Department of Electrical and Computer Engineering

Part IV Research Projects

Interim Report

A design environment for Intelligent Transportation Systems using Cities: Skylines

Prepared by Jarrod van den Heuvel

Project 25

Project partner Jake H. Thomas

Supervised by Dr. Partha Roop

Co-supervised by Dr. Nasser Giacaman

Declaration of Originality

This report is my own unaided work and was not copied from anyone or anywhere nor written in collaboration with any other person.

Jarrod van den Heuvel

18 September 2017

A design environment for Intelligent Transportation Systems using Cities: Skylines

Jarrod van den Heuvel

Department of Electrical and Computer Engineering, University of Auckland, New Zealand

***Abstract*—Big cities all around the world suffer from traffic congestion which has major social and economic costs. There are new traffic light algorithms being developed that are supposedly better than the current systems as they leverage more advanced technology that has been developed since the creation of the current systems. These new algorithms need a platform to be presented on in a city without the risk of implementing them in the real world. The report proposes a modification to the city simulation game, Cities: Skylines as a tool for the simulation of different traffic light algorithms in a city wide environment. Different adaptive ITS traffic light algorithms and non-adaptive traffic light algorithms were tested and compared with each other based on their ability to accommodate traffic flow. The results of these tests have been analysed and graphed. It was concluded that through the modification, Cities: Skylines is an effective tool for fast real time visualisation and presentation of traffic light algorithms in a city wide micro simulation while also giving an indicator of an algorithms viability.**

I. Introduction

Traffic is an issue which must be dealt with by every major city worldwide and it is guaranteed that if this issue is not addressed properly, a city will face major problems both economically and socially. Auckland in particular has not addressed the traffic issue well and it is estimated that costs of congestion in Auckland are around $1 Billion per year [1]. Therefore it can be concluded that changes must be made to the way a city with an increasing population like Auckland deal with traffic. One such solution is to change the traffic light algorithm, to another more optimised algorithm that is up to date with current and potential future technologies available. There are currently existing Intelligent Transport System (ITS) traffic light algorithms which take into account modern technology that did not exist when the current traffic lights in Auckland were implemented. Some of these ITS algorithms have been theoretically proven to perform better than what is currently implemented in Auckland when tested in a controlled small scale simulation. However, validating a traffic algorithm theoretically on a small scale and validating it in an entire city can produce vastly different results. It expensive and risky to implement a new traffic algorithm city wide that has not been implemented anywhere before and whose success is yet to be proven. Therefore, these ITS algorithms should be presented in an environment that simulates a city in order to get as close to a city wide implementation as possible without having to physically implement them. The goal of this project is to investigate different technologies in order to develop a framework for implementing and comparing different traffic algorithms in a city wide simulation so that the impact these algorithms have on the traffic of an entire simulated city can be analysed and visualised clearly. This report focuses on the development and implementation of three different traffic algorithms within the city simulation computer game, Cities: Skylines. Traffic data was recorded for each of these algorithms and conclusions were drawn regarding the individual algorithm. Conclusions were also drawn about the usefulness of the game as a city simulator for the current implementation and any future work.

II. Literature Review

A. Existing Traffic Systems

Adaptive Traffic Control Systems (ATCSs) adjust signal timing based on the current traffic conditions, demand, and system capacity in real time. Surveillance or a method of sensing traffic is required for the operation of ATCSs, usually in the form of inductive loop detectors [2]. The most commonly implemented ATCSs are the Sydney Co-ordinated Adaptive Traffic System (SCATS) and Splits-Cycle-Offsets-Optimization-Technique (SCOOT). SCATS is used in Auckland. SCATS uses data at the intersection, treats traffic as cyclical and will always follow the same phase order with green light time based on the demands of the previous cycle. SCOOT uses data from detectors located at the upstream end of each approach to estimate the structure of traffic platoons for each signal cycle so that it can adjust the timings based on the traffic platoon in order to clear the intersection during the green time, thus minimizing vehicle delays [3].

B. Intelligent Transportation Systems

Intelligent transportation systems apply information, data processing, communication, and sensor technologies to vehicles, transport infrastructure and transport users to increase the effectiveness, environmental performance, safety, resilience and efficiency of the transport system [4]. A few examples of ITSs are: self driving vehicles, electronic road charging and number plate recognition. The type of ITS that this paper is focused on however is the field of adaptive traffic control systems. There have been many proposed concepts around the idea of ITSs being used for ATCSs to improve traffic management. The following examples outline some of these concepts.

