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

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.

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.

This term 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 global integration 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.

2. Review Describe related work.

[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.

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

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.

Milestone 1:
- 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

Milestone 2:
- deadline: 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

Milestone 3:
- 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

Possible Traffic Policy Implementations:
- all light junctions
- all give way junctions

- mix lights/give way junctions
- variable speed limits

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 enable fast development, but some team members were unfamiliar with it, it is computationally slow, and so not a good choice for modelling with many objects.

We wanted to create an agent based simulation, as it seemed most appropriate for a traffic simulation. [5e, see 1465]

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.

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. 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.

The architecture for our simulation is a hierarchy of Java classes, where each class is an agent or a context. 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. No main method is required; the program is built around a context builder.

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.

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.

Originally we anticipated 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.

Agents and Environment:

Our team used Traffic Simulation Theory. We used a discrete time model, where state is updated every tick of the clock.

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.

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 Simphony 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.

Architecture:

The overall architecture of the code is model-viewer-controller (MVC), with an additional mediator component that coordinates between the MVC components. The main module is the simulation engine, which is actually a collection of Java plugins. Simphony's core framework is a provided plugin which offers all the elements you would expect of an ABM program, including behaviour activation, a discrete event scheduler (the clock), space management, etc. It also provides 2D and 3D visualisation and GIS plugins. One can create new or rewrite existing plug-ins if one wishes.

The architecture of the simulation engine is a hierarchy of Java classes, with each class being either an agent or a context. Agents are objects that exhibit behaviours. A context is a collection of agents. Behaviour can be embedded in three ways. First, each agent can read its environment and react accordingly. Second, a Watcher component can listen for certain events and trigger actions to occur. Third, events can be globally scheduled to happen.

GIS Maps:

Global integration 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.

Testing:

ABM presents some challenges for testing. 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 and testing must be done visually. Automated testing will be done for….

[Adeela's report content summarized and report document attached]

----- = -----

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

Code Overview:

Under src > signalGreen:

- singalGreenBuilder.Java This is the contextBuilder file, replacing the usual "Run" method in Java. It initializes the traffic simulator, including loading map, assigning and mapping junction types, and creating lanes.

- vehicle.Java Contains all vehicle behaviour and decision making. Uses kinematics equations. Assigns vehicle properties such as velocity and displacement.

- utils.Java multiple lanes, returns junctions for behaviour;

- constants.Java Very boring.

- coordinateAgent.Java Very boring.

- gisAgent.Java GIS coordinates, projections

- junction.Java [Detail] Controls vehicle junction behaviour. A queue of vehicles is formed at each junction

- laneAgent.Java na

- light.Java light object for traffic light junctions; returns color of current signal

- priorityBlockingDeque.Java **** This object was written by someone else

- Road.Java extends gisAgent. Roads, speed limits, lengths

- trafficLight.Java subclass of junction.Java; changes the light color

5. 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 Milestones 2 and 3 differentiated closer to the deadline of the Initial Report.

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.

Team Challenges:

Our team faced several challenges in working together and with our chosen tools. [Note: all members are here referred to as 'he', and different paragraphs may refer to one or more persons.]

- 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.

- One team 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.

- One 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.

- 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.

Team Roles:

It quickly became apparent that Yoann and Waqar were skilled with writing code. They created the general framework for the project and got the initial simulation working by the first week of February. James took responsibility for a block of coding logic from Yoann, and Adeela worked with helping Waqar. Adeela identified that Waqar was making progress more quickly than she, so the

team agreed she would shift to creating test cases and code testing. Andrea focused specifically on the documentation and on coordination for the team. James transitioned to assisting Andrea with documentation.

6. 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.

Our team was able to create a simulation that involved each member in the process. In spite of differing abilities and communication styles, each person was able to contribute in unique ways.

Milestone 1 – all aspects implemented
- 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

Milestone 2:
- deadline: 8 March – extended to 16 March
- implement basic vehicle type: car – implemented
- traffic flows bi-directionally – implemented
- junctions: lights, give way, roundabouts – signals implemented
- multiple maps in GIS standard – implemented
- cars have different behaviours: timid, aggressive, patient – ??
- multiple lanes on some roads – implemented

Milestone 3:
- deadline: 16 March – all coding stopped by 16 March
- vehicles appear visually different by some criteria (type/behaviour, speed, congestion) – ??
- speed limits on roads – implemented
- implement vehicle types: lorry, emergency, motorcycles – not implemented
- vehicles exhibit passing behaviours – ??

[insert Instagantt charts] We completed tasks very much on schedule with what we planned at the beginning.

Repast Simphony:

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.

Roles and Task Division:

The division of coding labour did not work out as originally intended. One member focused solely on documentation and coordination, and this was a decision based on listening to others who have worked on projects before. We intended for the coding to be written for easier division of labour, so that from February we would be all be able to code separate parts of the system. However, Yoann and Waqar were skilled in coding and also very ambitious. We found they occasionally rapidly implemented work that others had been assigned, so Adeela suggested she be moved to testing, and Andrea began focusing solely on documentation and communication.

Development and Reporting:

Originally we intended 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. This part of the project did not pan out as anticipated; early meetings had absences, which made SCRUM impractical as we waited until other members arrived (which they occasionally did not).

Our actual method of development looked more like a Waterfall style of development. One member had industry experience with Waterfall, and with the short time frame of the module and meeting reality of this project Waterfall would probably have been an equally appropriate choice. In the end, our development looked closer to this model.

As of mid-February, the group coordinator began asking for WhatsApp/asana updates from the team approximately every week so everyone was aware of what the others were doing and to help remind the team to continually progress. Team members requested help as needed from each other 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.

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.

- 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.

- 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.

- 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: