't? How did you do relative to your plan? What changes were the result of improved thinking and what changes were forced upon you? Note that you need to show that you understand the weaknesses in your work as well as its strengths. You may wish to identify relevant future work that could be done on your project.

Group Project

University group projects can be tricky to make work well; professional teams generally have the benefits of prior working relationships and clearly understood authority and accountability structures. SignalGreen team members were generally unfamiliar with each other before forming the group; four of our team formed based on where we were sitting in the introductory lecture, and the fifth was a person who we heard was looking for a group.

Distributing points is generally difficult to manage well, as giving too much power to democratic vote can be seen as unfair and hurtful by some members, and not allowing enough flexibility can result in members not feeling impressed to participate or work in a timely fashion.

In general, as masters students we are committed to doing good work, as our work for the project is part of our training for our careers. On this basis our team decided to share points evenly, and points lost due to agreed criteria would be distributed by vote. Some members did more work than our points can reflect, and we trust their hard work will pay off in personal dividends in the future.

Ultimately, the goal of the team was for everyone to get through the module with a pass (preferably much better). To this end, we are not group critiquing each other, nor are we calculating what we ourselves did for comparison to others. As we believe our team has been successful in the project we feel it would be better to end with a feeling of satisfaction for having completed the project successfully.

## Milestone Results

Milestone 1 was completed in its entirety by the deadline set by the team. At that time we extended the deadlines of the Milestones to have all coding finished by 16 March, and then turn to cleaning the code and final commenting while testing was finishing and traffic policies were compared during the week of 16 March.

All Milestone 2 goals were completed except for implementing roundabouts. The give way implementation took longer than anticipated to complete, and was finally finished on 20 March.

Milestone 3 optional criteria were all completed by 16 March.

[insert Instagantt charts]

We completed tasks very much on schedule with what we planned at the beginning. In the interest of better code, we extended the Milestone 2 deadline to March 16; this allowed some desired behaviours to be completed. All coding was originally scheduled to stop 16 March to leave time for continued testing, commenting and code cleaning, and documentation.

One member had issues with his code that put the team a week behind at the end. This was disappointing, but team members pulled together to complete the rest of the tasks on schedule and bring out our traffic simulation by the deadline.

## Final Characteristics

- vehicles run on a map
- multiple maps in GIS standard
- variable number of vehicles
- vehicles make decisions to reach a goal
- vehicles exhibit passing behaviours
- implement basic vehicle types: car, lorry
- cars have different behaviours: aggressive, patient
- vehicles appear visually different by type
- traffic flows bi-directionally
- junctions: signals, give way
- multiple lanes on some/all roads
- speed limits on roads

## Repast Simphony

Overall we think our use of Repast Simphony was successful for our project. It provided a foundation API and Eclipse configuration that was not directly involved with agents or behaviours. It allowed us to create attractive visuals in the simulation and analysis graphing. We spent little time debugging, and were able to focus on simulating traffic. Simphony provided many extras (XML persistent storage, installers, etc) that we did not use due to lack of need or lack of implementation time.

An unforseen disadvantage of Simphony arose in that it cannot be installed or used on KCL lab computers. One member of our group did not have access to Simphony at home or on campus (the group did not know this until after the initial presentation), and he was not able to find a way to use Simphony until almost March, and then not effectively. This greatly reduced his ability to be useful to the team in coding and held up work on a major segment of the code.

Originally we considered using a belief-desire-intention (BDI) model to code the agents' behaviour, but as the member intending to code this was unable to use Simphony, behaviours were instead coded into the agents themselves.

We originally intended to implement both signal and roundabout junctions, but the code for the signals and give way took much more work than anticipated, and so we decided not to implement roundabouts in this version of our code.

Further work:

"Validation data is usually macroscopic statistics such as flow rate, speeds and queue time, which can easily be compared with data from real traffic experiments." [4d]

- If we had more time we would have liked to download a road map from a town in England where we know road data and compare our model to actual roads. This would be a big step in validating our logic and making the code useful to others wishing to model traffic.

- Creating roundabouts was the next goal for our code, and this would make the code work for modelling most UK towns and cities. (Roundabouts are not generally used in the USA.)

- Further development can be done in displaying road closures for modelling construction or traffic accidents. This would make the model more realistic and useful in determining maintenance disruption.

- Implement source/sinks for allowing traffic to come on/off the map. This would allow traffic congestion to increase as more cars are fed into the simulation than are going out (and vice versa), and would allow modelling of road conditions created by end of workday, post-sport competition, etc.

- We did not make progress on the BDI as planned, so this was left out of the code. The code could be restructured to more neatly implement this if desired.

- Adding other vehicle types such as motorbikes or road features such as pedestrian crossings or high occupancy vehicle (HOV) lanes to model their effect on congestion and traffic flow.

7. Peer assessment In a simple table, allocate the 100 'points' you are given to each team member. Valid values range from 0 to 100 inclusive.

Waqar: **

James: **

Adeela: **

Andrea: **

Yoann: **