Department of Electrical & Computer Engineering
Part IV Research Projects
Final Report

A Design Environment for Intelligent Transportation Systems Using Cities: Skylines

Prepared by Jake H. Thomas

Project 25
Project Partner: Jarrod Van den Heuvel
Project Supervisor: Partha Roop
Second Supervisor: Nasser Giacaman

Declaration of Originality

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

Name: Jake H. Thomas
18th September 2017

A Design Environment for Intelligent Transportation Systems Using Cities: Skylines

Jake H. Thomas
Department of Electrical and Computer Engineering, University of Auckland, New Zealand

***Abstract*—This report details the implementation of a design environment for intelligent transportation systems using the video game Cities: Skylines. Current traffic simulation tools generally have poorly rendered graphical interfaces that display the simulation. They are also generally costly and are only able to tackle small scale road networks unless visual fidelity is sacrificed. Cities: Skylines is a city simulation game with existing traffic simulation features. The game allows programmers to create and distribute modifications to improve or add features to the game. We have created a modification that implements some adaptive traffic control systems and allows users to plug their own traffic light algorithms into the game's simulation. This was done by building on top of an existing Cities: Skylines traffic modification. Users are able to simulate a city's traffic system using different traffic light algorithms and then extract relevant data. A model of Auckland's inner city was also created for use within the game. This report describes the steps taken to adapt Cities: Skylines into a traffic simulation tool and analyses results from a test undertaken on implemented traffic control algorithms. Overall, we establish that, given careful modification, Cities: Skylines can be viewed as a promising tool in the development of intelligent transportation systems.**

I. Introduction

As the population increases there are more and more cars on the road. This is leading to increased congestion and longer travel times. One way to alleviate this is to implement an Intelligent Transportation System (ITS). Traffic engineers and scientists have been researching ways to control traffic in an adaptive and intelligent way for many years. Currently New Zealand uses an adaptive traffic control system (ATCS) called the Sydney Coordinated Adaptive Traffic System (SCATS). However, there are many other different ATCSs, so how do we decide which one is better? Is there a program that we can use that will both help in the decision making process and be able to display different traffic control systems in action? Can we do this in an aesthetically pleasing and simple way?

As transport technology improves, traffic light control algorithms can become more complex, yielding better traffic throughput. Due to the rise of the Internet of Things, traffic data is becoming ubiquitous. For example, some newer car models have the ability to communicate with each other with vehicular ad hoc networks in anticipation of the arrival of self-driving vehicles. [1] Hence it is not a stretch of the imagination that ATCSs could communicate with vehicles to obtain data more accurate than the currently used induction loops. With the concept of more knowledgeable traffic control systems in mind, a computer systems approach to traffic is possible.

With novel algorithms being produced, it is necessary to test them. There are many traffic simulation tools in the market the help with this, but none have a strong visual component that facilitates in demonstrating traffic control algorithms to the wider public. [2] Traffic simulation tools generally focus on small sets of intersections, making it difficult for the user to get an understanding on how the algorithm would perform on a city-wide scale. When these tools are able to run city-wide simulations, it is at the expense of the visual component of the application.

Conversely, contemporary video games have strong visual components, and game engines are very powerful. Because of this, researchers are interested in the application of games as informative tools and how they can be useful in a scholarly setting.

This project aimed to leverage the functionality of the city building and management game Cities: Skylines to create a design and review environment for ITSs. Cities: Skylines is a game which allows its community to modify the game code, which means that the public has created new features that are freely available. By utilising the game's original features, using existing modifications and creating a new modification, it is possible to treat the game as a software tool.

After analysing possible ways to use Cities: Skylines as an ITS design environment, we implemented several features. Two novel traffic light algorithms were implemented using the game and users are free to write their own algorithms, thanks to a fairly modular code base. Users can also record data as the simulation runs to compare algorithms within the game.

The rest of the report is organised as follows. Section II details a literature review of the project topic and related work, whereas Section III explores the proposed solution and its architecture. Section IV presents the results from the testing of the implemented traffic control algorithms, Section V discusses future work that could be undertaken as a result of this project, and Section VI concludes the report.

II. Literature Review

A. Project Background

In densely populated urban areas congestion is becoming increasingly problematic. One way to tackle this is to improve the area's traffic control systems. There are currently multiple competing ATCSs being used across the globe, and there is much research in the area of ITS.

An important part in creating a traffic control system is how the traffic is modelled. A more complex model can lead to a control system with higher predictive accuracy. For example, the application of game theory to vehicular traffic leads to a model of a multi-agent system. [3] [4] Agents (vehicles) compete and collaborate with each other to achieve their goals.

Using this model, traffic engineers have been able to derive traffic management optimisation algorithms. [5]

It is also possible to take a computer systems approach to traffic. Posing the traffic problem as a computing problem changes the model. Vehicles waiting at intersections can be viewed as tasks that require processing. It is not possible to process every task simultaneously, so some sort of task ordering is required. Each task therefore has some priority, which would relate to the vehicle's waiting time. In computer systems it is important to not starve tasks and enforce some kind of fairness. This is also applicable to traffic modelling as individual vehicles should not be completely neglected in favour of servicing a greater number of vehicles. Therefore, each task can be considered to have a deadline. However, these tasks are not critical; it is not absolutely necessary that their deadlines be met. Hence, traffic can be modelled as a mixed-criticality system.