- One such concept is a computer systems approach in which vehicles are treated like computer instructions and intersections are treated like processors in a computer. The vehicle movements through intersections are therefore treated in a similar way to how a computer schedules instructions to be

processed which is highly optimised. It utilises ideas such as priority scheduling, starvation prevention and other common concepts found in scheduling algorithms for operating systems [5].

- Another ITS approach relates to the prioritisation of emergency vehicles such as amublances. It functions by equipping emergency vehicles with a transmitter and intersections with a receiver. When an emergency vehicle is within range of an intersection on its path and is in an emergency state it will transmit a signal to the intersection and the intersection will respond by displaying a green light for the emergency vehicle's path [6].

- Deep learning can be used in ITSs for traffic flow prediction. Deep learning algorithms can represent traffic features without prior knowledge by utilising big transportation data. Machine learning is can be done using a stacked autoencoder (SAE) model to learn generic traffic flow features and it is trained in a layerwise greedy fashion. It is claimed that this provides good traffic flow prediction performance [7].

- Another ITS approach is the game theory approach in which the different ends of an intersection compete for green light time based on Cournot's model of duopoly [8].

The previously mentioned approaches to ITS traffic management are only a small subset of the proposed ITS traffic management ideas. Most of these approaches all suffer from the same problem, they have not been implemented in real world scenarios and are yet to be presented in a city wide microscopic simulation and thus it is hard to judge how these traffic light algorithms will affect an entire city.

C. Simulation Tools

There are currently many traffic simulation tools available which can be accurately calibrated with real world systems for macroscopic and/or microscopic simulations, Quadstone Paramics, AIMSUN and PTV VISSIM are just some of these tools. However, these tools come with some setbacks. In the case of AIMSUN, it struggles to simulate many intersections at once without experiencing significant slow down and it is not incredibly aesthetic [9]. A common issue with these tools is that they have expensive annual fees. AIMSUN has been used to prove the effectiveness of the computer systems ITS approach on three intersections running in a microscopic simulation [6]. However, because AIMSUN struggles to scale up to a city wide microscopic environment, another tool must be used to simulate traffic management systems on an entire city while remaining fast, visually appealing and microscopic.

This is where games are looked into as a simulation tool as they are often designed to be visually appealing while maintaining performance and a stable frame rate. Games have been used in the past as a way of simulating and presenting engineering problems and solutions. For example, Grand Theft Auto V has been used for the development and testing of self driving car AI [10]. Cities: Skylines is a city simulation game, see Fig. 1. It appears to be a valid candidate for simulating different traffic algorithms as it is able to simulate an entire city's traffic in a visually appealing way using its own traffic algorithm. In addition, it allows a user to inspect individual intersections at a microscopic level as well as allowing them to view the city as a whole at a macroscopic level all in real time.

Cities: Skylines has a large modding community and has been designed in a way that accommodates modifying the way the game functions. Cities Skylines has been used previously to simulate engineering problems. For example, one study used embedded boards in a test bed to control traffic lights within the game while allowing Cities: Skylines to simulate the rest of the system [11].

Fig. 1. A view of Cities: Skylines

D. Existing mods for Cities: Skylines

Because Cities: Skylines has a large modding community, there are many mods available for download (a mod is a modification to a game which changes the way the game works). By downloading and running some of these mods, an insight was gained into what is possible in terms of modding Cities: Skylines. One mod in particular was extremely useful - Traffic Manager: President Edition [12]. It is a mod for improving traffic flow and it implements the following features: adding traffic lights at junctions; adaptive timed traffic lights dependent on live traffic measurements; adding yield and stop signs to junctions to control vehicle priority; define custom speed limits; prohibit vehicle types from using certain roads; and the enforcement of parking restrictions. Another mod that was researched was cimtographer which is used for importing real world maps geographical data into the game. It can also be used for exporting Cities: Skylines maps into a .osm file which can be used in tools such as AIMSUN. This mod could prove to be useful with regards to calibrating the game simulation with a tool like AIMSUN which is considered to be a trusted traffic simulator.

E. Conclusions from the Literature Review

Overall, it is apparent that ITS traffic light algorithms face the problem of not yet being proven in the real world. Not only does Cities: Skylines have the potential to be a useful city wide traffic simulation and visualisation tool to test ITS traffic algorithms in a city wide micro and macroscopic environment, it also has the potential to be a useful simulator for testing concepts in other ITS fields such as electronic road user charging and self driving vehicle AI as the game allows the tracking and modification of individual vehicles. This is made possible through the use of existing mods and the modding API provided by the game's developers.

