Group 8-05

Category 8 Research Project

Traffic Optimisations of the Bukit Timah Road Region

Maximilian Ang 3I1 (18) Leader

Wan Runhao 3I1 (27)

Yang Ziyi 3P3 (23)

Hwa Chong Institution

(High School)

Contents

Abstract

This project aims to use mathematical modelling using a specific framework to simulate traffic of the Bukit Timah Road region, with the goal being to suggest several solutions to alleviate the congestion issue through constructing a model. The process of modelling will be done in two stages, data-collection and using a car-following framework. The first stage is where data in the form of the number of cars over time is collected through collaborating with the School Security Department. With major modifications to a pre-existing model, a model simulator with the Bukit Timah Road Region is compiled. The data collected will be put into the simulator coded via CoffeeScript and HTML5, obtaining a fairly accurate simulation of the traffic condition outside HCI at a specific timeframe, with slight deviation from empirical data. The simulator is then used to find the average waiting time for students. After obtaining such results, various modifications will be suggested and verified by the simulator to attempt to reduce the waiting time.

Introduction and Rationale

Traffic connects and embodies us as a society, making it an important aspect of modern transportation. For most students, transport is what every one of them experiences before entering school. When we look at the road outside Hwa Chong in the morning, especially just before reporting hours at 7.30 am, we can see that it is congested. This causes a great amount of delay when entering and leaving the school, which is not advisable for efficiency and sleep.

The rationale of this project is to improve that situation, by suggesting various feasible solutions to combat the traffic congestion issue. However, as of recently, little has been done on the research of the road situation outside the school and how we can improve it with various methods. Thus, we aim to create a traffic simulator to fill in that gap, by using the data collected from the security guardhouse and integrate that into the Intelligent Driver Model powering our simulator to predict traffic and suggest various improvements in the form of policies that can be made to reduce congestion.

Traffic simulation should be done via software to better figure out the various approaches we can take in combating the situation. With new advancements in mathematics, engineering and computing, simulation software programs are increasingly becoming faster, more powerful, more detail-oriented and more realistic (Mahmud et al, 2016). With the possibility of modelling traffic now, steps should be taken forward to better the traffic congestion of Bukit Timah Road.

Objectives

1. To generate statistics on congestions concerning Bukit Timah Road through 2 different sources of data
2. To provide and establish possible solutions to reduce the waiting time of students outside Hwa Chong Institution Gates 3 and 4

Research Questions

1. How do we generate a mathematical model that simulates the traffic condition of the roads outside of Hwa Chong Institution?
2. What is the average waiting time for students in the morning on the road outside of Hwa Chong Institution Gates 3 and 4 based on data generated from a model and obtained from a survey?
3. Which solutions can reduce the waiting time of students the most, based on the Web-based simulator created?

Fields of Math

1. Statistic
2. Numerical Analysis
3. Traffic Simulation

Terminology

Term	Definition
Macroscopic Traffic Flow Model	The macroscopic traffic flow model is a kind of mathematical traffic model that focuses on the relationships between traffic flow variables, such as density, flow, mean speed of a traffic stream, etc. Such models are originally derived by integrating them with microscopic traffic flow models and converting the single-entity level characteristics to comparable system-level characteristics.
Microscopic Traffic Flow Model	Microscopic traffic flow models are a type of mathematical model of vehicular traffic dynamics. Unlike macroscopic models, microscopic traffic flow models simulate vehicle-driver units singularly, so the dynamic variables of the models contain microscopic properties such as the position and velocity of single vehicles.
Intelligent Driver Model	Intelligent Driver Model (IDM) is a type of car-following model used to simulate freeway and urban traffic. It was developed by Treiber, Hennecke and Helbing in 2000 to improve upon results obtained from other types of microscopic traffic flow models.

Euler method	The Euler method is a first-order numerical procedure for solving ordinary differential equations with a proven initial value. It is one of the most basic methods used for the numerical integration of ordinary differential equations and is the simplest Runge–Kutta (RK) method, which is used by IDM.
Taylor series	In mathematics, the Taylor series of a function is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point. For most common functions, the function and the sum of its Taylor series are equal near this point.

Literature Review

Importance of Computation in Modelling

Traffic simulation refers to the mathematical modelling of transportation systems, such as freeway junctions, roundabouts, etc (D. Leonard II, Ph.D. 2008). This is done through the application of computer software to better help plan, construct, and run transportation systems. Computation is important in transportation because computational models are much faster than analytical or numerical treatment done by hand. Moreover, it can be used for experimental studies and reveal detailed relationships that would be lost in analytical or numerical treatment. Finally, it can produce visually sound representations of current and future scenarios, making it adaptable to many situations and allows it to be applied quickly to changing scenarios.

Macroscopic Models vs. Microscopic Models

To understand the methods of simulation, it is important to understand the concept of the system state, which is a set of variables that contains enough information to describe the evolution of the system over time (Sokolowski & Bank 2009). There are two main types of the system states, discrete or continuous. This will lead to the two main types of traffic simulation models, which are either macroscopic or microscopic. Macroscopic models are generally the traditional way of simulating traffic and are now largely outdated due to the lack of consideration of single-driver like units, but they are still important in understanding how the evolution of traffic modelling works. Nevertheless, we will be mainly focusing on microscopic models which are the current standard due to their higher degree of accuracy and predictability. In general, microscopic traffic flow models are classified according to state or space (Chapra & Canale, 2006).

Microscopic Traffic Flow Model

To better understand traffic simulation, we must go back to the origins of traffic modelling of the macroscopic traffic flow systems. Since the mid-1950s, research has been done to attempt to predict traffic. The original method by Leemis and Park (2006) of modelling traffic flow at the macroscopic level originated under an assumption that traffic streams as a whole are comparable to fluid streams. The first major step in macroscopic modelling of traffic was taken by Lighthill and Whitham (1955), when they indexed the comparability of 'traffic flow on long crowded roads' with 'flood movements in long rivers'. They proposed a functional relationship between flow and concentration for traffic on crowded arterial roads, from this a theory of the propagation of changes in traffic distribution along these roads may be deduced. The theory is applied to the problem of estimating how a 'hump', or region of increased concentration, will move along a crowded main road. A year later, P. I . Richards (1956) complemented the idea with the introduction of 'shock-waves on the highway', completing the LWR (Lighthill-Whitham-Richards) model. It suggested that traffic will move at a slightly slower than the mean vehicle speed and that vehicles passing through it will have to reduce speed rather suddenly at a 'shock wave' on entering it, but can increase speed again only very gradually as they leave it. The hump gradually spreads out along the road, and the time scale of this process is estimated. These pioneering studies of the attempt to predict traffic flow are what leads to the first steps in the process of predicting traffic.

Microscopic Traffic Flow Model

However, as time progresses, microscopic traffic flow models are developed and eventually replaced macroscopic models in the industry. In contrast to macroscopic traffic flow models, microscopic traffic flow models simulate single vehicle-driver units. This means that the dynamic variables of the models will represent microscopic properties, for example, the position and velocity of single vehicles. An example is the Intelligent driver model (IDM). The intelligent driver model is widely adopted in the research of Connected Vehicle (CV) and Connected and Autonomous Vehicle (CAV). IDM is a time-continuous car-following model for the simulation of the freeway and urban traffic (Martin et al., 2000), improved upon results provided with other "intelligent" driver models such as Gipps' model, which loses realistic properties in the deterministic limit (Spyropoulou & Ioanna, 2007). The two ordinary differential equations created by IDM are solved using Runge–Kutta methods (Kutta, 1901) of orders 1, 3, and 5 with the same time step, to show the effects of computational accuracy in the results. Surprisingly, IDM does not exhibit extremely unrealistic properties such as negative velocities and vehicles sharing the same space, even from a low order method such as with Euler's method (RK1). This means that it is more favourable (Wilson, R. E., 2001) than other car-following traffic modelling methods.

Conclusion

The importance of macroscopic models cannot be forgotten but we must also remember that microscopic models are now the industry norm due to them being superior to the latter. This resulted in the preference of such models when modelling traffic and ushered in the modern era of such fields. Therefore, we chose the Intelligent Driver's Model to act as the basis of our simulator as it fits best with our context and is the most reliable.

Methodology

The following is the methodology in which our project will undertake to achieve our goals.

1. Collect sufficient preliminary data about the traffic situation and numbers along with Bukit Timah Road Region.
    a. Done via CCTV footage provided by the school CCTV Department.
    b. Privacy concerns could arise but we promise to not share the footage including those car plate numbers with anyone else not in the project.
2. Experiment with various Microscopic Traffic Flow Models to find the most relevant and accurate. The Intelligent Driver Model will be used as it is the most optimum for our use case.
3. Find the average waiting time for students in the morning on the road outside of Hwa Chong Institution Gates 3 and 4, which is achieved from data obtained from a survey and generated from the model created.
4. Suggest solutions to the traffic congestion problem. Various suggestions will be suggested.
5. Create a Web-based computer programme to be more flexible and adaptable in other situations. We are aware of the amount of computational power not being enough, but CPU time on Google services could be bought at a reasonable rate to train/compute the model.

Results

Research Question 1

How do we generate a mathematical model that simulates the traffic condition of the roads outside of Hwa Chong Institution?

We have looked through the various traffic simulation models and have eventually decided on the Intelligent Driver's Model (IDM). The benefits of such microscopic traffic simulation models have been mentioned prior and it is the most fitting model for our usage. To generate a mathematical model digitally, we first need to employ the concept of the simulation model, find solutions to the equations generated and create the model using coding.

Concept of Intelligent Driver Model

On the technical side of things, IDM works in depth by describing the dynamics of the positions and velocities of a single-vehicle. For vehicles α, x_α denotes its position at time t and ν_α denotes its velocity. l_α, the length of the vehicle was also a variable in this model but was not considered due to the main focus being on private cars, and their length differences are negligible. This was also done to simplify the model to reduce the computing power required.

Afterwards, to simplify these variables into a simple notation, the net distance was defined as: $s_\alpha := x_{\alpha-1} - x_\alpha$,where $\alpha - 1$ refers to the vehicle directly in front of the other vehicle, α. The velocity difference between 2 cars (approaching rate) is defined as: $\Delta v_\alpha := v_\alpha - v_{\alpha-1}$

Therefore, two ordinary differential equations (Treiber et al. 2000) can be derived from these 2 notations above:

$$\dot{x}_\alpha = \frac{dx_\alpha}{dt} = v_\alpha$$

$$\dot{v}_\alpha = \frac{dv_\alpha}{dt} = \alpha \left(1 - \left(\frac{v_\alpha}{v_0} \right)^\delta - \left(\frac{s^* \left(v_\alpha, \Delta v_\alpha \right)}{s_\alpha} \right)^2 \right)$$

$$with\ s^* \left(v_\alpha, \Delta v_\alpha \right) = s_0 + v_\alpha T + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{a\ b}}$$

Where,

$v_0 =$ desired velocity,

$s_0 =$ minimum spacing,

$T =$ desired time headway,

$\alpha =$ acceleration,

$b =$ comfortable braking deceleration.

The exponent δ is set at 4, which is the general value used as it is the most accurate with the real-world data.

$\dot{x}_\alpha$ is defined as the derivative of x_α against time, giving us a velocity, given by a simple kinematic equation.

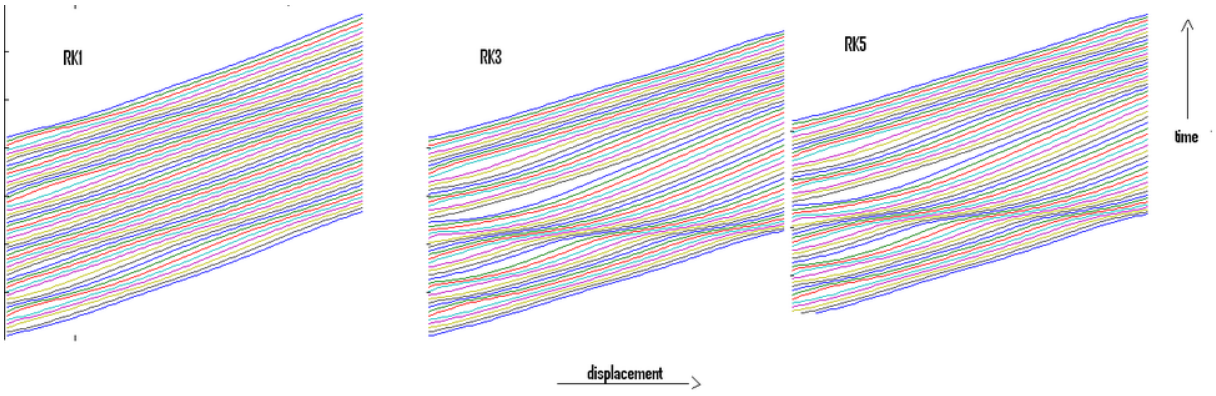
The acceleration of the car, $\dot{v}_\alpha$, is defined as the derivative of velocity of x_α against time and is also approximated by a constant α, scaled by another term. The desired velocity, v_0 minus

$\frac{v_\alpha}{v_0}$ represents the acceleration of a driver to its desired velocity that is defined by the velocity of the car in front of it, essentially $x_{\alpha-1}$. This term is scaled by an arbitrary constant δ to best approximate it to real world finding. Through empirical studies, the constant sigma is set at 4 to best follow real world scenarios (Malinauskas, 2014).

The difference between the actual velocity and the desired velocity is subtracted by the braking distance. The term $s\cdot\left(v_\alpha,\ \Delta v_\alpha\right)$ represents the braking distance. This is to ensure that mathematically, the cars do not "collide" with each other through trying to keep up. The term, $\sqrt{ab}$, is the acceleration multiplied by the comfortable braking distance, is also an arbitrary step to stay as close to real world scenarios as possible, and along with δ, could be modified to fit the specific road conditions.

Solutions to the Equations Generated

The 2 differential equations were solved using Runge-Kutta methods of orders 1, 3, 5 to account for computational accuracy in the results.



As we can see, the results do not have unrealistic properties that can happen with other driver models such as negative velocities or spacing irregularities. Even though traffic wave propagation is not as accurate in higher-order Runge-Kutta methods of 3 and 5, there are not many significant issues on the reliability of the model.