Using this model, a computer systems oriented ATCS has been designed. This ATCS is proactive in the actions it takes, which is different to the reactive nature of SCATS. This means that the traffic lights function in a way that services the current state of traffic, rather than previous traffic states. The paper in which this computer systems oriented ATCS is proposed acts as a precursor to this project. [6] Modelling the computer systems oriented ATCS was therefore a key motivation in this project. We implemented a version of this ATCS in Cities: Skylines and have named it the Auckland Wait Alleviation Intelligent Traffic System (AWAITS). Its performance in the game is discussed further in this report.

There is of course software already available that can sufficiently simulate traffic models and how they interact with ATCSs. However, they have mostly been designed with traffic engineers in mind. It is difficult to engage the casual observer with the traffic simulation and display to them results in an interesting manner. Therefore, this project was interested in using the video game Cities: Skylines as a modelling tool that can show the benefits of a computer systems oriented ATCS, like AWAITS, in an engaging way to the wider public. It has already been shown that the traffic simulation of Cities: Skylines can be used in an academic setting. [7] This project was also interested in the data collection aspect of traffic simulation and in providing accessible code for users to write their own traffic control algorithms.

B. The Problem Space

There are advantages and disadvantages to using a game as a modelling tool. The primary disadvantage is that generally a game is not designed to be used as a modelling tool, so alterations to the game code are necessary. On the other hand, the power that the game engine gives the user is a great advantage. Game engines are designed so that computations can be done without sacrificing the graphical output to the user. This is usually done by utilising a computer's GPU and CPU. Traffic simulation tools tend to be data oriented. They focus on processing traffic data, and this data is presented to the user in a simplistic and cold way.

The following subsections explore some of the disadvantages of traffic simulation tools and the advantages of using games as modelling tools, with specific reference to Cities: Skylines itself.

C. Traffic Simulation Tools

There are many different traffic simulation tools in the market, each with its own benefits. SUMO is an open-source traffic simulation tool which allows users to extend the program to suit their own needs. [8] However SUMO has a very minimalistic GUI which isn't very aesthetically pleasing. VISSIM is another traffic simulation tool that provides a realistic representation of traffic flow. [9] VISSIM provides a three-dimensional view of the traffic simulation. Other traffic simulation tools, such as PARAMICS and AIMSUN, also provide a similar view of their simulations. Comparisons between various traffic simulation tools have been made in the past, making conclusions such as "PARAMICS has the most comprehensive visual displays," and that "AIMSUN satisfies a large number of ITS criteria." [10]

It can be difficult then to decide which traffic simulation tool will best suit a user's goals. It is also worth noting that traffic simulation models can be classified as either microscopic or macroscopic. Microscopic models view vehicles individually, while macroscopic models aggregate the description of traffic flow. This project was interested in a traffic simulation that supports a microscopic model with a strong focus on the visual display to the user. Although many contemporary traffic simulation tools are microscopic, none have a strong enough visual element for our purposes. Therefore we had to look elsewhere to create such a tool.

D. Using Games as Educational Tools

For years, video games have been solely a source of entertainment. However, more recently games have been used as educational tools. Video games are powerful tools with respect to graphics rendering, physics simulation, and human-computer interaction. Because of these features, the adaptation of video games for serious applications has been researched with great interest recently. One important research topic of late is the use of game engines to create three-dimensional virtual environments for various purposes. [11] [12] These virtual environments can be realistic and interactive.

Furthermore, games can be used to support and educate about urban planning and urban systems. [13] Cities: Skylines is one such game that can be used as an educational tool that introduces students to urban systems. [14] [15] This is thanks to the game's in-depth city simulation features and support for modifications. It is therefore possible to take this idea of using a game as an educational tool and extend it to the case of using Cities: Skylines as a traffic simulation tool.

E. The Potential of Cities: Skylines as a Modelling Tool

The base Cities: Skylines game can already be seen as a microscopic traffic simulation. Vehicles in the game have their own individual destinations and make decisions in order to get to those destinations. Individual vehicles follow each other at safe distances and obey traffic rules. The game also uses