III. Understanding Cities: Skylines

To create a mod for Cities: Skylines it is important to understand the structure of the game's code and the typical structure of a mod for the game. Cities: Skylines was developed in Unity using C and in order to understand the code structure, a software tool called JustDecompile [13] was used. JustDecompile decompiled the dll file for the game so that the individual classes and the dependencies could be viewed. It was discovered that there were over 700 classes within the dll which made interpreting and understanding the entire game's code near impossible without comments. It was decided that a mod which utilises many of the game's relevant classes for implementing custom traffic light algorithms should be downloaded and looked into in order to obtain an understanding of how the game's code is used and modified. The code for Traffic Manager: President Edition which was covered in the related work section was downloaded from GitHub [12] along with an MIT license. It was decided that the code for this mod should be built upon to implement custom traffic light algorithms. This is because the mod already provides its own custom traffic light algorithm and therefore can be used as a guide. It was found that the custom traffic light algorithm provided by the mod was quite inflexible and significant change would be required in order to create a framework that facilitates a range of different algorithm types. In Cities: Skylines, sections of road are nodes, an intersection is a type of node. Each node has eight segments, with each segment representing an end of the intersection. Not all segments need to be utilised. A classic four way intersection only utilises four segments.

IV. Testing and Debugging

Once programming of modifications for the game began, it became apparent that the code would need to be tested in order to determine that it is working correctly. Because modifications are being made to a full game, it is near impossible to test changes made without running all of the extra features of the game. The process used for implementing and running the created mod is outlined below:

1) Write and compile the code into a dll file using Visual Studio.
2) Move a folder containing the created dll file into the Addons folder for the game and also delete the old version of the dll that may already be there.
3) Start the game up through steam.
4) Enable the mod in the content manager found in the game's main menu.
5) Finally, load a save game file and observe the effects of the mod.

It is clear that this process is tedious, it has to be done every time a change is made in order for the change to be tested. This significantly slows development down, the loading of the game and save game is particularly slow and can take over five minutes for a slower computer. Because of this problem, it was decided that most of the programming would be done on a fast desktop computer that could load the game and save games in seconds. This maximised the efficiency of the development process.

Another issue that arose when testing was when the code had problems which resulted in errors. Depending on the error, the game would either crash or the parts of the code in the mod would simply not get executed. It was obvious something was wrong when the game crashed but much less so when parts of the mod would not execute while the game still ran normally. If parts of the code did not get executed due to an error, the error would be written into an error/output log. However, these error messages did not contain line numbers making it hard to locate the source of the error. What had to be done was to fill the code with print statements to the output log so that when an error occurred, the line that caused the error could be pinpointed by checking what the last print line to be printed was.

V. Proposed Solution

The proposed solution is a mod for Cities: Skylines that allows the user to run different traffic light algorithms that either they have written themselves or one of the three traffic light algorithms that are already implemented in the mod, two of which are ITS traffic light algorithms. The code is written in such a way that it is easy to write new traffic light algorithms and implement them due to the flexible framework. Using this mod, qualitative data can also be recorded and output to a spreadsheet to compare different traffic light algorithms' impact on traffic flow.

A. Architecture

As shown in Fig. 2 on the following page there are three main components in the proposed solution: The base game code for Cities: Skylines, the code for the Traffic Manager: President Edition mod and the code added on top of the mod to implement custom traffic light algorithms. The way that this added code works is as follows. The user can select an algorithm in the user interface provided by the AlgorithmPanel class. The setup for the algorithm will then be executed from the APIGet class which will generate all possible phases for the traffic light intersections as well as converting the lights at these intersections from Base Game TrafficLight classes into FlexibleTrafficLight classes. The phases are stored as a