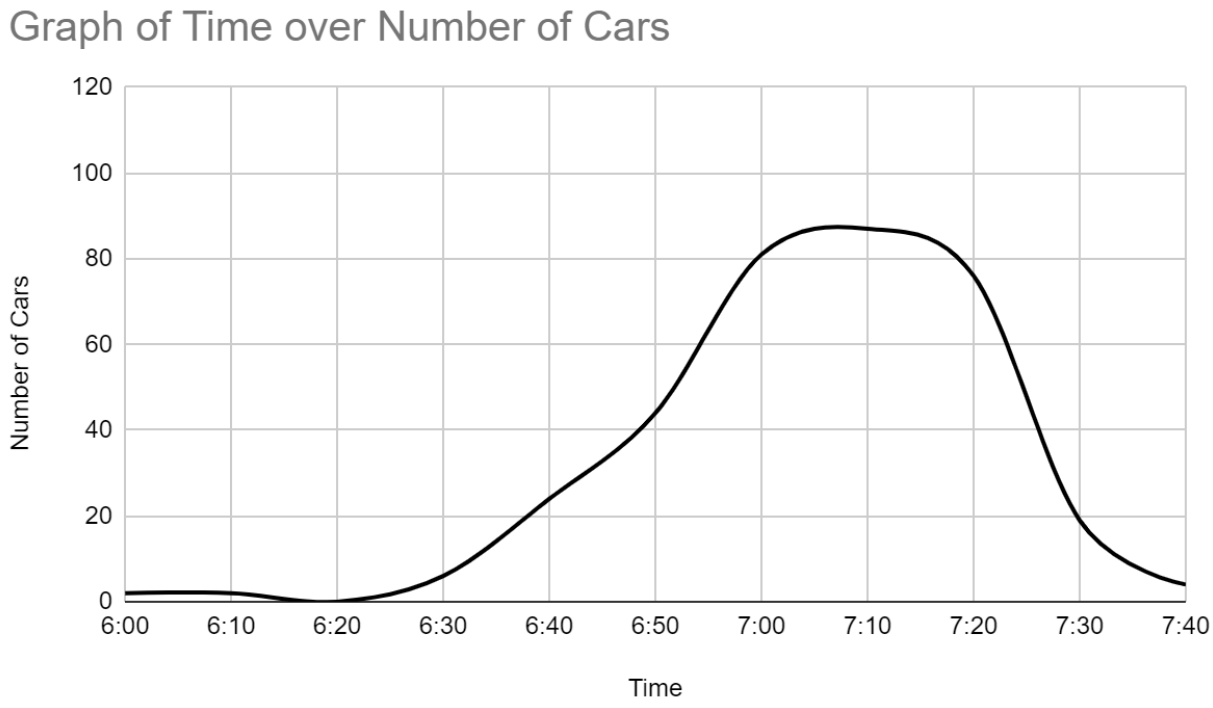
We can thus proceed to the next stage of employing the concept on the computers via coding using Coffeescript and HTML5, where the Runge-Kutta methods are hard coded into the simulator.

Creation of Computer Model

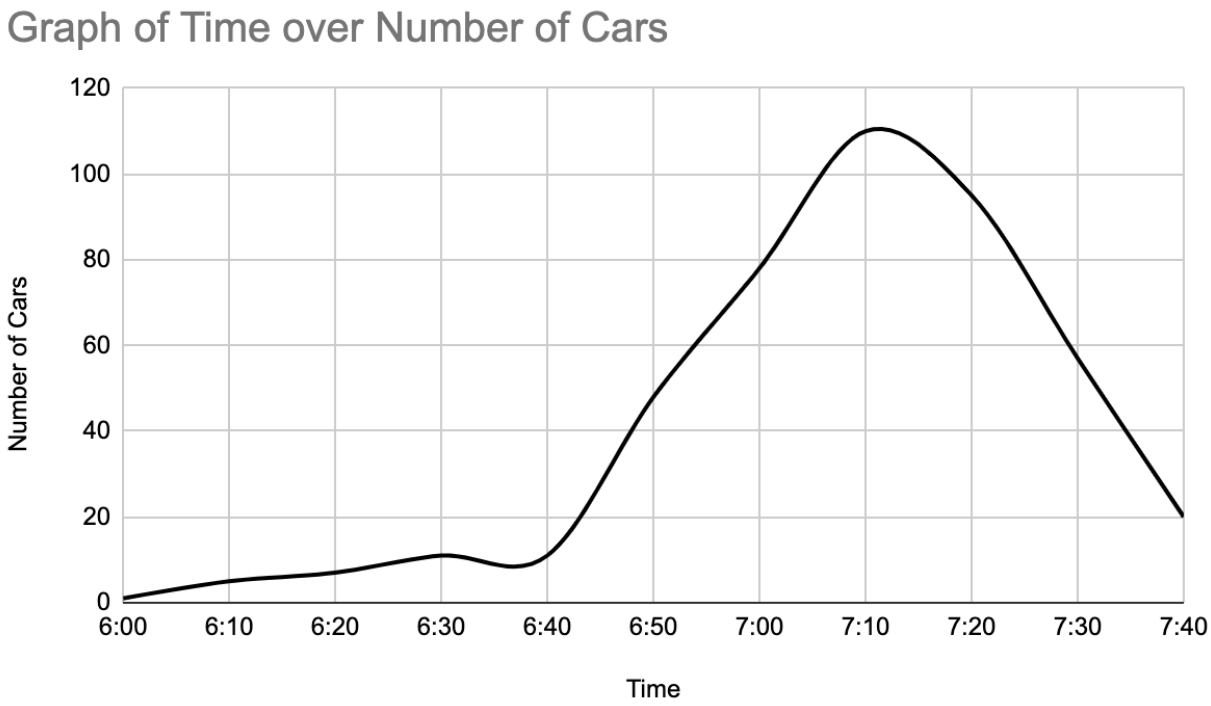
A GitHub source code, written in CoffeeScript and HTML5, was used and improved upon by us such that the simulator suited our needs, the full version of our code is appended below in the appendix section. Essentially, the Intelligent Driver Model was used to simulate the movement of cars inroads of similar shape to HCI premises.

Data to feed the simulator was collected by watching the school's CCTV footage. We chose to record the traffic data of 10 days which is a good balance between manpower required and sufficiency of data. After watching over 40 hours of footage over the period of 10 days, which was evenly spread across the year, the following graphs on the frequency of car entry is produced, which is in line with our model.

Gate 3:



Gate 4:



Sufficient preliminary data about the traffic along Bukit Timah Road Region was collected, and we can see how there is a spike of traffic in both Gate 3 and Gate 4 from 06:50 - 07:30. This is due to the surge of students entering the school, where the frequency of cars could reach 1 car per 6 seconds, which is the main contributor to traffic congestion in the school.

With that data obtained, we can input that into our simulator by changing the *number of cars* variable to simulate the traffic condition outside the school.

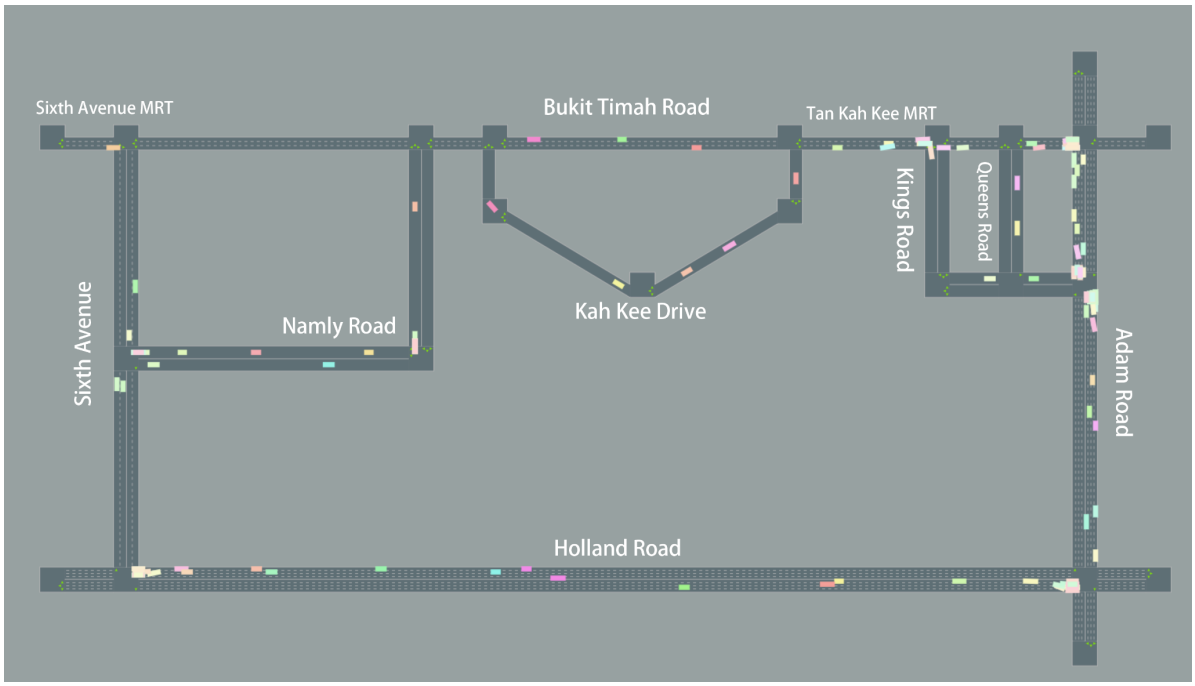
Research Question 2

What is the average waiting time for students in the morning on the road outside of Hwa Chong Institution Gates 3 and 4 based on data generated from a model and obtained from a survey?

Waiting time in our context is the time wasted waiting outside of the school, which is defined as "*time actually required"* **minus** *"time required",* where "time actually required" is the time actually taken to enter the school, and "time required" is the amount of time that is needed to enter the school without traffic. We used both our simulation data and survey collected to figure out the average waiting time.

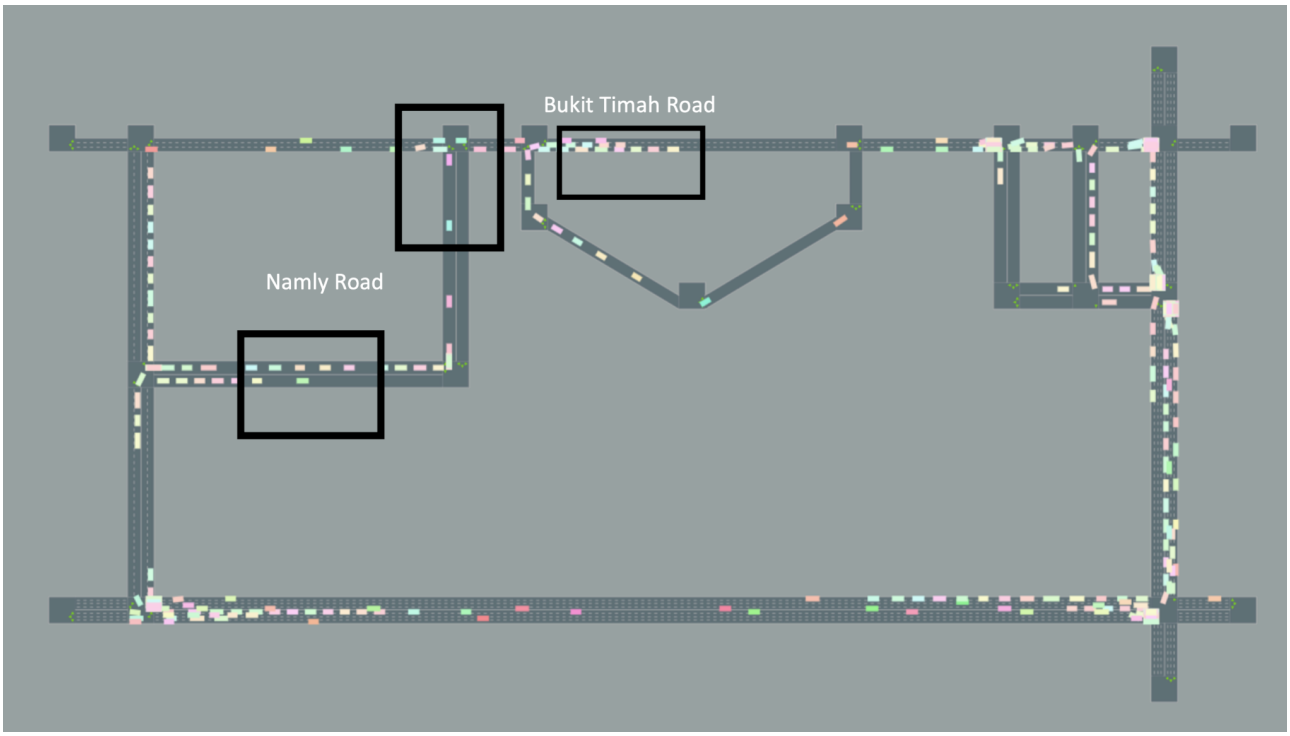
Simulation Data

From the data obtained previously, the model is able to run. This is a snapshot of the simulator, with the car number set at 100, which is the typical peak hour traffic flow rate.



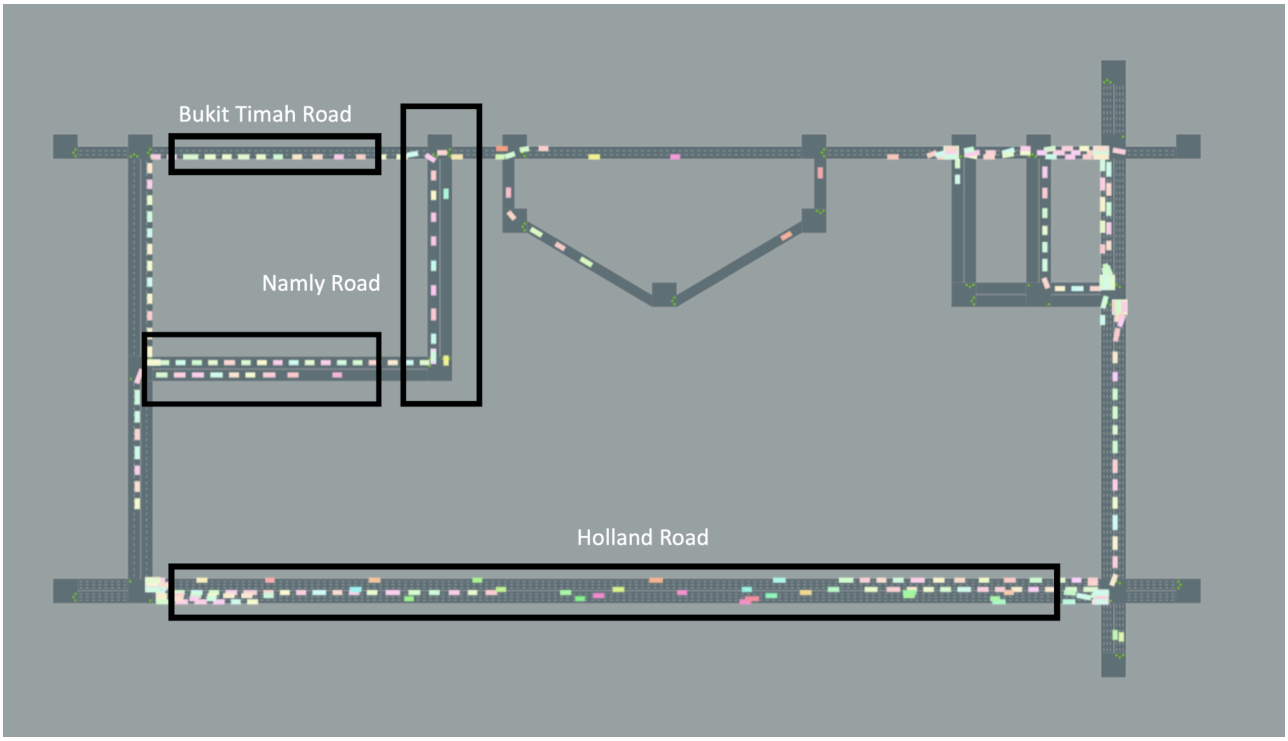
At this early stage of simulation, we can see that there are traces of regions of roads with high traffic flow but it is not a traffic jam yet. This is normally the morning data at around 06:30 hours, where it is the teacher's reporting time, but most students have not arrived yet.