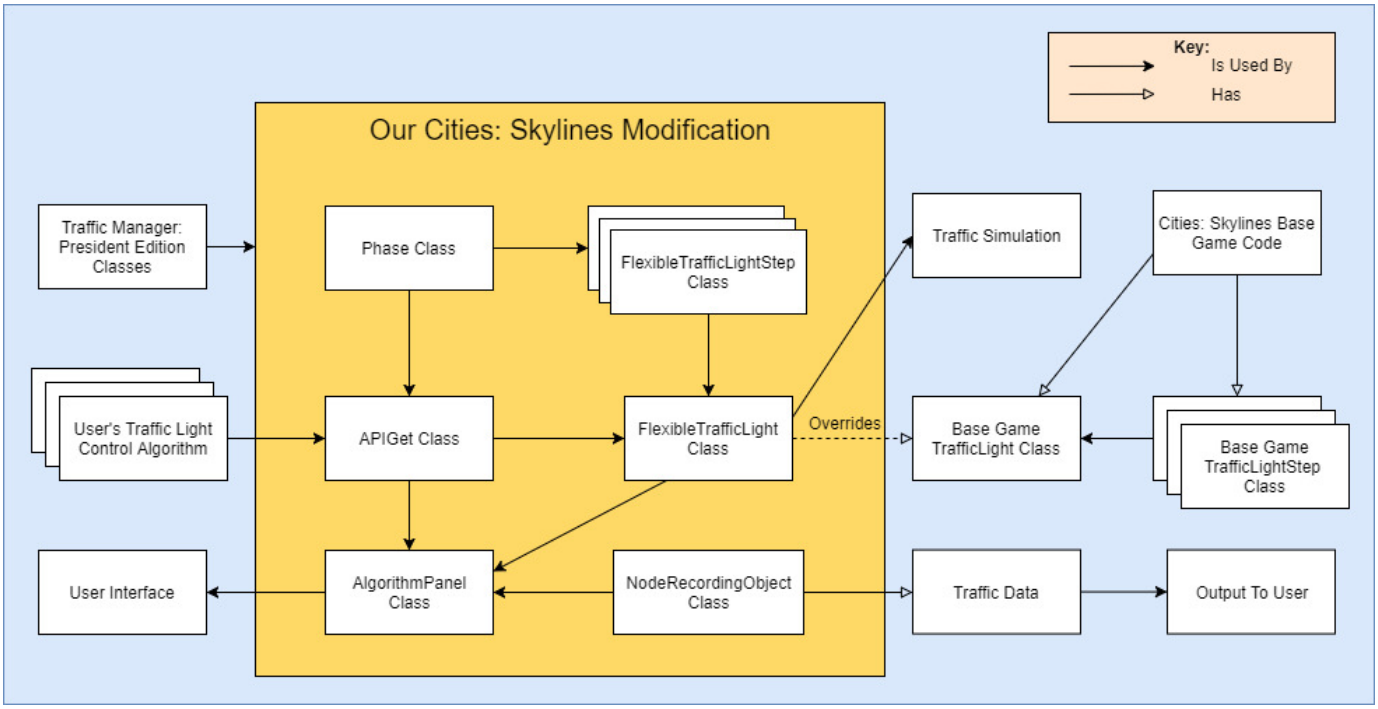

Fig. 1. An architectural diagram of the implemented Cities: Skylines modification.

its own traffic light algorithm that services longer queues in real time. Therefore, with the assistance of modifications, data such as the speed and wait times of individual vehicles can be gathered.

There are already existing modifications, such as Traffic Manager: President Edition [16] and Traffic++ [17], that expand on the traffic simulation capabilities of the game. Using these modifications a user can manually control traffic lights, set timers on traffic lights, and fine tune intersections, among other things. This shows that the traffic simulation of the game is flexible and that programmers can easily add additional features using their own modifications.

As a game, Cities: Skylines has a rich and engaging user interface. Cities are modelled and displayed in an artistic, yet realistic, way. It is also possible to get a wide range of data concerning the city. For example, players can use a real time heat map of traffic congestion which provides a macroscopic view of traffic. Cities can be designed from scratch, or cities that have been made by the game's community can be used. These cities can be of varying scale, and populations can reach the hundreds of thousands. Also, Cities: Skylines is relatively cheap, currently costing $35.99 NZD on Steam. This is much more affordable than some other traffic simulation tools, which can cost thousands of dollars.

This project aimed to harness the potential of Cities: Skylines by creating a modification for the game that users can use to create and test their own traffic control algorithms. Users are able to apply an algorithm across all of the traffic lights in any given Cities: Skylines map. They are then able to record data that can be used to showcase the algorithm's strengths and weaknesses, or compare with another algorithm.

F. Project Differences

There currently isn't any other project that is attempting to use Cities: Skylines as a traffic simulation tool. The graphics of Cities: Skylines are far better than that of any current traffic simulation tool, and it would be very difficult to improve an existing tool's graphics to a comparable degree. The scale of cities in Cities: Skylines is much bigger than what can be developed with 3D visualisation in current traffic simulation tools. Cities: Skylines also has the added benefit of already having many cities created by the community available for free public use.

This project was established as unique, as it was concerned with the development of a simple to use traffic simulation tool from an existing game with a strong focus on visual display.

III. Proposed Solution

Below is the solution that this project proposed for the creation of a design environment for ITSs using Cities: Skylines. This section explores the architecture of the implemented modification, the features that were implemented, and the developmental process.

A. Modification Architecture

Figure 1 shows a simplified diagram which represents the architecture of our Cities: Skylines modification. Because there were already existing modifications for Cities: Skylines that extended the traffic simulation features, we decided it would be easiest to transform the game into a simulation tool by building upon one of these already existing modifications. The most comprehensive one that we found is called Traffic Manager: President Edition (TMPE). Its source code is available on

GitHub and has an MIT licence, so it can be used and modified if the resulting code also has the same license. [16] As part of our modification we modified several existing classes in TMPE, and we also created some new classes. These new classes are mentioned in Figure 1, and their purposes are explained below:

- AlgorithmPanel Class: The class for the panel shown in the user interface. From the panel, users can access the algorithms we have implemented, along with the data recording functionality.
- APIGet Class: Contains functions that get different values and objects for the traffic control algorithms. This includes nodes, segments, the next phase, turn possibilities, etc. Users may use these functions in their own algorithms.
- FlexibleTrafficLight Class: Represents a set of traffic lights at an intersection that is using an applied algorithm. Every second the FlexibleTrafficLight's SimulationStep function is called. Depending on which algorithm the lights are running, this function will make a call to another function which makes the decision about which phase should be active for the next second.
- FlexibleTrafficLightStep Class: Represents a phase at a traffic light. A FlexibleTrafficLight has a list of FlexibleTrafficLightStep objects and when an algorithm decides that there needs to be a phase change at a node, a new FlexibleTrafficLightStep is activated.
- NodeRecordingObject Class: This class collects data about the traffic simulation and individual intersections when the data recording functionality is activated.
- Phase Class: Also represents a phase at a traffic light. Used in the construction of valid phases with non-conflicting movements. Once a valid phase is constructed it is converted to a FlexibleTrafficLightStep.