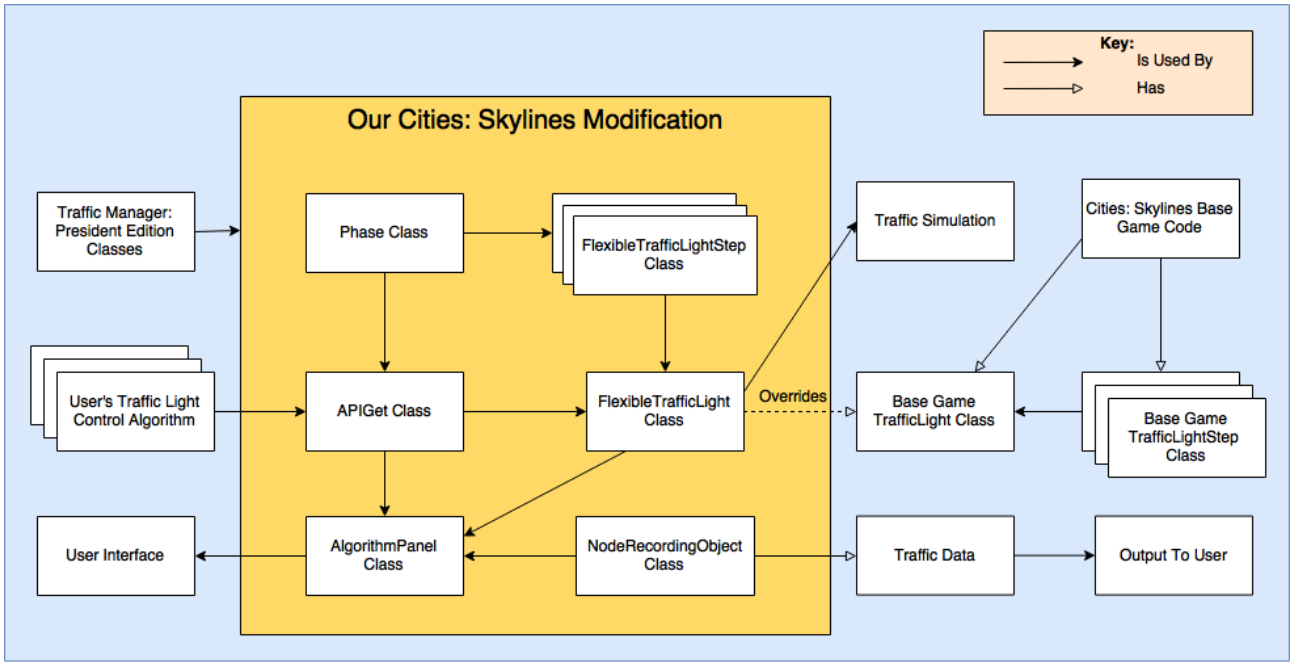

Fig. 2. Architecture of the implementation in Cities: Skylines

list of Phase classes. These phases are then converted into FlexibleTrafficLightStep classes so that they can be interpreted by the FlexibleTrafficLights. Once the setup is done, the selected algorithm will begin and can use the APIGet class to retrieve real time traffic data to make decisions.

B. Framework for Creating Algorithms

The way that the mod has been written allows for users to go into the mod's code and write their own traffic light algorithms. They are provided with a framework which gives them access to important information when writing their algorithm. It provides them with the list of all possible phases for each intersection as well as methods for retrieving real time data about the number of vehicles at ends of an intersection as well as their wait times. All the user has to do is write a function which determines the index of the phase in the list of phases that the traffic lights should be running for each individual intersection. The process for generating all possible phases is displayed in Algorithm 1.

Algorithm 1: Automatic Algorithm Application

```
tlsMan = traffic light simulation manager instance;
intersectionList = list of intersections in the current
  simulation;
algorithm = function to be evaluated by intersection
  every second;

foreach Intersection i in intersectionList do
    if i.hasLights then
        if i.numSegments <= 4 then
            trafficSim =
              tlsMan.addNodeToSimulation(i);
            trafficSim.setupFlexibleTrafficLight(i,
              algorithm);
            phaseList =
              calculateNonConflictingPhases(i);
            foreach Phase p in phaseList do
                trafficSim.FlexibleLight.addStep(p);
            end
            trafficSim.FlexibleLight.start();
        end
    end
end
```

C. AWAITS

Auckland Wait Alleviation Intelligent Transport System is the name of the first traffic light algorithm implemented in the mod. This is an ITS traffic light algorithm based on the computer systems approach to traffic management as covered in the ITS section of related work. In summary this algorithm attempts to maximise the number of vehicles serviced in each phase using knowledge of queue lengths at the intersection. The phase is chosen greedily by constructing a phase by looping through and selecting movements to make up the phase in descending order of queue length. The pseudo code can be found in Algorithm 2.