After running for a few minutes and changing the car amount, we can clearly see congestions that are starting to form on these roads, namely Namly Road and Bukit Timah Road:



This is in line with our data, where traffic congestion mainly occurs on the road near Gate 3 and Gate 4, as seen by the leftmost and rightmost road in the simulator above during pre-peak hours at around 07:10 hours.

The situation gets worse when the maximum number of cars are imputed into the simulator which is as much as the peak hour, even hoarding the roads at Holland Road:



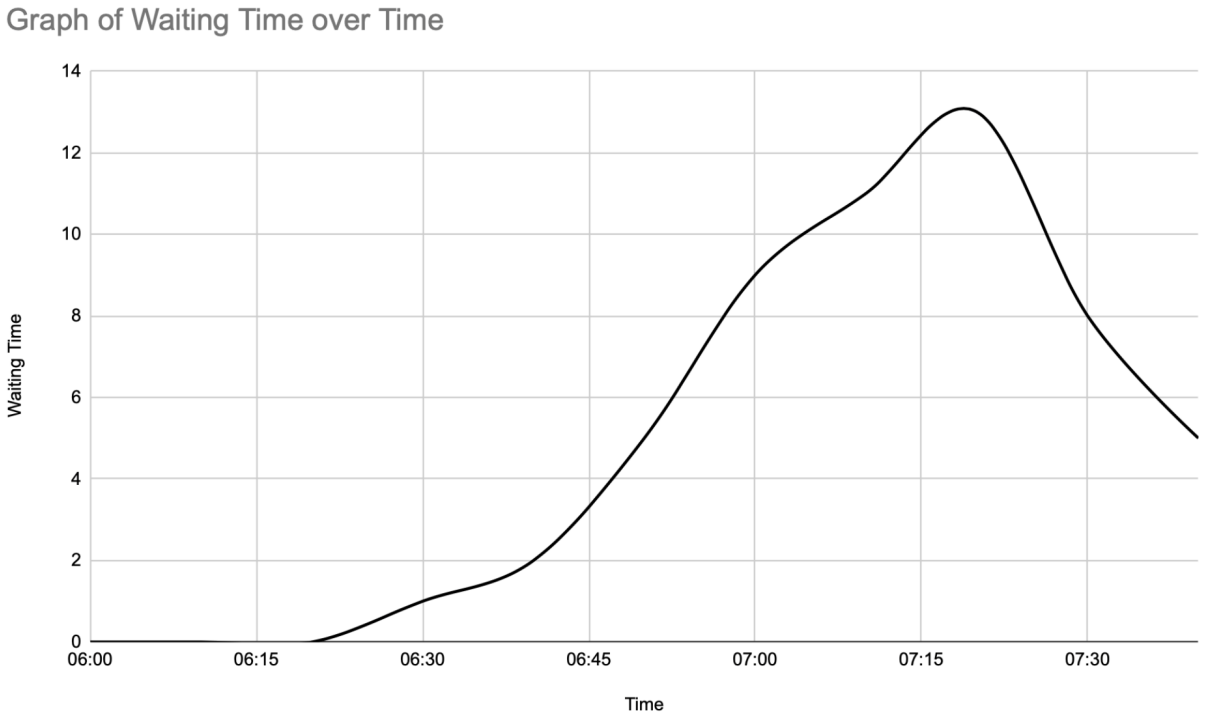
We can see how there is severe congestion on the roads outside of Gate 3 and 4, at peak hours 07:20 hours as most students are coming in now.

To understand the rationale behind the simulation, the cars generated by the simulator all essentially have a “mind” on their own, where their speed and pathway are determined by them and not predetermined. There is a slider at the debug menu of the simulation that was not shown on the picture above which allows us to adjust the number of cars appearing at this juncture, effectively recreating the different car frequencies of different times in the morning. With this, the average delay or waiting time generated by the simulator allows us to achieve our goal. This way of simulation is more accurate with the real world and this brings for more accurate results.

The results obtained by the different frequencies of cars are displayed below:

Time	Waiting Time
0600	0
0610	0
0620	0
0630	1
0640	2
0650	5
0700	9
0710	11
0720	13
0730	8
0740	5
Average:	**4.9**

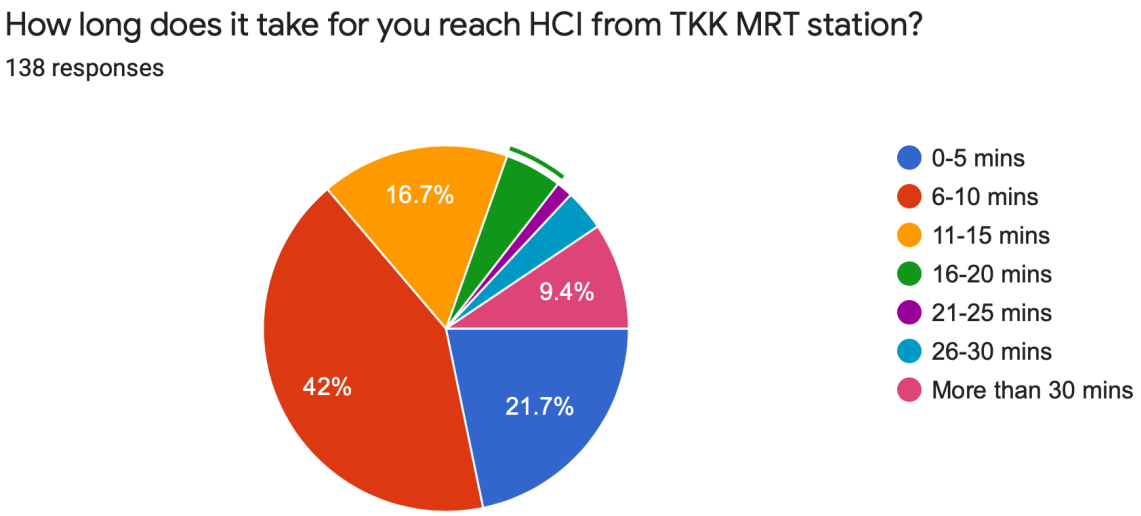
Thus, based on our simulation, the average waiting time is 4.9 minutes, or **4 minutes 54 seconds**. The graph below is to give a better grasp of the magnitude of the increase of waiting time as time progresses.



Survey Data

The data on the waiting time was also collected via a survey. A survey was sent out by us to the student population and received 138 responses.

The first question is on the time needed for students to take the bus and travel from the MRT station to the school.



It was calculated that the average time taken is approximately 11.12 minutes. As the travelling time without traffic on that road is 90 seconds, after dividing the length by the LTA approved cruising speed of buses, the average waiting time from our survey is 9.62 minutes, or **9 minutes 37 seconds**.

However, this number also accounts for the time needed for students to walk into the gate, which is probably what caused the time to be higher than that of our simulation.

Summary of the 2 sources of Data

Through the simulation data obtained, approximately 5 minutes are wasted on the journey to school alone. If we account for other factors, such as those mentioned in the survey, an average of 7 minutes is more realistic by averaging the results obtained from the simulator and survey. This brings on the next issue on what we can do to better the issue and reduce the waiting time.

Research Question 3

Which solutions can reduce the waiting time of students the most, based on the web-based simulator created?

After analysing the traffic congestion situation near HCI, some solutions should be suggested to reduce waiting time and increase productivity. The methods of the solutions can be mainly split into the HCI front and the Government front. Pushing back school reporting time, road pricing, a temporary number limit, and road upgrade could be explored to see if these methods are useful in mitigating traffic.

Pushing Back School Reporting Time

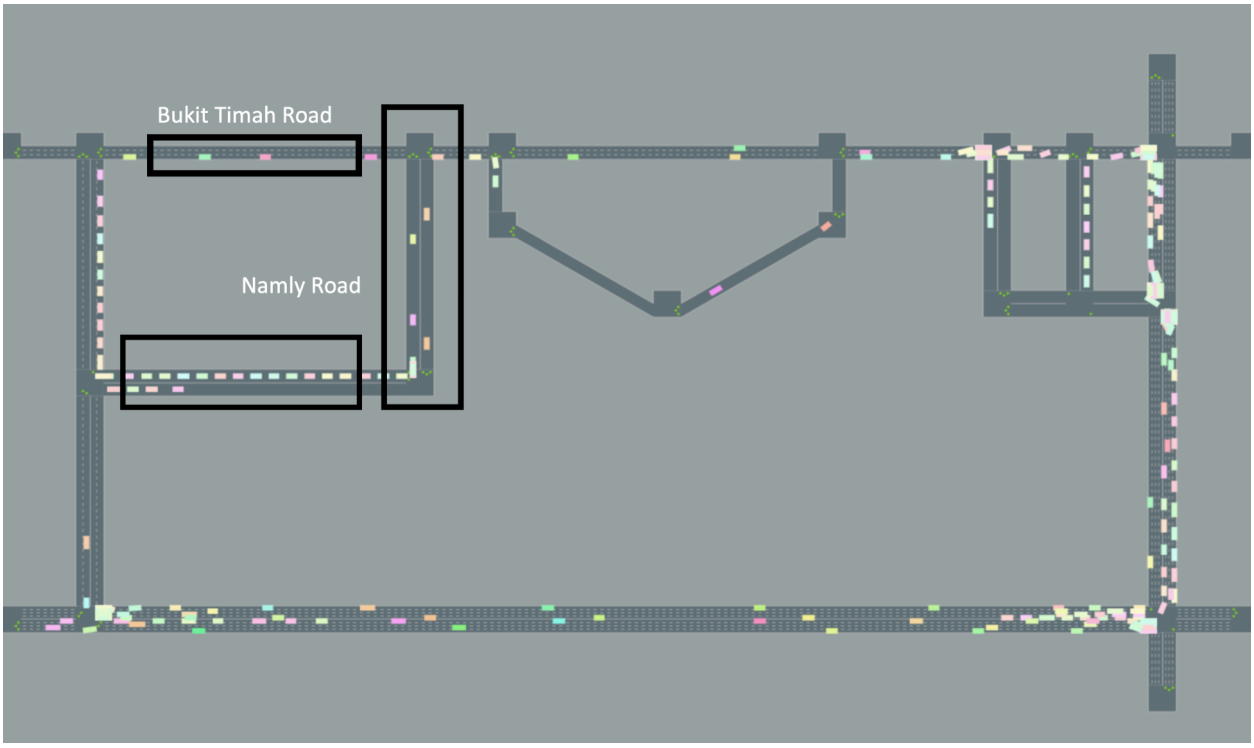
On the HCI front, what we can do is to push back school reporting time to avoid other cars coming into other schools at that current reporting time. Most of the traffic at 7:30 pm originates from parents driving to NYPS (Nanyang Primary School) and NJC (National Junior College) and they will combine with the HCI traffic to crowd Bukit Timah Road. By pushing back the reporting time to 7:45 pm, we are able to perfectly avoid the NYPS & NJC traffic at 7:30 pm and the NYGH (Nanyang Girls's High) traffic at 8:00 pm. By observing traffic data from LTA, we were able to find data on the average traffic speed on Bukit Timah and Dunearn Road at certain times of the day. We can thus work backwards and find the volume of traffic passing through the road, where this strategy is approximately able to reduce the number of cars on the road by 15%.

Below is a table showing the new data for the number of cars.

Time	Original Number of Cars	Current Number of Cars	Waiting Time
06:00	3	3	0
06:10	7	6	0
06:20	7	6	0
06:30	17	15	1
06:40	35	30	2
06:50	92	78	4
07:00	159	135	6
07:10	197	167	8
07:20	171	145	9
07:30	76	65	8
07:40	21	18	8
Average			**4.2**

The new waiting time is now 4.2 minutes, or **4 minutes 12 seconds**. We can see that by reducing the traffic by 15%, we can reduce the waiting time by 14%, which is a decent outcome. The

picture below is the simulated traffic, we can see how traffic congestion is relatively less serious in the school at peak hours.



However, we can see that this method is not as effective as the methods that will be mentioned below as it is merely a microscopic change to the traffic situation, which does not bring out much difference in the grand scheme of things.

There, we need a macroscopic change to combat the traffic situation which can only be done via government intervention. The following are 3 strategies that the government can use to reduce traffic congestion at Bukit Timah Road.

Road Pricing

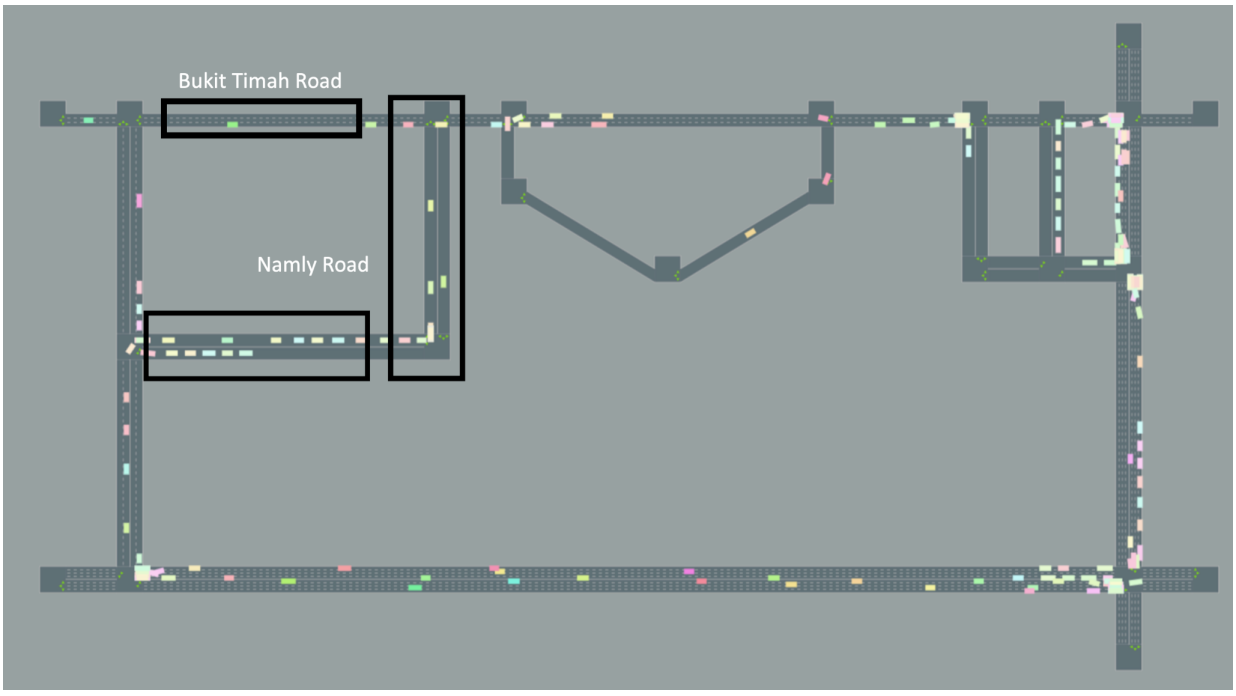
Road pricing is direct charges levied for the use of roads, including road tolls, distance or time-based fees, congestion charges and charges designed to discourage the use of certain classes of vehicle, fuel sources or more polluting vehicles. This is mainly to increase the cost of road usage so as to not incentivise drivers to use that road. This could be applied to the Hwa Chong context and it is a possibility that road pricing could be implemented on Bukit Timah Road to help to alleviate the situation.