The architecture that has been developed follows the conventions of the TMPE modification and allows our modification to be flexible in regards to the application of traffic light algorithms. When a user wants to use the modification they simply activate it in the main menu. Then when they load a map they can navigate to our menu by clicking on the TMPE crown symbol. From the menu the user can activate algorithms we have implemented or an algorithm they have designed. They can also start a data recording session.

We decided that the algorithms that were implemented should be able to be applied to any map that the user wanted. This meant that the algorithms had to be applied to each set of traffic lights in the map in an automatic fashion. Algorithm 1 shows pseudo-code that represents how this automatic algorithm application works.

Algorithm 1: Automatic Algorithm Application

```
tlsMan = traffic light simulation manager instance;
nodeList = list of nodes in the current map;
algorithm = function to be evaluated by node every
 second;
foreach Node n in nodeList do
    if n.hasLights then
        if n.numSegments <= 4 then
            trafficSim = tlsMan.addNodeToSimulation(n);
            trafficSim.setupFlexibleTrafficLight(n,
             algorithm);
            phaseList = calculateNonConflictingPhases(n);
            foreach Phase p in phaseList do
                trafficSim.FlexibleLight.addStep(p);
            end
            trafficSim.FlexibleLight.start();
        end
    end
end
```

B. Implemented Features

Before the features that we implemented are discussed, it is important to talk about the existing traffic features of Cities: Skylines and the TMPE modification. When a user plays the game they can start building a city from scratch or load a city that's already been completed to some degree. Users can share their cities via Steam and others can use their saved games to avoid the initial city building work. This is great for the purposes of a simulation tool as users do not necessarily need to construct a city to test their algorithms as there is a large library of cities available for use.

The vanilla Cities: Skylines game comes with a variety of traffic related features that were utilised in the creation of our modification. As stated in Section II, individual vehicles in the game have their own destinations and routes. This means that each vehicle is its own object in the game's code. This allows data regarding a vehicle, such as its travel time and intersection wait times, to be stored in the vehicle's object. Not only does the code allow access to the location of a vehicle through the vehicle's object, but it allows access to the vehicles on a stretch of road through the road segment's object. This makes it trivial to get the wait times of vehicles at a particular intersection.

The game also comes with traffic visualisation tools. Individual vehicles, or collections of vehicles, can have their current routes to their destinations visualised, as shown in Figure 2.

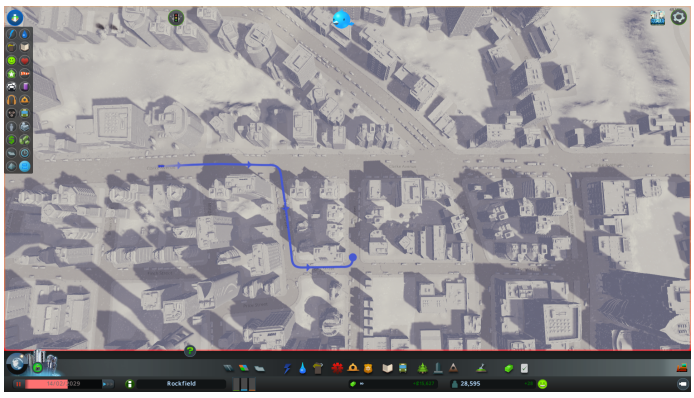

Fig. 2. An individual vehicle's route displayed in blue by the base game.

The game also has a view mode that overlays a heat map

onto the environment that is updated based on the current traffic flow in the city. This can be seen in Figure 3, with red road segments indicating congestion and green road segments indicating free flowing traffic. This provides a macroscopic view of the state of the traffic. These visualisations are good for users testing traffic control algorithms, as they can get views of which intersections are performing poorly under a particular algorithm, and which routes are being affected.

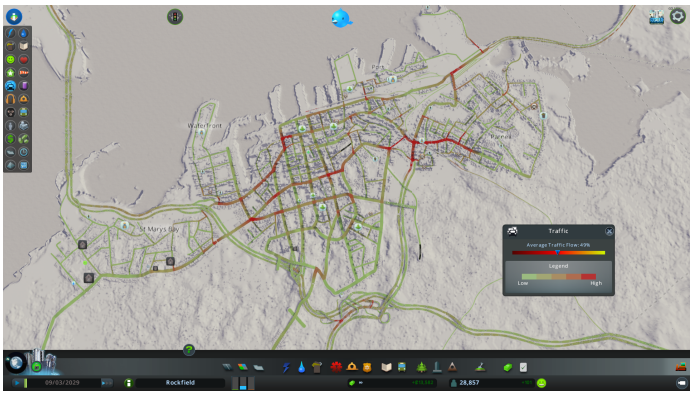

Fig. 3. A heat map that provides a macroscopic view of the city's current traffic flow.

The TMPE modification provides extra features that allow the user to perform more fine-grained tuning on individual intersections. Users can manually control the traffic lights at an intersection, transform the lights at an intersection to run a time-based algorithm, and apply extra traffic rules to intersections. Because this modification interacts with the base game's traffic code with great precision, we decided to follow the structure of several classes when making our own.

Below are the features that we implemented in order to allow users to treat Cities: Skylines as a traffic simulation tool:

- Implemented three traffic light algorithms that users can apply to any map in the game.
- Fairly modular code so that a user can write their own algorithms after learning how the code works. This is discussed in the modification architecture subsection above.
- Users can record data as the simulation runs to compare algorithms within the game.
- A version of Aucklands inner city was created to provide users with a more familiar environment to run their tests in.