D. AWAITS++

Auckland Wait Alleviation Intelligent Transport System++ is the name of the seconds traffic light algorithm implemented in the mod. This is also an ITS traffic light algorithm based on the computer systems approach to traffic management and builds off of AWAITS. In summary this algorithm potentially services the highest number vehicles across all movements. AWAITS++ differs from AWAITS in that it doesn't choose a phase greedily instead it calculates the number of vehicles that a phase can service for each phase and then chooses the phase that

Algorithm 2: AWAITS

```
phaseList = all possible phases at intersection;
movements = all possible movements at intersection;
bestPhase = phase currently running at intersection;
if 3 seconds have passed since last phase change
 then
    foreach movement m in Movements do
        m.queueLength = number of vehicles in m;
        if movement contains a vehicle that exceeds
         threshold wait time then
            m.queueLength = ushort.MAX
        end
    end
    tempPhaseList = phaseList;
    sort movements in descending order by
     movement.queueLength;
    for i=0; i <movements.length; i++ do
        tempPhaseList = phases in tempPhaseList
         that service movements[i];
        if tempPhaseList is not empty then
            phaseList = tempPhaseList;
        else
            tempPhaseList = phaseList;
        end
        if phaseList.length == 1 then
            bestPhase = phaseList[0];
            return bestPhase;
        end
    end
else
    return bestPhase;
end
```

Algorithm 3: AWAITS++

```
phaseList = all possible phases at intersection;
movements = all possible movements at intersection;
bestPhase = phase currently running at intersection;
if 3 seconds have passed since last phase change
 then
    bestPhase = phaseList[0];
    foreach phase p in PhaseList do
        p.vehiclesServiced = number of vehicles that
         p will service;
        p.starvedMovements = number of movements
         in phase that contain vehicles exceeding
         threshold wait time;
        if bestPhase.starvedMovements
         <p.starvedMovements then
            bestPhase = p;
        end
        if bestPhase.starvedMovements ==
         p.starvedMovements then
            if bestPhase.vehiclesServiced
             <p.vehiclesServiced then
                bestPhase = p;
            end
        end
    end
    return bestPhase;
else
    return bestPhase;
end
```

Algorithm 4: Round-Robin

```
phaseList = all possible non redundant phases at
 intersection;
currentPhase = phase currently running at
 intersection;
if timer is greater than some set time then
    currentPhaseIndex++;
    if currentPhaseIndex == phaseList.Length then
        currentPhaseIndex=0;
    end
    nextPhase = phaseList[currentPhaseIndex];
    timer = 0;
    return nextPhase;
else
    timer++;
    return currentPhase;
end
```

services the most. It also takes into account starvation so that movements with cars waiting for longer than some pre-determined threshold time will be prioritised. The pseudo code can be found in Algorithm 3.

E. Round-Robin

Round-Robin is the name of the third traffic light algorithm implemented in the mod. This is not an ITS traffic light algorithm. It is meant as a baseline algorithm to compare the ITS algorithms with. In summary this algorithm runs every possible phase at an intersection excluding phases in which the movements are a subset of the movements of some other phase. Each of the phases runs for the same amount of time and the order of the phases is cyclical. The pseudo code can be found in Algorithm 4.

F. Recording Data

The mod provides a button that can be pressed once to start recording data while displaying a timer and then pressed again to stop recording data and save it into a spreadsheet. The recording gathers the following variables for the entire city: total intersection throughput, average intersection wait time, average vehicle journey time and total vehicle journeys completed. The recording also gathers the following variables for individual traffic light intersections: average wait time, intersection throughput and longest wait time.

VI. Testing Environment

After the development of the mod and the three algorithms, a testing environment to analyse and compare the algorithms quantitatively had to be developed. The mod cimtographer was used to import Auckland CBD's geographical data into the game. That data was then used to guide the placement of roads, intersections and buildings to achieve a representation of Auckland City

within Cities: Skylines and can be seen in Fig. 3.