According to LTA, approximately 25% of traffic is reduced due to Singapore's road pricing. We can thus reduce the number of cars to 75% of the original value which will give us a simulation of what happens if road pricing is implemented. The data below is the new number of cars and the new average waiting time.

Time	Original Number of Cars	Current Number of Cars	Waiting Time
06:00	3	2	0
06:10	7	5	0
06:20	7	5	0
06:30	17	13	1
06:40	35	26	1
06:50	92	69	2

07:00	159	119	4
07:10	197	148	7
07:20	171	128	9
07:30	76	57	6
07:40	21	16	3
Average			**3.0**

The new waiting time is now 3.0 minutes, or **3 minutes 0 seconds**. We can see that by reducing the traffic by 25%, we can reduce the waiting time by 39%, which is a good outcome. The picture below is the simulated traffic, we can see how traffic congestion is less serious in the school at peak hours.



Temporary Number Limit

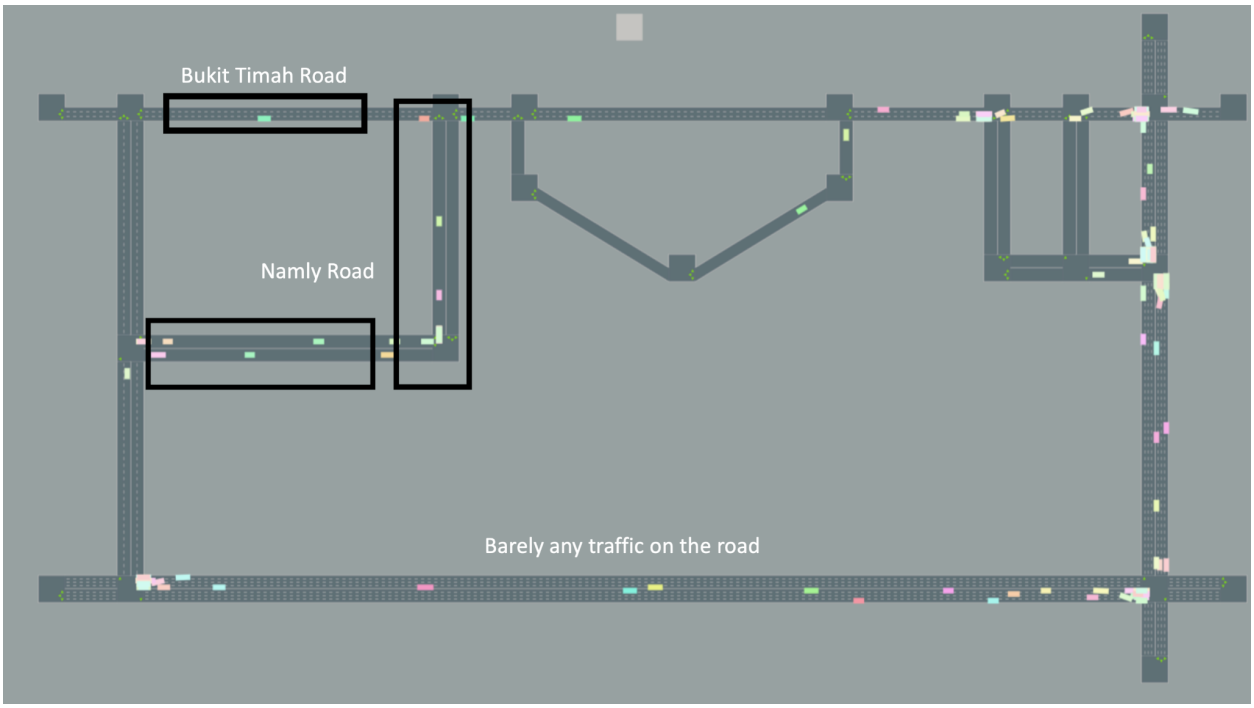
Temporary Number Limit is the implementation of a rule such that specific cars can be on the road on specific days. For example, an even/odd number limit means that only those cars whose license plates are even/odd can drive on even/odd days respectively. This is very common in crowded China cities and has done its part to reduce traffic congestion in these places. Nevertheless, this strategy will still cause inconvenience to many people but it is the most effective strategy as it is able to reduce traffic by 50%.

The data below is the new number of cars and the new average waiting time.

Time	Original Number of Cars	Current Number of Cars	Waiting Time
06:00	3	2	0
06:10	7	4	0
06:20	7	4	0
06:30	17	8	1
06:40	35	17	1
06:50	92	46	1
07:00	159	79	2
07:10	197	98	4

07:20	171	85	6
07:30	76	38	3
07:40	21	10	1
Average			**1.7**

The new waiting time is now 1.7 minutes, or **1 minute 42 seconds**. We can see that by reducing the traffic by 50%, we can reduce the waiting time by 65%, which is the best outcome but has many trade-offs. The picture below is the simulated traffic, we can see how traffic congestion is much less serious in the school at peak hours.



Road Upgrade

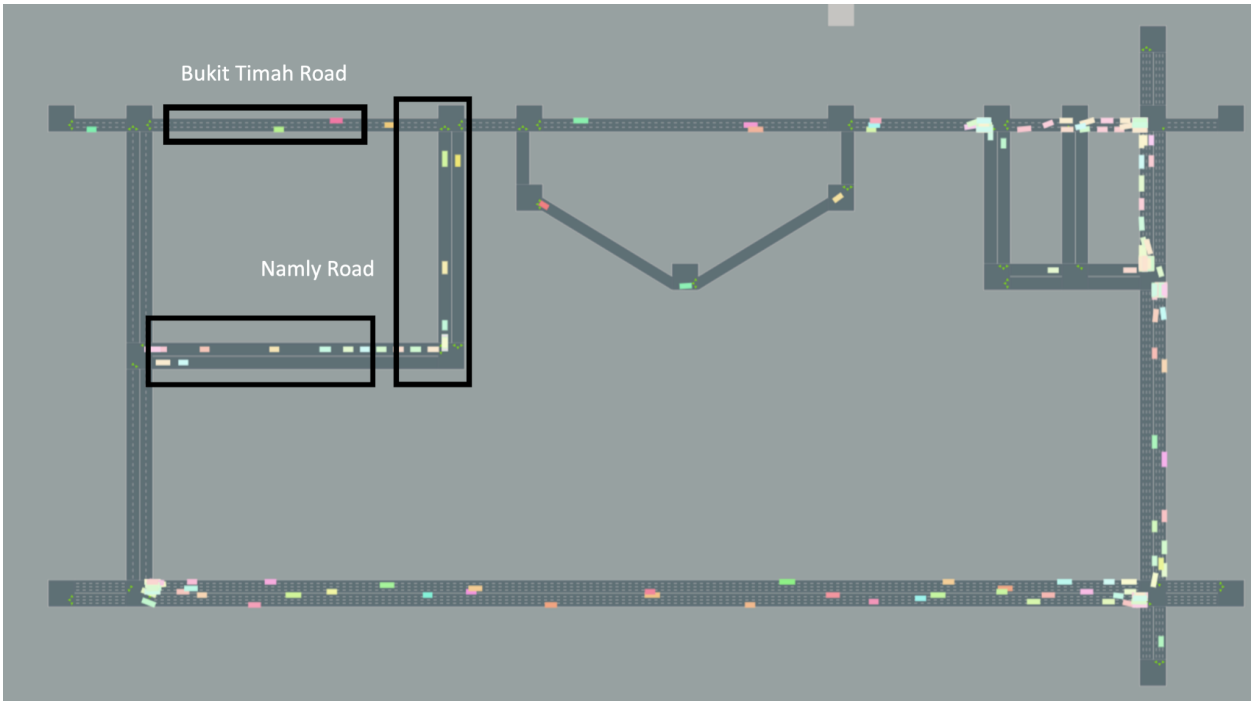
Lane widening, which is the increase of lane width of roads, could be considered to reduce traffic congestion. The upgrade of road infrastructure will help to increase the capacity of roads such that more cars can enter the road without overloading it. By just increasing Bukit Timah Road by 1 lane, we can increase the capacity of the road by 20%, which is a good expenditure to resolve the congestion issue.

The table below shows the new number of cars and the waiting time obtained from the simulation.

Time	Original Number of Cars	Current Number of Cars	Waiting Time
06:00	3	2	0
06:10	7	6	0
06:20	7	6	0
06:30	17	14	1
06:40	35	28	2
06:50	92	74	4
07:00	159	127	4
07:10	197	158	6

07:20	171	137	8
07:30	76	61	11
07:40	21	17	5
Average			**3.7**

We can see that the new waiting time is 3.7 minutes, or **3 minutes 42 seconds.** We can see that by increasing road capacity by 20%, we can reduce the waiting time by 24%, which is not a lot compared to the strategies above. Nevertheless, the picture below is the simulated traffic, we can see how traffic congestion is less serious in the school at peak hours.



Evaluation

In summary, pushing back school reporting time, road pricing, a temporary number limit, and road upgrade have reduced the waiting time by 14%, 39%, 65%, 24% respectively. These are all relatively noticeable results and all will help to better the situation of traffic congestion outside of HCI. However, the issues surrounding these methods are still existing and we should weigh the trade-off that each method brings to select the best one.

Pushing back school reporting time is a feasible strategy in which the school can implement but it lacks the effectiveness that other macro-scaled policies could have. As traffic is something that is on a macroscale, changing it slightly on the microscale will not affect the overall results by too much. This will inherently cause any strategies implemented by the school to be less effective than the other government-level strategies which are unfortunate. Nevertheless, this is the only strategy in which the school can implement, and something that the school can possibly do. Thus, despite the implications of this method, it should still be taken into consideration to reduce traffic outside the school.

Road pricing is feasible to implement but it could cause the people to be unhappy as they now have to pay to use Bukit Timah Road. When they drive into the road, they will have to pay a certain amount of money. However, this will definitely cause negative sentiments which can reflect badly on the school. Furthermore, the money needed to build an ERP gantry is also significant and maintenance is an issue. The question of funding is uncertain and it definitely needs government intervention to properly work. Nevertheless, it still brings good results by nearly half the waiting time and can be considered.

A temporary number limit is difficult to implement as it causes a lot of inconveniences to the consumers as they can now only drive their cars for half the week. This builds on the randomness of car number plates and bans certain numbers on certain days and it will definitely be the most aggressive but effective strategy. It is the one that can bring the most benefit as it reduces the waiting time to nearly 0. Despite the risk of the methods, it brings great results and should also be considered from implementation. It only depends on the willingness of the government to implement it.

Road upgrade is the most accepted form of managing traffic as the concept has been proven and tested. By just increasing the road capacity, we can allow for more cars to pass through it at one time without congesting the road. However, it costs a lot of money and land is an issue as there is little available land around Bukit Timah Road. Unfortunately, the results are often less than expected due to the ineffectiveness of the method. Nevertheless, the conservativeness of the method allows the feasibility to be very high and the technology required is well established. It can also be considered for adaptation.

In conclusion, all of these methods are doable and should be considered to alleviate the traffic congestion situation at HCI. However, both pros and cons exist in these strategies and careful consideration is needed to decide on the one to use. It all depends on the style of the decider on whether a high-risk high reward strategy is preferred or a low-risk low-reward method is favoured. This project only wishes to put out the facts and there they are.

Limitations

One limitation of this project is the lack of time to collect data for a long term project as the CCTV footage of the school only saves up to 1 month. We thus have to go every month to collect the data, and there is no long term data of the past few years available. However, for this model to be more accurate, months and years of data have to be collected to form a comprehensive database.

Furthermore, the lack of data on roads other than the ones near the school premises could cause some inaccuracies when simulating the traffic flow as the traffic condition of the roads further along the school are not considered due to the absence of long term data. The constraints of the data have limited us to approximate the traffic condition to a higher degree of accuracy and it is something that can be improved on.

Another limitation of this project is the lack of professional knowledge in the field of computing, which has limited the scope and depth of the model. The accuracy of the results could also be better if the model is improved. Nevertheless, despite the existence of these various limitations, this project can still serve as a prototype that is applicable in real life.

Further Extension

The cars' length or the l_α value can be considered in the model if there is sufficient computational power. By looking at the cars on the individual level, where the length of each car is accounted for, even more, accurate results can be achieved, even though the assumption is valid that most private cars are about the same length.

It is also possible to collaborate with external organizations such as LTA (Land Transport Authority) to gain sufficient data of the entire Bukit Timah Road and its branching roads to create a larger environment for simulation. This will enable the project to be of a larger scale and have higher accuracy.

Further additions to the model can also be done to make it adaptable, where it can be easily modified to fit into other situations. It could also change from a web-based simulator to a C++ based simulator for GPU acceleration to allow for more cars to be added and faster calculator times to allow for a larger and more comprehensive environment.

References

[1] Azlan, N. N. N., & Rohani, M. M. (2018). Overview of application of traffic simulation model. In MATEC Web of Conferences (Vol. 150, p. 03006). EDP Sciences.

[2] Chapra, S. C., & Canale, R. P. Numerical methods for engineers. Boston: McGraw-Hill, 2006

[3] Chu, L., Liu, H. X., Oh, J. S., & Recker, W. (2003, October). A calibration procedure for microscopic traffic simulation. In Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems (Vol. 2, pp. 1574-1579). IEEE.