In implementing these features it was not our goal to create a full traffic simulation tool, but to create a simple prototype that acts as a proof of concept.

The three traffic control algorithms we implemented are named Round Robin, AWAITS and AWAITS++. Before these algorithms are described in more detail, it is important to note some of the traffic jargon used. A phase is a series of non-conflicting movements, with a movement being a single direction that traffic can take from a segment. A segment is a section of road that enters into an intersection. Finally, the term node is what we use to refer to an individual intersection in our code.

Round Robin is perhaps the most basic traffic control algorithm, and it is not adaptive at all. It services every phase for a fixed amount of time in some cyclic sequence. We implemented Round Robin to serve as a baseline algorithm. Any properly designed ATCS should perform much better than Round Robin, otherwise we can assume something has gone wrong. Round Robin does guarantee that every car will eventually be serviced, but this may take a very long time. Pseudo-code that represents the Round Robin algorithm can be seen in Algorithm 2.

Algorithm 2: Round Robin

```
phaseList = all possible non-redundant phases at node;
currentPhase = phase currently active at node;
phaseLength = some fixed time in seconds;
if timer > phaseLength then
    currentPhaseIndex++;
    if currentPhaseIndex == phaseList.Length then
        currentPhaseIndex = 0;
    end
    nextPhase = phaseList[currentPhaseIndex];
    timer = 0;
    return nextPhase;
else
    timer++;
    return currentPhase;
end
```

AWAITS, the traffic algorithm that takes a computer systems approach to the problem of traffic discussed in the literature review section, is the second algorithm that we implemented. AWAITS assumes total knowledge of the traffic. In the real world this information would likely come from vehicle-to-vehicle communication, but we abstract away the hardware requirements in the game. The algorithm works by greedily servicing movements with the longest queue lengths. If individual cars reach a predefined wait-time deadline they immediately become the top priority. This ensures that vehicles aren't starved because they are sitting in a queue with a short length. The algorithm also uses a minimum phase time so that in the event of equally sized queues, vehicles in the active phase have enough time to move through the intersection. We used 3 seconds in our version of AWAITS, but experiments could be conducted to find a more optimal time. Algorithm 3 shows pseudo-code that represents the AWAITS algorithm.

AWAITS++ is an algorithm that we developed after implementing AWAITS. We decided that the greedy aspect of AWAITS was perhaps not the best approach to servicing vehicles, so we developed AWAITS++ to make a comparison. In AWAITS++, instead of servicing movements greedily, the algorithm chooses the phase that will potentially service the most amount of cars across all its movements. AWAITS++ still prioritises cars who have waited longer than some soft deadline. The pseudo-code of AWAITS++ in Algorithm 4 shows some of the differences between AWAITS++ and AWAITS.

Users simply toggle a particular algorithm by clicking its button in the menu. There is also a button for a user created

Algorithm 3: AWAITS

```
phaseList = list of non-conflicting phases at node;
movementsList = list of possible movements at node;
bestPhase = phase currently active at node;
waitTimeThreshold = number of seconds that must pass
 for waiting vehicle to be given top priority;
if 3 seconds have passed since last phase change then
    foreach Movement m in movementsList do
        m.queueLength = number of vehicles waiting to
         make movement m;
        if queue contains a vehicle with a waitTime that
         exceeds waitTimeThreshold then
            m.queueLength = ushort.MAX;
        end
    end
    tempPhaseList = phaseList;
    sort movementsList by descending
     movement.queueLength;
    for i=0; i < movementsList.length; i++ do
        tempPhaseList = phases in tempPhaseList that
         service movementsList[i];
        if tempPhaseList is not empty then
            phaseList = tempPhaseList;
        else
            tempPhaseList = phaseList;
        end
        if phaseList.length >= 1 then
            bestPhase = phaseList[0];
        end
        return bestPhase;
    end
else
    return bestPhase;
end
```

Algorithm 4: AWAITS++

```
phaseList = list of non-conflicting phases at node;
bestPhase = phase currently active at node;
waitTimeThreshold = number of seconds that must pass
 for waiting vehicle to be given top priority;
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 (sum of queue lengths);
        p.starvedMovements = number of movements in
         p that contain vehicles with a waitTime
         exceeding waitTimeThreshold;
        if bestPhase.starvedMovements <
         p.starvedMovements then
            bestPhase = p;
        end
        if bestPhase.starvedMovements ==
         p.starvedMovements then
            if p.vehiclesServiced >
             bestPhase.vehiclesServiced then
                bestPhase = p;
            end
        end
    end
    return bestPhase;
else
    return bestPhase;
end
```

ATCS, but if a user wants to design more than one algorithm they will have to insert extra buttons to toggle them. All of the implemented algorithms will work in cities where cars drive on either side of the road, which is important in maintaining consistency. Once an algorithm is activated, a user can start recording data about traffic flow. As the game runs in real time, that is the speed at which data is collected. In other words, if a user wants to collect data for an algorithm running for half an hour, they will have to start recording, wait half an hour, then stop recording. When a recording session finishes, the data is output to a .CSV file. The file records traffic statistics for every intersection with traffic lights in the map, as well as overall statistics for the city. This includes data such as average intersection wait time, number of vehicles processed, and average journey time. We ran a test on all three of our implemented algorithms to have a quick glance at their performances. The results of this test are discussed in Section IV.