Fig. 3. Auckland City Recreation

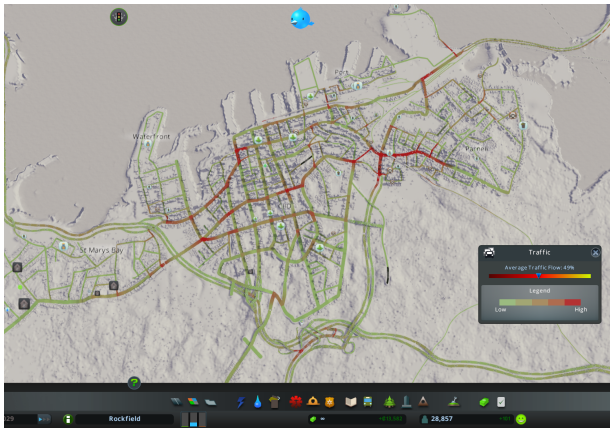

Fig. 4. Traffic density heat map view of the city

Each algorithm was run on the Auckland City simulation for 15 minutes each. Because Cities: Skylines is a full city simulation game it includes elements like population growth, day and night cycles, electricity and many other features which impact traffic density. Therefore after every 15 minute test the city had to be reset to the state it was in before the test since some of the elements may have changed during the recording that could affect the next recording. The reset was done by simply exiting the game state without saving and then reloading the save state from before the recording started. During the actual testing of the algorithm a traffic density heat map can be viewed in real time to obtain a broader and more qualitative understanding of the traffic light algorithm's effects, this can be seen in Fig. 4.

VII. Results

The three implemented algorithms: AWAITS, AWAITS++ and Round-Robin have been compared quantitatively using the results of the tests outlined in the previous section. This section graphs and analyses these results.

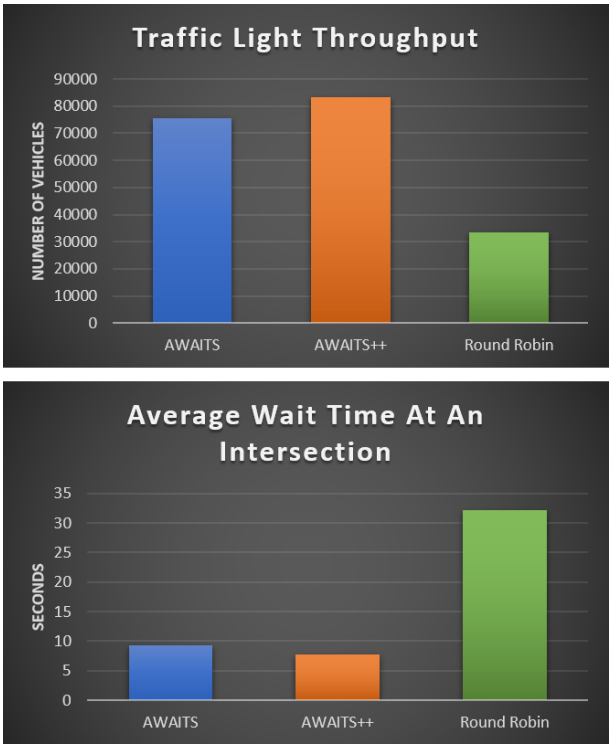

Fig. 5. Intersection throughput and average wait time

A. Intersection Analysis

1) City Wide Intersection Results: As seen in Fig. 5 Traffic Light Throughput has been graphed, this is the total number of times a car passed through an intersection within the city. The graph shows that AWAITS and AWAITS++ have significantly higher throughput than Round-Robin as would be expected since Round-Robin is not an adaptive algorithm and only has a throughput of 33536 vehicles. AWAITS++ has a throughput of 83334 vehicles and AWAITS has a throughput of 75591 vehicles so AWAITS++ is better than AWAITS by a 7743 vehicle margin. Fig. 5 also contains a graph of Average Wait Time which is the time a vehicle will spend waiting at at an intersection on average. Again, the graph indicates that AWAITS and AWAITS++ are both significantly better than Round-Robin which has an average wait time of 32.2 seconds. AWAITS++ has an average wait time of 7.7 seconds and AWAITS has an average wait time of 9.3 seconds so AWAITS++ is better than AWAITS by a 1.6 second margin.

2) Individual Intersection Results: Fig. 6 shows a comparison of the three traffic light algorithms based on the the number of intersections at each wait time. AWAITS++ is the most consistent algorithm with all intersections having an average wait time in the range of 1-38 seconds. Both AWAITS and Round-Robin had much greater ranges, 1-115 seconds and 3-120 seconds respectively. Even though AWAITS and Round-Robin have similar ranges, AWAITS was far more consistent than Round-Robin. 72% of intersections under AWAITS were within the average wait time range of 1-10 seconds while only 22% of intersections under Round-Robin were in this range. AWAITS++ had 82% of its intersections in the 1-10 second wait time range. Another interesting statistic drawn from these results is