[4] Jaume, B. (Ed.). (2010). *Fundamentals of Traffic Simulation*. Springer Publishing. https://doi.org/10.1007/978-1-4419-6142-6

[5] Klefstad, R., Zhang, Y., Lai, M., Jayakrishnan, R., & Lavanya, R. (2005, September). A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005. (pp. 813-818). IEEE.

[6] Kotusevski, G., & Hawick, K. A. (2009). A review of traffic simulation software.

[7] Kutta M.W., "Beitrag zur näherungsweisen Integration totaler Differentialgleichungen", Zeitschrift für Mathematik und Physik, 46: 435–453, 1901

[8] Leemis, L. M., & Park, S. K. . Discrete-event simulation: A first course. Upper Saddle River, N.J.: Pearson Prentice Hall, 2006

[9] Lighthill M.J., Whitham G.B., On kinematic waves II: A theory of traffic flow on long, crowded roads. Proceedings of the Royal Society of London Series A 229, 317-345, 1955

[10] Mahmud, Khizir; Town, Graham E. "A review of computer tools for modelling electric vehicle energy requirements and their impact on power distribution networks". Applied Energy. 172: 337–359. doi:10.1016/j.apenergy.2016.03.100, 2016

[11] Pell, A., Meingast, A., & Schauer, O. (2017). Trends in real-time traffic simulation. Transportation research procedia, 25, 1477-1484.

[12] P. I. Richards, Shock waves on the highway, Operations Research 4, 42–51, 1956

[13] Raney, B., Voellmy, A., Cetin, N., Vrtic, M., & Nagel, K. (2002, April). Towards a microscopic traffic simulation of all of Switzerland. In the International Conference on Computational Science (pp. 371-380). Springer, Berlin, Heidelberg.

[14] Sg Traffic Watch. (n.d.). Real Time Singapore Traffic Watching. Sg Trafficwatch.Org. Retrieved August 9, 2021, from https://sgtrafficwatch.org/

[15] Shiraishi, T., Hanabusa, H., Kuwahara, M., Chung, E., Tanaka, S., Ueno, H., ... & Yamamoto, T. (2004, October). Development of a microscopic traffic simulation model for an interactive traffic environment. In Proc. 11th World Congress on Intelligent Transport Systems and Services (ITSWC 2004), Nagoya, Japan.

[16] Sokolowski J.A., & Banks C.M. Principles of modelling and simulation: A multidisciplinary approach. Hoboken, N.J.: John Wiley, 2009

[17] Sommer, C., German, R., & Dressler, F. (2010, July). Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/TMC.2010.133

[18] Spyropoulou, Ioanna. "Simulation Using Gipps' Car-Following Model—An In-Depth Analysis". Transportmetrica. 3 (3): 231–245. doi:10.1080/18128600708685675, 2007

[19] Treiber M., Hennecke A., Helbing D. Congested Traffic States in Empirical Observations and Microscopic Simulations. Physical Review E 62, 1805-1824, 2000

[20] Volkhin, A. (2015, April 25). GitHub - volkin/RoadTrafficSimulator: Road traffic simulator and signals optimizer in CoffeeScript & HTML5. GitHub. https://github.com/volkhin/RoadTrafficSimulator

[21] Wilson, R. E. . "An analysis of Gipps' car-following model of highway traffic". IMA

Journal of Applied Mathematics. 66 (5): 509–537. Bibcode:2001JApMa..66..509W.

doi:10.1093/imamate/66.5.509., 2001

Appendix(s)

Appendix I - Overview of the Code of Simulator

File	Description
Index.html	Main web page
geom/curve.coffee	For object curve
geom/point.coffee	For object point
geom/rect.coffee	For object rectangle
geom/segment.coffee	For object segment
model/car.coffee	For object car
model/control-signals.coffee	For object control signals
model/intersection.coffee	For object intersection
model/lane-position.coffee	For object road lane position
model/lane.coffee	For object road lane
model/pool.coffee	For object pool
model/road.coffee	For object road
model/trajectory.coffee	For object trajectory
model/world.coffee	For object world

visualizer/graphics.coffee	For drawing graphics
visualizer/highlighter.coffee	For object highlighter
visualizer/intersection-builder.coffee	For object intersection builder
visualizer/intersection-mover.coffee	For object intersection mover
visualizer/mover.coffee	For object mover
visualizer/road-builder.coffee	For object road builder
visualizer/tool.coffee	For object tool
visualizer/visualizer.coffee	For object visualizer
visualizer/zoomer.coffee	For object zoomer
app.coffee	For object app
runner.coffee	For object runner
settings.coffee	For object settings

Appendix II - Specific Code of Simulator

Index.html - Main web page

```
<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="css/style.css" />
    <link rel="icon" href="images/favicon.png" />
    <script src="dist/main.js"></script>
    <link rel="stylesheet" href="css/dat-gui.css" />
    <title>Road traffic simulator</title>
</head>
<body>
</body>
</html>
```

geom/curve.coffee - For object curve

```
'use strict'


require '../helpers'
Segment = require './segment'


class Curve
  constructor: (@A, @B, @O, @Q) ->
    @AB = new Segment @A, @B
    @AO = new Segment @A, @O
    @OQ = new Segment @O, @Q
    @QB = new Segment @Q, @B
    @_length = null


  @property 'length',
    get: ->
      if not @_length?
        pointsNumber = 10
        prevoiusPoint = null
        @_length = 0
        for i in [0..pointsNumber]
          point = @getPoint i / pointsNumber
          @_length += point.subtract(prevoiusPoint).length if prevoiusPoint
```

```
        prevoiusPoint = point
      return @_length


  getPoint: (a) ->
    # OPTIMIZE avoid points and segments
    p0 = @AO.getPoint(a)
    p1 = @OQ.getPoint(a)
    p2 = @QB.getPoint(a)
    r0 = (new Segment p0, p1).getPoint a
    r1 = (new Segment p1, p2).getPoint a
    (new Segment r0, r1).getPoint a


  getDirection: (a) ->
    # OPTIMIZE avoid points and segments
    p0 = @AO.getPoint(a)
    p1 = @OQ.getPoint(a)
    p2 = @QB.getPoint(a)
    r0 = (new Segment p0, p1).getPoint a
    r1 = (new Segment p1, p2).getPoint a
    (new Segment r0, r1).direction


module.exports = Curve
```

geom/point.coffee - For object point

```
'use strict'


{sqrt, atan2} = Math

require '../helpers'


class Point

  constructor: (@x = 0, @y = 0) ->


  @property 'length',

    get: ->

      sqrt @x * @x + @y * @y


  @property 'direction',

    get: ->

      atan2 @y, @x


  @property 'normalized',

    get: ->

      new Point @x / @length, @y / @length


  add: (o) ->

    new Point @x + o.x, @y + o.y
```

```
  subtract: (o) ->
    new Point @x - o.x, @y - o.y

  mult: (k) ->
    new Point @x * k, @y * k

  divide: (k) ->
    new Point @x / k, @y / k

module.exports = Point
```

geom/rect.coffee - For object rectangle

```
'use strict'


{abs} = Math

require '../helpers'

_ = require 'underscore'

Point = require './point'

Segment = require './segment'


class Rect

  constructor: (@x, @y, @_width = 0, @_height = 0) ->


  @copy: (rect) ->

    new Rect rect.x, rect.y, rect._width, rect._height


  toJSON: ->

    _.extend {}, this


  area: ->

    @width() * @height()


  left: (left) ->

    @x = left if left?
```

```
  @x

right: (right) ->
  @x = right - @width() if right?
  @x + @width()

width: (width) ->
  @_width = width if width?
  @_width

top: (top) ->
  @y = top if top?
  @y

bottom: (bottom) ->
  @y = bottom - @height() if bottom?
  @y + @height()

height: (height) ->
  @_height = height if height?
  @_height

center: (center) ->
```

```
  if center?
    @x = center.x - @width() / 2
    @y = center.y - @height() / 2
  new Point @x + @width() / 2, @y + @height() / 2

containsPoint: (point) ->
  @left() <= point.x <= @right() and @top() <= point.y <= @bottom()

containsRect: (rect) ->
  @left() <= rect.left() and rect.right() <= @right() and
  @top() <= rect.top() and rect.bottom() <= @bottom()

getVertices: ->
  [
    new Point(@left(), @top()),
    new Point(@right(), @top()),
    new Point(@right(), @bottom()),
    new Point(@left(), @bottom()),
  ]

getSide: (i) ->
  vertices = @getVertices()
  new Segment vertices[i], vertices[(i + 1) % 4]
```

```
  getSectorId: (point) ->
    offset = point.subtract @center()
    return 0 if offset.y <= 0 and abs(offset.x) <= abs(offset.y)
    return 1 if offset.x >= 0 and abs(offset.x) >= abs(offset.y)
    return 2 if offset.y >= 0 and abs(offset.x) <= abs(offset.y)
    return 3 if offset.x <= 0 and abs(offset.x) >= abs(offset.y)
    throw Error 'algorithm error'

  getSector: (point) ->
    @getSide @getSectorId point

module.exports = Rect
```

geom/segment.coffee - For object segment

```
'use strict'

require '../helpers'

class Segment
  constructor: (@source, @target) ->

  @property 'vector',
    get: ->
      @target.subtract @source

  @property 'length',
    get: ->
      @vector.length

  @property 'direction',
    get: ->
      @vector.direction

  @property 'center',
    get: ->
      @getPoint 0.5
```

```
  split: (n, reverse) ->
    order = if reverse then [n - 1 .. 0] else [0 .. n - 1]
    @subsegment k / n, (k + 1) / n for k in order

  getPoint: (a) ->
    @source.add (@vector.mult a)

  subsegment: (a, b) ->
    offset = @vector
    start = @source.add (offset.mult a)
    end = @source.add (offset.mult b)
    new Segment start, end

module.exports = Segment
```

model/car.coffee - For object car

```
'use strict'


{max, min, random, sqrt} = Math

require '../helpers'

_ = require 'underscore'

Trajectory = require './trajectory'


class Car

  constructor: (lane, position) ->

    @id = _.uniqueId 'car'

    @color = (300 + 240 * random() | 0) % 360

    @_speed = 0

    @width = 1.7

    @length = 3 + 2 * random()

    @maxSpeed = 30

    @s0 = 2

    @timeHeadway = 1.5

    @maxAcceleration = 1

    @maxDeceleration = 3

    @trajectory = new Trajectory this, lane, position

    @alive = true

    @preferedLane = null
```

```
  @property 'coords',
    get: -> @trajectory.coords

  @property 'speed',
    get: -> @_speed
    set: (speed) ->
      speed = 0 if speed < 0
      speed = @maxSpeed if speed > @maxSpeed
      @_speed = speed

  @property 'direction',
    get: -> @trajectory.direction

  release: ->
    @trajectory.release()

  getAcceleration: ->
    nextCarDistance = @trajectory.nextCarDistance
    distanceToNextCar = max nextCarDistance.distance, 0
    a = @maxAcceleration
    b = @maxDeceleration
    deltaSpeed = (@speed - nextCarDistance.car?.speed) || 0
```

```
    freeRoadCoeff = (@speed / @maxSpeed) ** 4
    distanceGap = @s0
    timeGap = @speed * @timeHeadway
    breakGap = @speed * deltaSpeed / (2 * sqrt a * b)
    safeDistance = distanceGap + timeGap + breakGap
    busyRoadCoeff = (safeDistance / distanceToNextCar) ** 2
    safeIntersectionDistance = 1 + timeGap + @speed ** 2 / (2 * b)
    intersectionCoeff =
    (safeIntersectionDistance / @trajectory.distanceToStopLine) ** 2
    coeff = 1 - freeRoadCoeff - busyRoadCoeff - intersectionCoeff
    return @maxAcceleration * coeff

  move: (delta) ->
    acceleration = @getAcceleration()
    @speed += acceleration * delta

    if not @trajectory.isChangingLanes and @nextLane
      currentLane = @trajectory.current.lane
      turnNumber = currentLane.getTurnDirection @nextLane
      preferedLane = switch turnNumber
        when 0 then currentLane.leftmostAdjacent
        when 2 then currentLane.rightmostAdjacent
        else currentLane
```

```
    if preferedLane isnt currentLane
      @trajectory.changeLane preferedLane

  step = @speed * delta + 0.5 * acceleration * delta ** 2
  # TODO: hacks, should have changed speed
  console.log 'bad IDM' if @trajectory.nextCarDistance.distance < step

  if @trajectory.timeToMakeTurn(step)
    return @alive = false if not @nextLane?
  @trajectory.moveForward step

pickNextRoad: ->
  intersection = @trajectory.nextIntersection
  currentLane = @trajectory.current.lane
  possibleRoads = intersection.roads.filter (x) ->
    x.target isnt currentLane.road.source
  return null if possibleRoads.length is 0
  nextRoad = _.sample possibleRoads

pickNextLane: ->
  throw Error 'next lane is already chosen' if @nextLane
  @nextLane = null
  nextRoad = @pickNextRoad()
```

```
    return null if not nextRoad
    # throw Error 'can not pick next road' if not nextRoad
    turnNumber = @trajectory.current.lane.road.getTurnDirection nextRoad
    laneNumber = switch turnNumber
      when 0 then nextRoad.lanesNumber - 1
      when 1 then _.random 0, nextRoad.lanesNumber - 1
      when 2 then 0
    @nextLane = nextRoad.lanes[laneNumber]
    throw Error 'can not pick next lane' if not @nextLane
    return @nextLane

  popNextLane: ->
    nextLane = @nextLane
    @nextLane = null
    @preferedLane = null
    return nextLane

module.exports = Car
```

model/control-signals.coffee - For object control signals

```
'use strict'


{random} = Math

require '../helpers'

settings = require '../settings'


class ControlSignals

  constructor: (@intersection) ->

    @flipMultiplier = random()

    @phaseOffset = 100 * random()

    @time = @phaseOffset

    @stateNum = 0


  @copy: (controlSignals, intersection) ->

    if !controlSignals?

      return new ControlSignals intersection

    result = Object.create ControlSignals::

    result.flipMultiplier = controlSignals.flipMultiplier

    result.time = result.phaseOffset = controlSignals.phaseOffset

    result.stateNum = 0

    result.intersection = intersection

    result
```

```
  toJSON: ->
    obj =
      flipMultiplier: @flipMultiplier
      phaseOffset: @phaseOffset

  states: [
    ['L', '', 'L', ''],
    ['FR', '', 'FR', ''],
    ['', 'L', '', 'L'],
    ['', 'FR', '', 'FR']
  ]

  @STATE = [RED: 0, GREEN: 1]

  @property 'flipInterval',
    get: -> (0.1 + 0.05 * @flipMultiplier) * settings.lightsFlipInterval

  _decode: (str) ->
    state = [0, 0, 0]
    state[0] = 1 if 'L' in str
    state[1] = 1 if 'F' in str
    state[2] = 1 if 'R' in str
```

```
    state


  @property 'state',
    get: ->
      stringState = @states[@stateNum % @states.length]
      if @intersection.roads.length <= 2
        stringState = ['LFR', 'LFR', 'LFR', 'LFR']
      (@_decode x for x in stringState)


  flip: ->
    @stateNum += 1


  onTick: (delta) =>
    @time += delta
    if @time > @flipInterval
      @flip()
      @time -= @flipInterval


module.exports = ControlSignals
```