Finally, a model version of Auckland's inner city was created for this project. The terrain of the map is quite accurate and the roads are in the correct locations. Because of the limitations of the game's road creation, the road widths and lane counts are often inaccurate. A screenshot of the map is shown in Figure 4. This model Auckland was created by first importing the terrain data into the game. Then an image of Auckland's road network was overlaid onto the map. Using this overlay, roads were then drawn and buildings placed. A previously created Sky Tower asset was also placed at an appropriate position. The benefits of using this map are two-fold. Users are likely to be more impressed with the game if the map they use reflects a real life environment they are familiar with and traffic data that's collected has more meaning if it doesn't come from a map based in pure fantasy.

C. Developmental Process

After becoming familiar with the game's interface and gameplay, we had to gain an understanding of how the game's source code worked. Although the game supports modification, and there is a community of players who share modifications they have made, there is not much documentation explaining how the code works. There's also no repository for the source code either, so the only way to access it is by decompiling the game. This removes any comments that may have been in the source code, which decreases its readability. Functions and variables do have logical names, so this did make things easier.

Fig. 4. The Cities: Skylines map that is a model of Auckland.

Deciding which modifications to use to help us was fairly straight forward. The traffic modifications with more features are generally incompatible with each other, so we had to decide on one to use. We eventually settled on the TMPE modification, as it already provided the user with some of the features we were interested in. The source code for TMPE is freely available, and while it is commented, it's not commented very well. Hence, much time was spent working out how the functions and variables in the code connected with each other.

Another task that took up a significant chunk of time was testing. Because of the nature of the game, in order to make any change to the code, the game needs to be stopped, the code rebuilt, and the game started up again. There is also no real way to test the validity of traffic control algorithms using unit tests. You could use them if you knew where specific cars were going to be at certain times and how exactly the algorithm should service them, but as the game simulates the locations of cars based on a whole list of factors, this is simply not possible. Instead, we had to use a combination of including debug statements and testing by observation. This way we could focus on intersections ourselves, and step through the processes. The time wasted every time we had to add a new debug statement and run a test again highlighted a problem with using Cities: Skylines; if a user isn't comfortable with the code, and includes a bug in their algorithm, they may have a tough time tracing and fixing that bug.

Another problem with Cities: Skylines as a simulation tool that presented itself was the simple fact that it was designed as a game. Cities: Skylines receives updates from the developers through patches every so often, and these are automatically installed if you boot up the game with internet access. Sometimes these patches make modifications incompatible, and it is up to the authors of the modifications to resolve this. At one point during our modification's development, a traffic-related update was released that made a lot of our code not work anymore, and we really had no choice but to wait for the TMPE authors to update their code, so we could do the same. For future work I would recommend acquiring a copy of the game from the developers that didn't run into this issue.

Because working on this project could become tedious due to the reasons expressed above, we did much of our work via peer programming. This made testing much faster and was most beneficial when one problem was preventing either me or my project partner from making progress alone. We did occasionally work on separate parts of the code on our own, but we would always come back together to make sure our work integrated properly.

IV. Results

To test whether the modification we created could generate useful data we conducted an experiment involving the three algorithms that were implemented. We ran each algorithm within the Auckland map for 15 minutes and collected data for that time frame. This period was at the same time of day in the game's simulation for each algorithm, as traffic demand changes during the game's day/night cycle. The Auckland map has 125 intersections with traffic lights, and our data recording functionality generates data for all of them. Presented below are the results from this simple test.

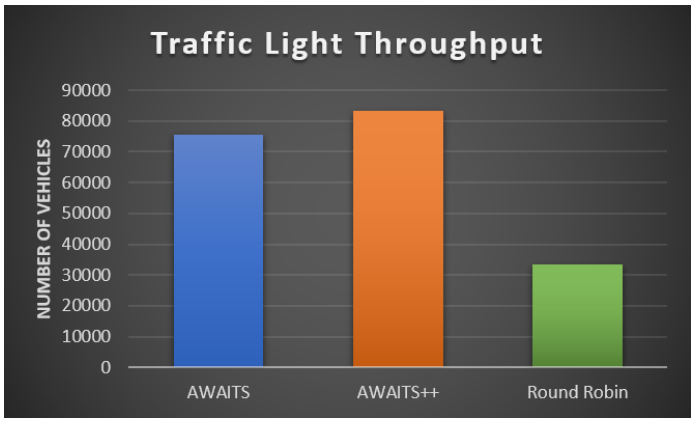

Fig. 5. Total vehicle throughput across the map's intersections for each algorithm.

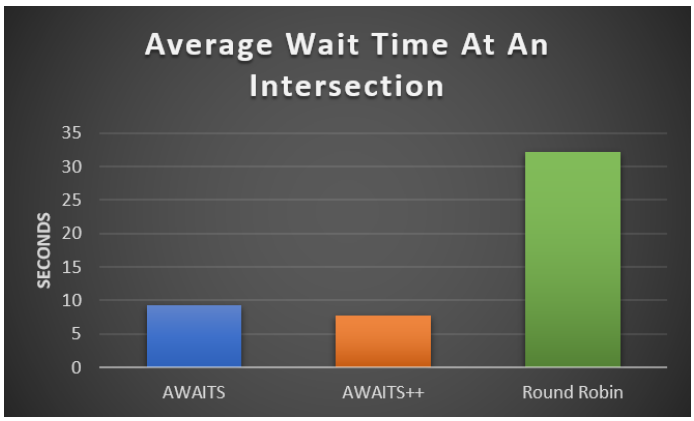

Fig. 6. Average wait time of vehicles at intersections for each algorithm.