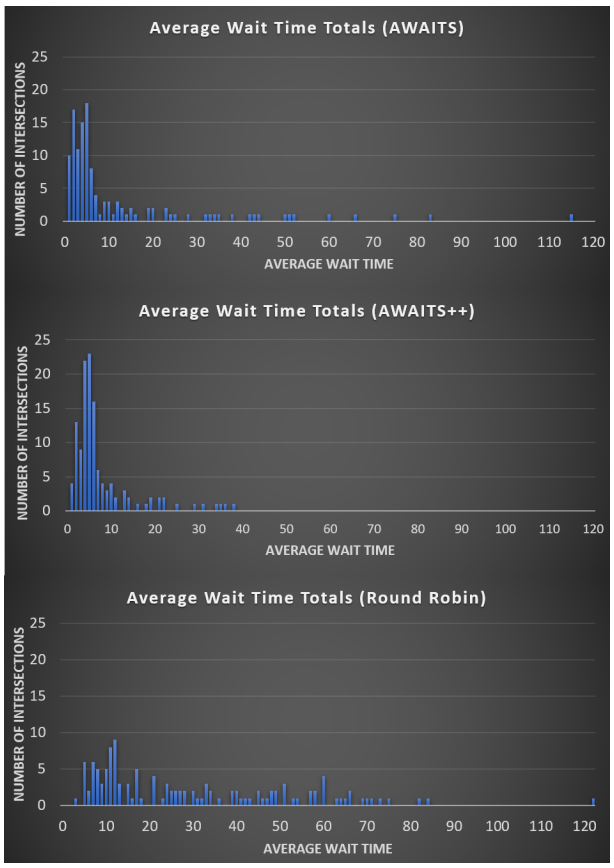

Fig. 6. Average wait time for each intersection

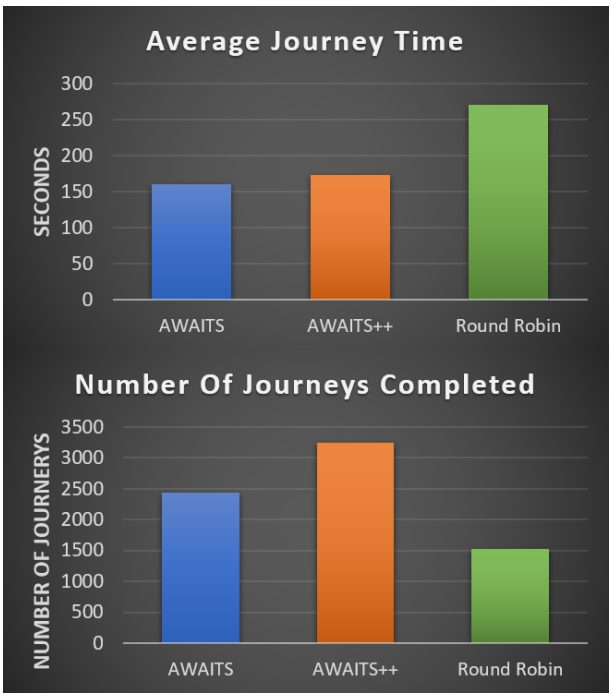

Fig. 7. Average vehicle journey time and number of journeys completed

that the intersection with the highest average wait time in AWAITS++ had a better average wait time than 9% of intersections running AWAITS and 33% of intersections running Round-Robin.

B. Journey Analysis

In addition to measuring data for intersections, data for vehicle journeys was recorded and graphed. From Fig. 7 it is apparent that AWAITS actually has a lower average journey time than AWAITS++. The average journey time for AWAITS is 159.7 seconds and for AWAITS++ it is 173.5 seconds. This could be considered unexpected since all of the previous data indicates that AWAITS++ is better than AWAITS. However, this could be explained by the second graph in Fig. 7. AWAITS++ completed 3249 journeys, 809 more journeys than AWAITS, many of these extra journeys may have been naturally longer journeys that may not have even had enough time to complete under AWAITS in the 15 minute recording time. These long journeys would therefore not have been recorded and would not contribute to the average journey time in AWAITS. Round-Robin was considerably worse than the other two algorithms for both average journey time (270.1 seconds) and total journeys completed (1521).

VIII. Future Work

There is plenty of potential to expand the functionality of the proposed solution.

- The mod enables easy implementation of new algorithms that can be applied city wide automatically. Thus, in future more traffic light algorithms could be developed and added to the mod to be compared with the existing traffic light algorithms already implemented in the proposed solution.

- The next most obvious step for this mod would be to create an implementation of SCATS, the traffic light system that Auckland uses. This was not implemented in the proposed solution as it is not an algorithm that can be applied automatically, it requires user input. SCATS requires intersections to have marriages with each other so that they can synchronise. The marriages between intersections must be manually determined. Therefore, to implement SCATS in the future a user interface for all of the manual functionality would need to be implemented.