model/intersection.coffee - For object intersection

```
'use strict'


require '../helpers'
_ = require 'underscore'
ControlSignals = require './control-signals'
Rect = require '../geom/rect'


class Intersection
  constructor: (@rect) ->
    @id = _.uniqueId 'intersection'
    @roads = []
    @inRoads = []
    @controlSignals = new ControlSignals this

  @copy: (intersection) ->
    intersection.rect = Rect.copy intersection.rect
    result = Object.create Intersection::
    _.extend result, intersection
    result.roads = []
    result.inRoads = []
    result.controlSignals = ControlSignals.copy result.controlSignals, result
    result
```

```
  toJSON: ->
    obj =
      id: @id
      rect: @rect
      controlSignals: @controlSignals

  update: ->
    road.update() for road in @roads
    road.update() for road in @inRoads

module.exports = Intersection
```

model/lane-position.coffee - For object road lane position

```
'use strict'


require '../helpers'
_ = require 'underscore'


class LanePosition
  constructor: (@car, lane, @position) ->
    @id = _.uniqueId 'laneposition'
    @free = true
    @lane = lane


  @property 'lane',
    get: -> @_lane
    set: (lane) ->
      @release()
      @_lane = lane
      # @acquire()


  @property 'relativePosition',
    get: -> @position / @lane.length


  acquire: ->
```

```
  if @lane?.addCarPosition?
    @free = false
    @lane.addCarPosition this

release: ->
  if not @free and @lane?.removeCar
    @free = true
    @lane.removeCar this

getNext: ->
  return @lane.getNext this if @lane and not @free

@property 'nextCarDistance',
  get: ->
    next = @getNext()
    if next
      rearPosition = next.position - next.car.length / 2
      frontPosition = @position + @car.length / 2
      return result =
        car: next.car
        distance: rearPosition - frontPosition
    return result =
      car: null
```

```
        distance: Infinity


module.exports = LanePosition
```

model/lane.coffee - For object road lane

```
'use strict'


require '../helpers'
_ = require 'underscore'
Segment = require '../geom/segment'


class Lane
  constructor: (@sourceSegment, @targetSegment, @road) ->
    @leftAdjacent = null
    @rightAdjacent = null
    @leftmostAdjacent = null
    @rightmostAdjacent = null
    @carsPositions = {}
    @update()


  toJSON: ->
    obj = _.extend {}, this
    delete obj.carsPositions
    obj


  @property 'sourceSideId',
    get: -> @road.sourceSideId
```

```
  @property 'targetSideId',
    get: -> @road.targetSideId

  @property 'isRightmost',
    get: -> this is @.rightmostAdjacent

  @property 'isLeftmost',
    get: -> this is @.leftmostAdjacent

  @property 'leftBorder',
    get: ->
      new Segment @sourceSegment.source, @targetSegment.target

  @property 'rightBorder',
    get: ->
      new Segment @sourceSegment.target, @targetSegment.source

  update: ->
    @middleLine = new Segment @sourceSegment.center, @targetSegment.center
    @length = @middleLine.length
    @direction = @middleLine.direction
```

```
  getTurnDirection: (other) ->
    return @road.getTurnDirection other.road

  getDirection: ->
    @direction

  getPoint: (a) ->
    @middleLine.getPoint a

  addCarPosition: (carPosition) ->
    throw Error 'car is already here' if carPosition.id of @carsPositions
    @carsPositions[carPosition.id] = carPosition

  removeCar: (carPosition) ->
    throw Error 'removing unknown car' unless carPosition.id of @carsPositions
    delete @carsPositions[carPosition.id]

  getNext: (carPosition) ->
    throw Error 'car is on other lane' if carPosition.lane isnt this
    next = null
    bestDistance = Infinity
    for id, o of @carsPositions
      distance = o.position - carPosition.position
```

```
      if not o.free and 0 < distance < bestDistance
        bestDistance = distance
        next = o
    next

module.exports = Lane
```

model/pool.coffee - For object pool

```
'use strict'


require '../helpers'


class Pool
  constructor: (@factory, pool) ->
    @objects = {}
    if pool? and pool.objects?
      for k, v of pool.objects
        @objects[k] = @factory.copy(v)

  toJSON: ->
    @objects

  get: (id) ->
    @objects[id]

  put: (obj) ->
    @objects[obj.id] = obj

  pop: (obj) ->
    id = obj.id ? obj
```

```
    result = @objects[id]
    result.release?()
    delete @objects[id]
    result

  all: ->
    @objects

  clear: ->
    @objects = {}

  @property 'length',
    get: -> Object.keys(@objects).length

module.exports = Pool
```

model/road.coffee - For object road

```
'use strict'


{min, max} = Math

require '../helpers'

_ = require 'underscore'

Lane = require './lane'

settings = require '../settings'


class Road

  constructor: (@source, @target, @maxLanesNumber) ->

    @id = _.uniqueId 'road'

    @lanes = []

    @lanesNumber = null

    @update()


  @copy: (road) ->

    result = Object.create Road::

    _.extend result, road

    result.lanes ?= []

    result


  toJSON: ->
```

```
    obj =
      id: @id
      source: @source.id
      target: @target.id

  @property 'length',
    get: -> @targetSide.target.subtract(@sourceSide.source).length

  @property 'leftmostLane',
    get: -> @lanes[@lanesNumber - 1]

  @property 'rightmostLane',
    get: -> @lanes[0]

  getTurnDirection: (other) ->
    throw Error 'invalid roads' if @target isnt other.source
    side1 = @targetSideId
    side2 = other.sourceSideId
    # 0 - left, 1 - forward, 2 - right
    turnNumber = (side2 - side1 - 1 + 8) % 4

  update: ->
    throw Error 'incomplete road' unless @source and @target
```

```
    @sourceSideId = @source.rect.getSectorId @target.rect.center()
    @sourceSide = @source.rect.getSide(@sourceSideId).subsegment 0, 0.5
    @targetSideId = @target.rect.getSectorId @source.rect.center()
    @targetSide = @target.rect.getSide(@targetSideId).subsegment 0.5, 1.0
    @lanesNumber = min(@sourceSide.length, @targetSide.length) | 0
    @lanesNumber = max @maxLanesNumber or 2, @lanesNumber / settings.gridSize | 0
    sourceSplits = @sourceSide.split @lanesNumber, true
    targetSplits = @targetSide.split @lanesNumber
    if not @lanes? or @lanes.length < @lanesNumber
      @lanes ?= []
      for i in [0..@lanesNumber - 1]
        @lanes[i] ?= new Lane sourceSplits[i], targetSplits[i], this
    for i in [0..@lanesNumber - 1]
      @lanes[i].sourceSegment = sourceSplits[i]
      @lanes[i].targetSegment = targetSplits[i]
      @lanes[i].leftAdjacent = @lanes[i + 1]
      @lanes[i].rightAdjacent = @lanes[i - 1]
      @lanes[i].leftmostAdjacent = @lanes[@lanesNumber - 1]
      @lanes[i].rightmostAdjacent = @lanes[0]
      @lanes[i].update()

module.exports = Road
```

model/trajectory.coffee - For object trajectory

```
'use strict'


{min, max} = Math

require '../helpers'

LanePosition = require './lane-position'

Curve = require '../geom/curve'

_ = require 'underscore'


class Trajectory

  constructor: (@car, lane, position) ->

    position ?= 0

    @current = new LanePosition @car, lane, position

    @current.acquire()

    @next = new LanePosition @car

    @temp = new LanePosition @car

    @isChangingLanes = false


  @property 'lane',

    get: -> @temp.lane or @current.lane


  @property 'absolutePosition',

    get: -> if @temp.lane? then @temp.position else @current.position
```

```
  @property 'relativePosition',
    get: -> @absolutePosition / @lane.length

  @property 'direction',
    get: -> @lane.getDirection @relativePosition

  @property 'coords',
    get: -> @lane.getPoint @relativePosition

  @property 'nextCarDistance',
    get: ->
      a = @current.nextCarDistance
      b = @next.nextCarDistance
      if a.distance < b.distance then a else b

  @property 'distanceToStopLine',
    get: ->
      return @getDistanceToIntersection() if not @canEnterIntersection()
      return Infinity

  @property 'nextIntersection',
    get: -> @current.lane.road.target
```

```
  @property 'previousIntersection',
    get: -> @current.lane.road.source

  isValidTurn: ->
    #TODO right turn is only allowed from the right lane
    nextLane = @car.nextLane
    sourceLane = @current.lane
    throw Error 'no road to enter' unless nextLane
    turnNumber = sourceLane.getTurnDirection nextLane
    throw Error 'no U-turns are allowed' if turnNumber is 3
    if turnNumber is 0 and not sourceLane.isLeftmost
      throw Error 'no left turns from this lane'
    if turnNumber is 2 and not sourceLane.isRightmost
      throw Error 'no right turns from this lane'
    return true

  canEnterIntersection: ->
    nextLane = @car.nextLane
    sourceLane = @current.lane
    return true unless nextLane
    intersection = @nextIntersection
    turnNumber = sourceLane.getTurnDirection nextLane
```

```
    sideId = sourceLane.road.targetSideId
    intersection.controlSignals.state[sideId][turnNumber]

  getDistanceToIntersection: ->
    distance = @current.lane.length - @car.length / 2 - @current.position
    if not @isChangingLanes then max distance, 0 else Infinity

  timeToMakeTurn: (plannedStep = 0) ->
    @getDistanceToIntersection() <= plannedStep

  moveForward: (distance) ->
    distance = max distance, 0
    @current.position += distance
    @next.position += distance
    @temp.position += distance
    if @timeToMakeTurn() and @canEnterIntersection() and @isValidTurn()
      @_startChangingLanes @car.popNextLane(), 0
    tempRelativePosition = @temp.position / @temp.lane?.length
    gap = 2 * @car.length
    if @isChangingLanes and @temp.position > gap and not @current.free
      @current.release()
    if @isChangingLanes and @next.free and
    @temp.position + gap > @temp.lane?.length
```

```
    @next.acquire()
  if @isChangingLanes and tempRelativePosition >= 1
    @_finishChangingLanes()
  if @current.lane and not @isChangingLanes and not @car.nextLane
    @car.pickNextLane()

changeLane: (nextLane) ->
  throw Error 'already changing lane' if @isChangingLanes
  throw Error 'no next lane' unless nextLane?
  throw Error 'next lane == current lane' if nextLane is @lane
  throw Error 'not neighbouring lanes' unless @lane.road is nextLane.road
  nextPosition = min @current.position + 3 * @car.length, @lane.length-1
  @_startChangingLanes nextLane, nextPosition
  # else
  #   throw Error 'too late to change lane' unless nextPosition < @lane.length

_getIntersectionLaneChangeCurve: ->

_getAdjacentLaneChangeCurve: ->
  p1 = @current.lane.getPoint @current.relativePosition
  p2 = @next.lane.getPoint @next.relativePosition
  distance = p2.subtract(p1).length
  direction1 = @current.lane.middleLine.vector.normalized.mult distance * 0.3
```

```
    control1 = p1.add direction1
    direction2 = @next.lane.middleLine.vector.normalized.mult distance * 0.3
    control2 = p2.subtract direction2
    curve = new Curve p1, p2, control1, control2

  _getCurve: ->
    # FIXME: race condition due to using relativePosition on intersections
    @_getAdjacentLaneChangeCurve()

  _startChangingLanes: (nextLane, nextPosition) ->
    throw Error 'already changing lane' if @isChangingLanes
    throw Error 'no next lane' unless nextLane?
    @isChangingLanes = true
    @next.lane = nextLane
    @next.position = nextPosition

    curve = @_getCurve()

    @temp.lane = curve
    @temp.position = 0 # @current.lane.length - @current.position
    @next.position -= @temp.lane.length

  _finishChangingLanes: ->
```

```
    throw Error 'no lane changing is going on' unless @isChangingLanes
    @isChangingLanes = false
    # TODO swap current and next
    @current.lane = @next.lane
    @current.position = @next.position or 0
    @current.acquire()
    @next.lane = null
    @next.position = NaN
    @temp.lane = null
    @temp.position = NaN
    @current.lane

  release: ->
    @current?.release()
    @next?.release()
    @temp?.release()

module.exports = Trajectory
```