Figures 5-8 show data for each algorithm's performance when looking at the city overall. Figure 5 shows that AWAITS++ had the highest traffic light vehicle throughput at 83000 vehicles over the 15 minutes that the data was recorded. Conversely, Round Robin had the worst vehicle throughput, with 32000 vehicles. Figure 6 shows that AWAITS++ had

the lowest average intersection wait time at approximately 7 seconds, which is 2 seconds lower than the average intersection wait time of AWAITS. Round Robin had the worst average intersection wait time of 33 seconds. From just these two graphs it is clear that Round Robin performs significantly worse than the two ATCSs.

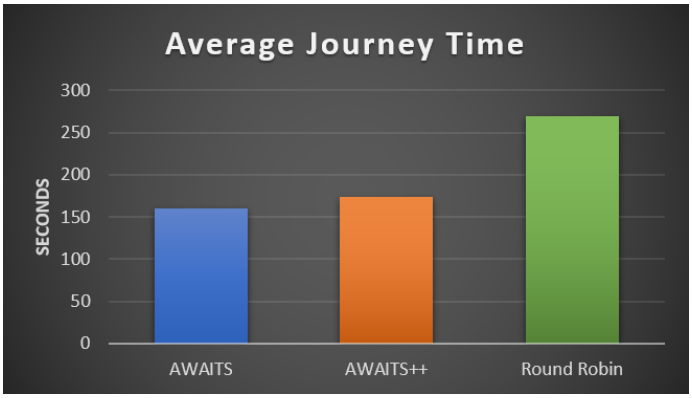

Fig. 7. Average journey time of journeys completed by vehicles for each algorithm.

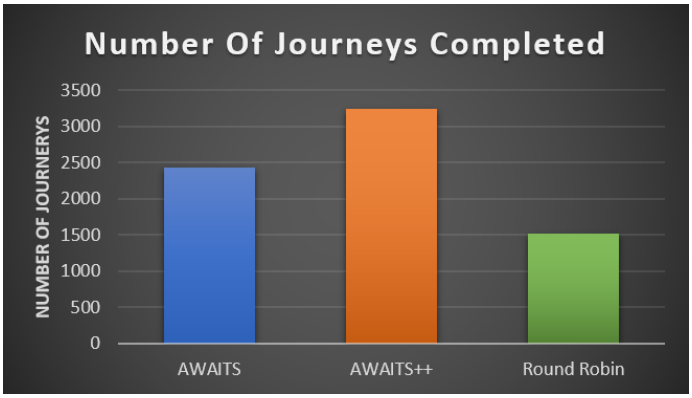

Fig. 8. Total number of journeys completed by vehicles for each algorithm.

Figure 7 shows that AWAITS had the lowest average journey time of 160 seconds, 13 seconds lower than AWAITS++. This is the only graph that shows AWAITS performing better than AWAITS++, but this can perhaps be explained by Figure 8. AWAITS++ had the highest number of journeys completed in 15 minutes, 3250 journeys compared with 2450 journeys under AWAITS. Our data recording system currently only records journeys that start and finish within the recording time frame. Since AWAITS++ completed a higher number of journeys, it stands to reason that longer journeys were completed under AWAITS++ that do not get completed under AWAITS. Therefore, it makes sense that AWAITS++ would have a slightly higher average journey time than AWAITS, as longer journeys are being completed. Round Robin once again performed poorly, with an average journey time of 270 seconds and only 1500 completed journeys.

Figure 9 shows the performance of the algorithms across the intersections in the map, namely the counts of particular average wait times. A right-skewed distribution, as shown

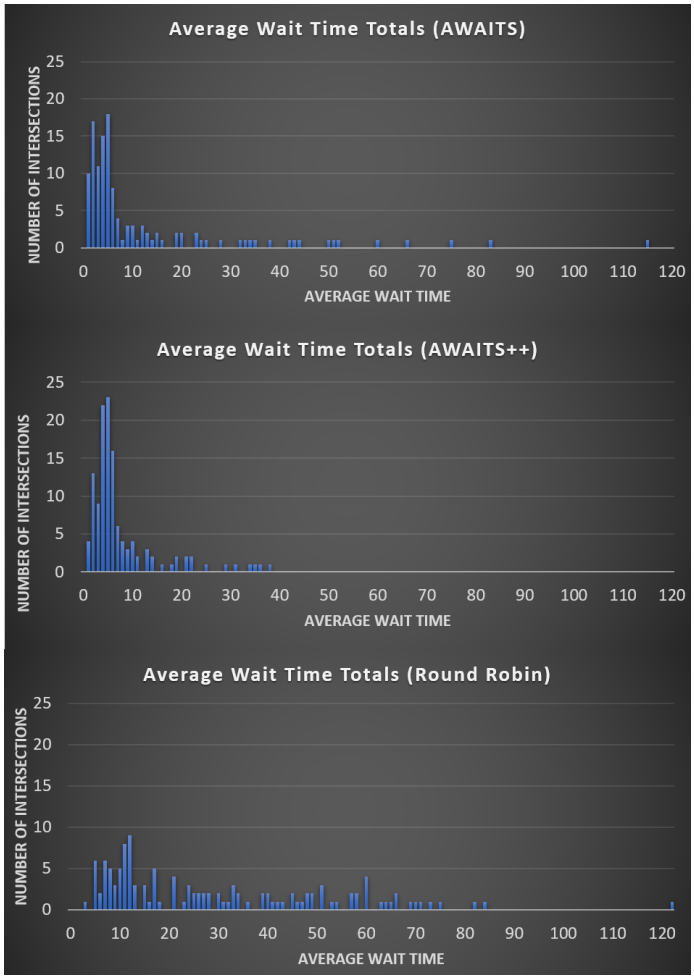

Fig. 9. Spread of average vehicle wait times at individual intersections for each algorithm.

in the graphs for AWAITS and AWAITS++, indicates good performance as most of the intersections in the map have low average wait times. The graph for the average wait time totals under Round Robin is much less skewed and many intersections had poor average wait times.