- With regards to calibration of Cities: Skylines, traffic information and intersection layout for a few intersections would need to be recorded and then exported from Cities: Skylines to AIMSUN, a more universally accepted traffic simulation tool. This is so an implementation of AWAITS in AIMSUN can be compared with the implementation of AWAITS in Cities: Skylines using the same intersections and traffic data to determine Cities: Skylines' effectiveness as an accurate traffic simulation tool.

- Finally, Cities: Skylines has other potential uses outside of traffic light simulation. One such example that could be implemented in future is road user charging. In Cities: Skylines the vehicles store many different variables and can be modified to store even

more custom variables. Some of the variables they currently store are: location, vehicle type and noise pollution. All of these variables could potentially be used to inform on road user charges.

IX. Conclusions

A. Implemented Mod Evaluation

After analysing the results of the algorithm tests it was concluded that AWAITS++ was the best traffic algorithm implemented in the mod and Round-Robin was the worst. Overall, the mod implemented in the proposed solution opens many doors for further use of Cities: Skylines for the simulation of ITS traffic algorithms as well as many other ITS concepts such as electronic road user charges and self driving vehicle AI. This is due to the ability the mod provides to manipulate the in game objects.

B. Cities: Skylines Evaluation

Overall, as a city simulation tool, Cities: Skylines is concluded to be effective for the visualisation and presentation of ideas such as the ITS traffic light algorithms that were implemented in the proposed solution as well as only costing $36. Other traffic simulation tools have annual costs up to and over $1000. It was determined that because no calibration of Cities: Skylines was done, it is impossible to use the results of the traffic light algorithms tested in the game to prove that the algorithms will be effective in the real world. However, it still provides a very good starting point as to whether an algorithm is viable for further testing. What Cities: Skylines excels at is its ability to quickly run mods and provide a high quality visualisation without slowdown in a city wide micro simulation which can be viewed in real time and traversed flexibly. This is a capability that many existing traffic simulation tools such struggle to achieve.

X. Acknowledgements

I would like to thank Partha Roop for his guidance and support throughout the project and Nasser Giacaman for his insightful feedback. I would also like to thank the PRETzel team at Auckland University. In particular I would like to thank Mahmood Hikmet for providing his research into ITS traffic light algorithms using a computer systems approach as well as Hammond Pearce for his development of Auckland in Cities: Skylines.

References

[1] Ian Wallis. *The costs of congestion reappraised / Ian Wallis and David Lupton*. Wellington, N.Z. : NZ Transport Agency, Wellington, N.Z., 2013. ID: dedupmrg677961616; Includes bibliographical references (p. 50-52).

[2] P. R. Lowrie. Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic. 1990.

[3] Alexander Skabardonis, Steven Shladover, Wei bin Zhang, Liping Zhang, Jing-Quan Li, Kun Zhou, Juan Argote, Matthew Barth, Kanok Boriboonsomsin, and Haitao Xia. Advanced traffic signal control algorithms. *Advanced Traffic Signal Control Algorithms*, 2013.

[4] Intelligent transport systems (its), Apr 1 2017.

[5] Moody Hikmet. *From worst case gini coefficient to preemptive multi-tasking: Consideration for fairness and efficiency in intelligent transportation systems*. PhD thesis, 2017.

[6] Rajeshwari Sundar, Santhoshs Hebbar, and Varaprasad Golla. Implementing intelligent traffic control system for congestion control, ambulance clearance, and stolen vehicle detection. *IEEE Sensors Journal*, 15(2):1109–1113, 2015.

[7] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2015.

[8] M. Khanjary. Using game theory to optimize traffic light of an intersection. In *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 249–253, 2013.

[9] G. Kotusevski and K. A. Hawick. A review of traffic simulation software. 2009.

[10] Katyanna Quach. Train your self-driving car ai in grand theft auto v what could possibly go wrong?, 2017.

[11] William Emfinger, Abhishek Dubey, Peter Volgyesi, Janos Sallai, and Gabor Karsai. Demo abstract: Riapsa resilient information architecture platform for edge computing. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 119–120. IEEE, 2016.

[12] Victor-Philipp Negoescu. Cities-skylines-traffic-manager-president-edition, 2015.

[13] Telerik. Justdecompile .net assembly decompiler and browser, 2017.