model/world.coffee - For object world

```
'use strict'


{random} = Math

require '../helpers'

_ = require 'underscore'

Car = require './car'

Intersection = require './intersection'

Road = require './road'

Pool = require './pool'

Rect = require '../geom/rect'

settings = require '../settings'


class World

  constructor: ->

    @set {}


  @property 'instantSpeed',

    get: ->

      speeds = _.map @cars.all(), (car) -> car.speed

      return 0 if speeds.length is 0

      return (_.reduce speeds, (a, b) -> a + b) / speeds.length
```

```
  set: (obj) ->
    obj ?= {}
    @intersections = new Pool Intersection, obj.intersections
    @roads = new Pool Road, obj.roads
    @cars = new Pool Car, obj.cars
    @carsNumber = 0
    @time = 0

  save: ->
    data = _.extend {}, this
    delete data.cars
    localStorage.world = JSON.stringify data

  load: (data) ->
    data = data or localStorage.world
    data = data and JSON.parse data
    return unless data?
    @clear()
    @carsNumber = data.carsNumber or 0
    for id, intersection of data.intersections
      @addIntersection Intersection.copy intersection
    for id, road of data.roads
      road = Road.copy road
```

```
    road.source = @getIntersection road.source
    road.target = @getIntersection road.target
    @addRoad road

  generateMap: (minX = -4, maxX = 5, minY = -2, maxY = 2) ->
    @clear()
    intersectionsNumber = (0.8 * (maxX - minX + 1) * (maxY - minY + 1)) | 0
    map = {}
    gridSize = settings.gridSize
    step = 3 * gridSize
    @carsNumber = 100
    @timer = 0

    # while intersectionsNumber > 0
    #   x = _.random minX, maxX
    #   y = _.random minY, maxY
    #   console.log('x=', x, '|y=', y)
    #   unless map[[x, y]]?
    #     rect = new Rect step * x, step * y, gridSize, gridSize
    #     intersection = new Intersection rect
    #     @addIntersection map[[x, y]] = intersection
    #     intersectionsNumber -= 1
    intersectionXY = [
```

```
  [6,-3],
  [-8,-2], [-7,-2], [-3,-2], [-2,-2], [2,-2], [4,-2],[5,-2],[6,-2], [7,-2]
  [-2,-1], [2,-1],
  [0,0], [4,0], [5,0], [6,0],
  [-7,1], [-3,1],
  [-8,4], [-7,4], [6,4],[7,4],
  [6,5]
]
for p in intersectionXY
  x = p[0]
  y = p[1]
  rect = new Rect step * x, step * y, gridSize, gridSize
  intersection = new Intersection rect
  @addIntersection map[[x, y]] = intersection

# dunearn road
# @addRoad new Road map[[-8,-2]], map[[-7,-2]], 3
# @addRoad new Road map[[-7,-2]], map[[6,-2]], 3
# @addRoad new Road map[[6,-2]], map[[7,-2]], 3
# bukit timah road
@roadBukitTimah1 = new Road map[[7,-2]], map[[6,-2]], 3
@addRoad @roadBukitTimah1
@roadBukitTimah2 = new Road map[[6,-2]], map[[5,-2]], 3
```

```
@addRoad @roadBukitTimah2

@roadBukitTimah3 = new Road map[[5,-2]], map[[4,-2]], 3

@addRoad @roadBukitTimah3

@roadBukitTimah4 = new Road map[[4,-2]], map[[2,-2]], 3

@addRoad @roadBukitTimah4

@addRoad new Road map[[2,-2]], map[[-2,-2]], 3

@addRoad new Road map[[-2,-2]], map[[-3,-2]], 3

@addRoad new Road map[[-3,-2]], map[[-7,-2]], 3

@addRoad new Road map[[-7,-2]], map[[-8,-2]], 3

# holland road

@addRoad new Road map[[-8,4]], map[[-7,4]], 4

@addRoad new Road map[[-7,4]], map[[-8,4]], 4

@roadHolland3 = new Road map[[-7,4]], map[[6,4]], 4

@addRoad @roadHolland3

@roadHolland4 = new Road map[[6,4]], map[[-7,4]], 4

@addRoad @roadHolland4

@addRoad new Road map[[6,4]], map[[7,4]], 4

@addRoad new Road map[[7,4]], map[[6,4]], 4

# sixth aventh

@addRoad new Road map[[-7,-2]], map[[-7,1]], 2

@addRoad new Road map[[-7,1]], map[[-7,-2]], 2

@addRoad new Road map[[-7,1]], map[[-7,4]], 2

@roadSixAve4 = new Road map[[-7,4]], map[[-7,1]], 2
```

```
@addRoad @roadSixAve4

# adam road

@roadAdam1 = new Road map[[6,-3]], map[[6,-2]], 4

@addRoad @roadAdam1

@addRoad new Road map[[6,-2]], map[[6,-3]], 4

@addRoad new Road map[[6,-2]], map[[6,0]], 4

@roadAdam4 = new Road map[[6,0]], map[[6,-2]], 4

@addRoad @roadAdam4

@roadAdam5 = new Road map[[6,0]], map[[6,4]], 4

@addRoad @roadAdam5

@addRoad new Road map[[6,4]], map[[6,0]], 4

@addRoad new Road map[[6,4]], map[[6,5]], 4

@addRoad new Road map[[6,5]], map[[6,4]], 4

# namly road

@addRoad new Road map[[-3,-2]], map[[-3,1]], 1

@roadNamlyRoad2 = new Road map[[-3,1]], map[[-3,-2]], 1

@addRoad @roadNamlyRoad2

@addRoad new Road map[[-3,1]], map[[-7,1]], 1

@roadNamlyRoad4 = new Road map[[-7,1]], map[[-3,1]], 1

@addRoad @roadNamlyRoad4

# hci circular

#@addRoad new Road map[[-2,-2]], map[[-2,-1]], 1

@addRoad new Road map[[-2,-1]], map[[-2,-2]], 1
```

```
#@addRoad new Road map[[-2,-1]], map[[0,0]], 1
@addRoad new Road map[[0,0]], map[[-2,-1]], 1

@addRoad new Road map[[2,-2]], map[[2,-1]], 1
#@addRoad new Road map[[2,-1]], map[[2,-2]], 1
@addRoad new Road map[[2,-1]], map[[0,0]], 1
#@addRoad new Road map[[0,0]], map[[2,-1]], 1
# king's road
@addRoad new Road map[[4,-2]], map[[4,0]], 1
@addRoad new Road map[[4,0]], map[[4,-2]], 1
@addRoad new Road map[[4,0]], map[[5,0]], 1
@addRoad new Road map[[5,0]], map[[4,0]], 1
# queen's road
@addRoad new Road map[[5,-2]], map[[5,0]], 1
@addRoad new Road map[[5,0]], map[[5,-2]], 1
@addRoad new Road map[[5,0]], map[[6,0]], 1
@addRoad new Road map[[6,0]], map[[5,0]], 1
# for x in [minX..maxX]
#   previous = null
#   for y in [minY..maxY]
#     intersection = map[[x, y]]
#     if intersection?
#       if random() < 0.9
```

```
  #         @addRoad new Road intersection, previous if previous?
  #         @addRoad new Road previous, intersection if previous?
  #       previous = intersection
  # for y in [minY..maxY]
  #   previous = null
  #   for x in [minX..maxX]
  #     intersection = map[[x, y]]
  #     if intersection?
  #       if random() < 0.9
  #         @addRoad new Road intersection, previous if previous?
  #         @addRoad new Road previous, intersection if previous?
  #       previous = intersection
  null



clear: ->
  @set {}


onTick: (delta) =>
  throw Error 'delta > 1' if delta > 1
  @time += delta
  @refreshCars()
  for id, intersection of @intersections.all()
```

```
    intersection.controlSignals.onTick delta
  for id, car of @cars.all()
    car.move delta
    @removeCar car unless car.alive

refreshCars: ->
  @addCar new Car @roadNamlyRoad2.leftmostLane
  @timer = (@timer + 1) % 4
  if @timer == 0
    @addCar new Car @roadNamlyRoad4.leftmostLane
    @addCar new Car _.sample @roadBukitTimah2.lanes
    @addCar new Car _.sample @roadBukitTimah4.lanes
    # @addCar new Car @roadBukitTimah4.leftmostLane
    # @addCar new Car @roadSixAve4.leftmostLane
    # @addCar new Car _.sample @roadAdam1.lanes
    @addCar new Car _.sample @roadAdam4.lanes
    @addCar new Car _.sample @roadAdam5.lanes
    @addCar new Car _.sample @roadHolland3.lanes
    @addCar new Car _.sample @roadHolland4.lanes
  # @addCar new Car @roadAdam4.leftmostLane
  @addRandomCar() if @cars.length < @carsNumber
  @removeRandomCar() if @cars.length > @carsNumber
```

```
  addRoad: (road) ->
    @roads.put road
    road.source.roads.push road
    road.target.inRoads.push road
    road.update()

  getRoad: (id) ->
    @roads.get id

  addCar: (car) ->
    @cars.put car

  getCar: (id) ->
    @cars.get(id)

  removeCar: (car) ->
    @cars.pop car

  addIntersection: (intersection) ->
    @intersections.put intersection

  getIntersection: (id) ->
    @intersections.get id
```

```
  addRandomCar: ->
    road = _.sample @roads.all()
    if road?
      lane = _.sample road.lanes
      @addCar new Car lane if lane?

  removeRandomCar: ->
    car = _.sample @cars.all()
    if car?
      @removeCar car

module.exports = World
```

visualizer/graphics.coffee - For drawing graphics

```
'use strict'


{PI} = Math

require '../helpers.coffee'


class Graphics

  constructor: (@ctx) ->


  fillRect: (rect, style, alpha) ->

    @ctx.fillStyle = style if style?

    _alpha = @ctx.globalAlpha

    @ctx.globalAlpha = alpha if alpha?

    @ctx.fillRect rect.left(), rect.top(), rect.width(), rect.height()

    @ctx.globalAlpha = _alpha


  drawRect: (rect) ->

    @ctx.beginPath

    vertices = rect.getVertices()

    @ctx.beginPath()

    @moveTo vertices[0]

    @lineTo point for point in vertices[1..]

    @ctx.closePath()
```

```
  drawImage: (image, rect) ->
    @ctx.drawImage image, rect.left(), rect.top(), rect.width(), rect.height()

  clear: (color) ->
    @ctx.fillStyle = color
    @ctx.fillRect 0, 0, @ctx.canvas.width, @ctx.canvas.height

  moveTo: (point) ->
    @ctx.moveTo point.x, point.y

  lineTo: (point) ->
    @ctx.lineTo point.x, point.y

  drawLine: (source, target) ->
    @ctx.beginPath()
    @moveTo source
    @lineTo target

  drawSegment: (segment) ->
    @drawLine segment.source, segment.target

  drawCurve: (curve, width, color) ->
```

```
    pointsNumber = 10
    @ctx.lineWidth = width
    @ctx.beginPath()
    @moveTo curve.getPoint 0
    for i in [0..pointsNumber]
      point = curve.getPoint i / pointsNumber
      @lineTo point
    if curve.O
      @moveTo curve.O
      @ctx.arc curve.O.x, curve.O.y, width, 0, 2 * PI
    if curve.Q
      @moveTo curve.Q
      @ctx.arc curve.Q.x, curve.Q.y, width, 0, 2 * PI
    @stroke color if color

  drawTriangle: (p1, p2, p3) ->
    @ctx.beginPath()
    @moveTo p1
    @lineTo p2
    @lineTo p3

  fill: (style, alpha) ->
    @ctx.fillStyle = style
```

```
    _alpha = @ctx.globalAlpha
    @ctx.globalAlpha = alpha if alpha?
    @ctx.fill()
    @ctx.globalAlpha = _alpha

  stroke: (style) ->
    @ctx.strokeStyle = style
    @ctx.stroke()

  polyline: (points...) ->
    if points.length >= 1
      @ctx.beginPath()
      @moveTo points[0]
      for point in points[1..]
        @lineTo point
      @ctx.closePath()
  save: ->
    @ctx.save()
  restore: ->
    @ctx.restore()

module.exports = Graphics
```

visualizer/highlighter.coffee - For object highlighter

```
'use strict'


require '../helpers.coffee'
Tool = require './tool.coffee'
settings = require '../settings.coffee'


class ToolHighlighter extends Tool
  constructor: ->
    super arguments...
    @hoveredCell = null


  mousemove: (e) =>
    cell = @getCell e
    hoveredIntersection = @getHoveredIntersection cell
    @hoveredCell = cell
    for id, intersection of @visualizer.world.intersections.all()
      intersection.color = null
    if hoveredIntersection?
      hoveredIntersection.color = settings.colors.hoveredIntersection


  mouseout: =>
    @hoveredCell = null
```

```
  draw: =>
    if @hoveredCell
      color = settings.colors.hoveredGrid
      @visualizer.graphics.fillRect @hoveredCell, color, 0.5


module.exports = ToolHighlighter
```

visualizer/intersection-builder.coffee - For object intersection builder

```
'use strict'


require '../helpers.coffee'
Tool = require './tool.coffee'
Intersection = require '../model/intersection.coffee' # TODO: decouple


class ToolIntersectionBuilder extends Tool
  constructor: ->
    super arguments...
    @tempIntersection = null
    @mouseDownPos = null


  mousedown: (e) =>
    @mouseDownPos = @getCell e
    if e.shiftKey
      @tempIntersection = new Intersection @mouseDownPos
      e.stopImmediatePropagation()


  mouseup: =>
    if @tempIntersection
      @visualizer.world.addIntersection @tempIntersection
      @tempIntersection = null
```

```
    @mouseDownPos = null

  mousemove: (e) =>
    if @tempIntersection
      rect = @visualizer.zoomer.getBoundingBox @mouseDownPos, @getCell e
      @tempIntersection.rect = rect

  mouseout: =>
    @mouseDownPos = null
    @tempIntersection = null

  draw: =>
    if @tempIntersection
      @visualizer.drawIntersection @tempIntersection, 0.4

module.exports = ToolIntersectionBuilder
```

visualizer/intersection-mover.coffee - For object intersection mover

```
'use strict'


require '../helpers.coffee'
Tool = require './tool.coffee'


class ToolIntersectionMover extends Tool
  constructor: ->
    super arguments...
    @intersection = null


  mousedown: (e) =>
    intersection = @getHoveredIntersection @getCell e
    if intersection
      @intersection = intersection
      e.stopImmediatePropagation()


  mouseup: =>
    @intersection = null


  mousemove: (e) =>
    if @intersection
      cell = @getCell e
```

```
      @intersection.rect.left(cell.x)
      @intersection.rect.top(cell.y)
      @intersection.update()

  mouseout: =>
    @intersection = null

module.exports = ToolIntersectionMover
```

visualizer/mover.coffee - For object mover

```
'use strict'


require '../helpers.coffee'
Tool = require './tool.coffee'


class Mover extends Tool
  constructor: ->
    super arguments...
    @startPosition = null


  contextmenu: ->
    false


  mousedown: (e) =>
    @startPosition = @getPoint e
    e.stopImmediatePropagation()


  mouseup: =>
    @startPosition = null


  mousemove: (e) =>
    if @startPosition
```

```
      offset = @getPoint(e).subtract(@startPosition)
      @visualizer.zoomer.moveCenter offset
      @startPosition = @getPoint e

  mouseout: =>
    @startPosition = null

module.exports = Mover
```

visualizer/road-builder.coffee - For object road builder

```
'use strict'


require '../helpers.coffee'
Tool = require './tool.coffee'
Road = require '../model/road.coffee' #TODO decouple


class ToolRoadBuilder extends Tool
  constructor: ->
    super arguments...
    @sourceIntersection = null
    @road = null
    @dualRoad = null


  mousedown: (e) =>
    cell = @getCell e
    hoveredIntersection = @getHoveredIntersection cell
    if e.shiftKey and hoveredIntersection?
      @sourceIntersection = hoveredIntersection
      e.stopImmediatePropagation()


  mouseup: (e) =>
    @visualizer.world.addRoad @road if @road?
```

```
    @visualizer.world.addRoad @dualRoad if @dualRoad?
    @road = @dualRoad = @sourceIntersection = null
  mousemove: (e) =>
    cell = @getCell e
    hoveredIntersection = @getHoveredIntersection cell
    if (@sourceIntersection and hoveredIntersection and
    @sourceIntersection.id isnt hoveredIntersection.id)
      if @road?
        @road.target = hoveredIntersection
        @dualRoad.source = hoveredIntersection
      else
        @road = new Road @sourceIntersection, hoveredIntersection
        @dualRoad = new Road hoveredIntersection, @sourceIntersection
    else
      @road = @dualRoad = null

  mouseout: (e) =>
    @road = @dualRoad = @sourceIntersection = null

  draw: =>
    @visualizer.drawRoad @road, 0.4 if @road?
    @visualizer.drawRoad @dualRoad, 0.4 if @dualRoad?
module.exports = ToolRoadBuilder
```

visualizer/tool.coffee - For object tool

```
'use strict'


require '../helpers.coffee'
$ = require 'jquery'
_ = require 'underscore'
Point = require '../geom/point.coffee'
Rect = require '../geom/rect.coffee'
require('jquery-mousewheel') $


METHODS = [
  'click'
  'mousedown'
  'mouseup'
  'mousemove'
  'mouseout'
  'mousewheel'
  'contextmenu'
]


class Tool
  constructor: (@visualizer, autobind) ->
    @ctx = @visualizer.ctx
```

```
  @canvas = @ctx.canvas
  @isBound = false
  @bind() if autobind

bind: ->
  @isBound = true
  for method in METHODS when @[method]?
    $(@canvas).on method, @[method]

unbind: ->
  @isBound = false
  for method in METHODS when @[method]?
    $(@canvas).off method, @[method]

toggleState: ->
  if @isBound then @unbind() else @bind()

draw: ->

getPoint: (e) ->
  new Point e.pageX - @canvas.offsetLeft, e.pageY - @canvas.offsetTop

getCell: (e) ->
```

```
    @visualizer.zoomer.toCellCoords @getPoint e


  getHoveredIntersection: (cell) ->
    intersections = @visualizer.world.intersections.all()
    for id, intersection of intersections
      return intersection if intersection.rect.containsRect cell


module.exports = Tool
```

visualizer/visualizer.coffee - For object visualizer

```
'use strict'


{PI} = Math

require '../helpers'

$ = require 'jquery'

_ = require 'underscore'

chroma = require 'chroma-js'

Point = require '../geom/point'

Rect = require '../geom/rect'

Graphics = require './graphics'

ToolMover = require './mover'

ToolIntersectionMover = require './intersection-mover'

ToolIntersectionBuilder = require './intersection-builder'

ToolRoadBuilder = require './road-builder'

ToolHighlighter = require './highlighter'

Zoomer = require './zoomer'

settings = require '../settings'


class Visualizer

  constructor: (@world) ->

    @$canvas = $('#canvas')

    @canvas = @$canvas[0]
```

```
    @ctx = @canvas.getContext('2d')


    @carImage = new Image()
    @carImage.src = 'images/car.png'


    @updateCanvasSize()
    @zoomer = new Zoomer 4, this, true
    @graphics = new Graphics @ctx
    @toolRoadbuilder = new ToolRoadBuilder this, true
    @toolIntersectionBuilder = new ToolIntersectionBuilder this, true
    @toolHighlighter = new ToolHighlighter this, true
    @toolIntersectionMover = new ToolIntersectionMover this, true
    @toolMover = new ToolMover this, true
    @_running = false
    @previousTime = 0
    @timeFactor = settings.defaultTimeFactor
    @debug = false


  drawIntersection: (intersection, alpha) ->
    color = intersection.color or settings.colors.intersection
    @graphics.drawRect intersection.rect
    @ctx.lineWidth = 0.4
    @graphics.stroke settings.colors.roadMarking
```

```
    @graphics.fillRect intersection.rect, color, alpha


  drawSignals: (road) ->
    lightsColors = [settings.colors.redLight, settings.colors.greenLight]
    intersection = road.target
    segment = road.targetSide
    sideId = road.targetSideId
    lights = intersection.controlSignals.state[sideId]


    @ctx.save()
    @ctx.translate segment.center.x, segment.center.y
    @ctx.rotate (sideId + 1) * PI / 2
    @ctx.scale 1 * segment.length, 1 * segment.length
    # map lane ending to [(0, -0.5), (0, 0.5)]
    if lights[0]
      @graphics.drawTriangle(
        new Point(0.1, -0.2),
        new Point(0.2, -0.4),
        new Point(0.3, -0.2)
      )
      @graphics.fill settings.colors.greenLight
    if lights[1]
      @graphics.drawTriangle(
```

```
      new Point(0.3, -0.1),
      new Point(0.5, 0),
      new Point(0.3, 0.1)
    )
    @graphics.fill settings.colors.greenLight
  if lights[2]
    @graphics.drawTriangle(
      new Point(0.1, 0.2),
      new Point(0.2, 0.4),
      new Point(0.3, 0.2)
    )
    @graphics.fill settings.colors.greenLight
  @ctx.restore()
  if @debug
    @ctx.save()
    @ctx.fillStyle = "black"
    @ctx.font = "1px Arial"
    center = intersection.rect.center()
    flipInterval = Math.round(intersection.controlSignals.flipInterval * 100) / 100
    phaseOffset = Math.round(intersection.controlSignals.phaseOffset * 100) / 100
    @ctx.fillText flipInterval + ' ' + phaseOffset, center.x, center.y
    @ctx.restore()
```

```
drawRoad: (road, alpha) ->
  throw Error 'invalid road' if not road.source? or not road.target?
  sourceSide = road.sourceSide
  targetSide = road.targetSide

  @ctx.save()
  @ctx.lineWidth = 0.4
  leftLine = road.leftmostLane.leftBorder
  @graphics.drawSegment leftLine
  @graphics.stroke settings.colors.roadMarking

  rightLine = road.rightmostLane.rightBorder
  @graphics.drawSegment rightLine
  @graphics.stroke settings.colors.roadMarking
  @ctx.restore()

  @graphics.polyline sourceSide.source, sourceSide.target,
  targetSide.source, targetSide.target
  @graphics.fill settings.colors.road, alpha

  @ctx.save()
  for lane in road.lanes[1..]
    line = lane.rightBorder
```

```
    dashSize = 1
    @graphics.drawSegment line
    @ctx.lineWidth = 0.2
    @ctx.lineDashOffset = 1.5 * dashSize
    @ctx.setLineDash [dashSize]
    @graphics.stroke settings.colors.roadMarking
  @ctx.restore()

drawCar: (car) ->
  angle = car.direction
  center = car.coords
  rect = new Rect 0, 0, 1.1 * car.length, 1.7 * car.width
  rect.center new Point 0, 0
  boundRect = new Rect 0, 0, car.length, car.width
  boundRect.center new Point 0, 0

  @graphics.save()
  @ctx.translate center.x, center.y
  @ctx.rotate angle
  l = 0.90 - 0.30 * car.speed / car.maxSpeed
  style = chroma(car.color, 0.8, l, 'hsl').hex()
  # @graphics.drawImage @carImage, rect
  @graphics.fillRect boundRect, style
```

```
    @graphics.restore()
    if @debug
      @ctx.save()
      @ctx.fillStyle = "black"
      @ctx.font = "1px Arial"
      @ctx.fillText car.id, center.x, center.y

      if (curve = car.trajectory.temp?.lane)?
        @graphics.drawCurve curve, 0.1, 'red'
      @ctx.restore()

  drawGrid: ->
    gridSize = settings.gridSize
    box = @zoomer.getBoundingBox()
    return if box.area() >= 2000 * gridSize * gridSize
    sz = 0.4

    for i in [box.left()..box.right()] by gridSize
      for j in [box.top()..box.bottom()] by gridSize
        rect = new Rect i - sz / 2, j - sz / 2, sz, sz
        @graphics.fillRect rect, settings.colors.gridPoint

  updateCanvasSize: ->
```

```
    if @$canvas.attr('width') isnt $(window).width or
    @$canvas.attr('height') isnt $(window).height
      @$canvas.attr
        width: $(window).width()
        height: $(window).height()

  draw: (time) =>
    delta = (time - @previousTime) || 0
    if delta > 30
      delta = 100 if delta > 100
      @previousTime = time
      @world.onTick @timeFactor * delta / 1000
      @updateCanvasSize()
      @graphics.clear settings.colors.background
      @graphics.save()
      @zoomer.transform()
      @drawGrid()
      for id, intersection of @world.intersections.all()
        @drawIntersection intersection, 0.9
      @drawRoad road, 0.9 for id, road of @world.roads.all()
      @drawSignals road for id, road of @world.roads.all()
      @drawCar car for id, car of @world.cars.all()
      @toolIntersectionBuilder.draw() # TODO: all tools
```

```
    @toolRoadbuilder.draw()
    @toolHighlighter.draw()
    @graphics.restore()
  window.requestAnimationFrame @draw if @running

@property 'running',
  get: -> @_running
  set: (running) ->
    if running then @start() else @stop()

start: ->
  unless @_running
    @_running = true
    @draw()

stop: ->
  @_running = false

module.exports = Visualizer
```

visualizer/zoomer.coffee - For object zoomer

```
'use strict'


{min, max} = Math

require '../helpers.coffee'

Point = require '../geom/point.coffee'

Rect = require '../geom/rect.coffee'

Tool = require './tool.coffee'

settings = require '../settings.coffee'


class Zoomer extends Tool

  constructor: (@defaultZoom, @visualizer, args...) ->

    super @visualizer, args...

    @ctx = @visualizer.ctx

    @canvas = @ctx.canvas

    @_scale = 1

    @screenCenter = new Point @canvas.width / 2, @canvas.height / 2

    @center = new Point @canvas.width / 2, @canvas.height / 2


  @property 'scale',

    get: -> @_scale

    set: (scale) -> @zoom scale, @screenCenter
```

```
  toCellCoords: (point) ->
    gridSize = settings.gridSize
    centerOffset = point.subtract(@center).divide(@scale)
    x = centerOffset.x // (@defaultZoom * gridSize) * gridSize
    y = centerOffset.y // (@defaultZoom * gridSize) * gridSize
    new Rect x, y, gridSize, gridSize

  getBoundingBox: (cell1, cell2) ->
    cell1 ?= @toCellCoords new Point 0, 0
    cell2 ?= @toCellCoords new Point @canvas.width, @canvas.height
    x1 = cell1.x
    y1 = cell1.y
    x2 = cell2.x
    y2 = cell2.y
    xMin = min cell1.left(), cell2.left()
    xMax = max cell1.right(), cell2.right()
    yMin = min cell1.top(), cell2.top()
    yMax = max cell1.bottom(), cell2.bottom()
    new Rect xMin, yMin, xMax - xMin, yMax - yMin

  transform: ->
    @ctx.translate @center.x, @center.y
    k = @scale * @defaultZoom
```

```
    @ctx.scale k, k


  zoom: (k, zoomCenter) ->
    k ?= 1
    offset = @center.subtract zoomCenter
    @center = zoomCenter.add offset.mult k / @_scale
    @_scale = k


  moveCenter: (offset) ->
    @center = @center.add offset


  mousewheel: (e) =>
    offset = e.deltaY * e.deltaFactor
    zoomFactor = 2 ** (0.001 * offset)
    @zoom @scale * zoomFactor, @getPoint e
    e.preventDefault()


module.exports = Zoomer
```

app.coffee - For object app

```
'use strict'


require './helpers'
$ = require 'jquery'
_ = require 'underscore'
Visualizer = require './visualizer/visualizer'
DAT = require 'dat-gui'
World = require './model/world'
settings = require './settings'


$ ->
  canvas = $('<canvas />', {id: 'canvas'})
  $(document.body).append(canvas)


  window.world = new World()
  world.load()
  if world.intersections.length is 0
    world.generateMap()
    world.carsNumber = 100
  window.visualizer = new Visualizer world
  visualizer.start()
  gui = new DAT.GUI()
```

```
guiWorld = gui.addFolder 'world'
guiWorld.open()
guiWorld.add world, 'save'
guiWorld.add world, 'load'
guiWorld.add world, 'clear'
guiWorld.add world, 'generateMap'
guiVisualizer = gui.addFolder 'visualizer'
guiVisualizer.open()
guiVisualizer.add(visualizer, 'running').listen()
guiVisualizer.add(visualizer, 'debug').listen()
guiVisualizer.add(visualizer.zoomer, 'scale', 0.1, 2).listen()
guiVisualizer.add(visualizer, 'timeFactor', 0.1, 10).listen()
guiWorld.add(world, 'carsNumber').min(0).max(200).step(1).listen()
guiWorld.add(world, 'instantSpeed').step(0.00001).listen()
gui.add(settings, 'lightsFlipInterval', 0, 400, 0.01).listen()
```

runner.coffee - For object runner

```
#!/usr/bin/env coffee

'use strict'


require './helpers'

World = require './model/world'

_ = require 'underscore'

settings = require './settings'

fs = require 'fs'


measureAverageSpeed = (setupCallback) ->

  world = new World()

  map = fs.readFileSync './experiments/map.json', {encoding: 'utf8'}

  console.log map

  # world.generateMap()

  world.load map

  world.carsNumber = 50

  setupCallback?(world)

  results = []

  for i in [0..10000]

    world.onTick 0.2

    # console.log world.instantSpeed

    results.push world.instantSpeed
```

```
  (results.reduce (a, b) -> a + b) / results.length



getParams = (world) ->
  params = (i.controlSignals.flipMultiplier for id, i of world.intersections.all())
  # console.log JSON.stringify(params)
  params


settings.lightsFlipInterval = 160


experiment1 = () ->
  out = fs.createWriteStream './experiments/1.data'
  out.write 'multiplier avg_speed\n'
  for multiplier in [0.0001, 0.01, 0.02, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 2, 3, 4, 5]
    do (multiplier) ->
      result = measureAverageSpeed (world) ->
        i.controlSignals.flipMultiplier = multiplier for id, i of world.intersections.all()
        getParams world
      out.write(multiplier + ' ' +  result + '\n')


experiment2 = () ->
  out = fs.createWriteStream './experiments/2.data'
  out.write 'it avg_speed\n'
```

```
  for it in [0..9]
    result = measureAverageSpeed (world) ->
      i.controlSignals.flipMultiplier = Math.random() for id, i of world.intersections.all()
      getParams world
    out.write(it + ' ' +  result + '\n')


experiment3 = () ->
  out = fs.createWriteStream './experiments/3.data'
  out.write 'it avg_speed\n'
  for it in [0..10]
    result = measureAverageSpeed (world) ->
      i.controlSignals.flipMultiplier = 1 for id, i of world.intersections.all()
      i.controlSignals.phaseOffset = 0
      getParams world
    out.write(it + ' ' +  result + '\n')


# experiment1()
# experiment2()
# experiment3()
```

settings.coffee - For object settings

```
'use strict'


settings =
  colors:
    background: '#97a1a1'
    redLight: 'hsl(0, 100%, 50%)'
    greenLight: '#85ee00'
    intersection: '#586970'
    road: '#586970'
    roadMarking: '#bbb'
    hoveredIntersection: '#3d4c53'
    tempRoad: '#aaa'
    gridPoint: '#586970'
    grid1: 'rgba(255, 255, 255, 0.5)'
    grid2: 'rgba(220, 220, 220, 0.5)'
    hoveredGrid: '#f4e8e1'
  fps: 30
  lightsFlipInterval: 160
  gridSize: 8
  defaultTimeFactor: 5


module.exports = settings
```