AWAITS++ and AWAITS had 83% and 72% of intersections with average wait times between 1-10 seconds, respectively. Under Round Robin only 22% of intersections had an average wait time between 1-10 seconds. The worst performing intersection in AWAITS++ has a better average wait time than 9% of intersections running AWAITS and 33% running Round Robin. In fact, the actual intersection that performed the worst was the same across all the algorithms. Under AWAITS++ it had an average wait time of 38 seconds, under AWAITS its average wait time was 115 seconds, and under Round Robin its average wait time was 120 seconds.

These results should by no means be considered to be valid. They could be, but there isn't enough evidence to assert that. There were many things that we could have done to improve the validity of the results. We could have performed the test ten times and taken the average results to avoid outliers. We could have also performed the test over different recording time frames. However, it was not our goal to produce valid

results, but merely demonstrate that the modification that we created could produce data that could then be graphed and used to compare algorithms within the game. This test showed that AWAITS++ performed the best overall, but more testing may prove that there are particular disadvantages to using it that are not present when using AWAITS.

These results were however consistent with how we initially thought the algorithms would perform. Although the data cannot be used to prove how an algorithm would perform in the real world, it can certainly indicate its qualities. Therefore, we propose that these results indicate that a combination of the game's visualisation of traffic in real-time and the raw data that can be collected and processed is a good way of conveying a traffic control algorithm's performance.

V. Future Work

The purpose of this project was to investigate the potential of Cities: Skylines and create a modification that served as a prototype for a traffic simulation tool. It is our belief that we have been successful to that end, but we also think that, while our work has been important and fruitful, there is work that could be done in the future that builds upon what we have done.

One feature that we initially intended to implement was the creation of a version of SCATS. Unfortunately, we realised that the traffic control algorithm would not work particularly well with our system of automatically applying the algorithm to intersections. This is because SCATS has the concept of intersection marriages. Two or more intersections can be married, which means that their timings are synchronised in a way that enables better throughput in a particular direction. These marriages are purposeful, and so when running SCATS a user would have to manually assign these marriages. We felt that we didn't have enough time to implement this kind of functionality, but we also think that an implementation of SCATS would be an interesting feature that would provide insight into the accuracy of the game's simulation.

The user interface for our modification is not perfect, and is currently meant for users who understand the modification. This could be improved in the future so that the menu is more usable. After this is done, a usability study and heuristic evaluation could be performed to evaluate how useful the modification really is. This way the application's benefits could be more formalised and there would be evidence that it does serve a need.

Further tests to confirm the validity of the algorithms we have implemented could also be conducted. Results from experiments could be compared with similar results from other traffic simulation tools. This could also assist in the calibration of the game to better reflect reality. Also, because traffic control is rather complex on a city-wide scale, there could still be bugs in the code we have written. The application of the algorithms could also be improved, as currently we can only automatically generate a list of phases for an intersection with four segments or less.

Finally, once the modification has been improved sufficiently, it could be used to evaluate other transport-related issues. One such issue involves the application of electronic road user charges. There are many different pricing models that could be used to charge road users, but currently no real way to test their effectiveness. The city simulation of Cities: Skylines could be of benefit here, so we recommend looking into this area in future.

VI. Conclusions

This project was concerned with using Cities: Skylines as a design environment for ITSs. It has been determined that Cities: Skylines can offer features that other mainstream traffic simulation tools cannot. Cities: Skylines has a strong focus on the visualisation aspect of traffic simulation and it can offer a simulation of a city-scale environment. We have created a modification for the game that is easy to use and offers several traffic simulation features.

Three traffic control algorithms were evaluated using our modification, and while the results may not be reliable, they show that the modification can produce data that is consistent with previous reasoning. The results summarise the simulation that they were obtained from in a way that differs from the effect of simply viewing the simulation. The visual and statistical aspects of the developed modification are both important and add to the user's experience.

Our Cities: Skylines solution to current problems with traffic simulation tools is inexpensive, visually appealing and easy to use. This project was the first step in providing software capable of city-scale simulations that are displayed in an aesthetically pleasing manner. Overall, this project has shown that games can be used as informative tools when modified appropriately and that we should be looking to novel technologies to develop and evaluate intelligent transportation systems.

Acknowledgements

I would like to thank Dr. Partha Roop, Mahmood Hikmet and Hammond Pearce for their continued help and support. This project would not have been possible without their efforts.

References

[1] E. C. Eze, S. Zhang, and E. Liu, "Vehicular ad hoc networks (VANETs): Current state, challenges, potentials and way forward," in *2014 20th International Conference on Automation and Computing*, 2014, pp. 176–181.

[2] P. Hidas, "A functional evaluation of the AIMSUN, PARAMICS and VISSIM microsimulation models," *Road and Transport Research*, vol. 14, no. 4, pp. 45–59, 2005.

[3] R. Mandiau, A. Champion, J.-M. Auberlet, S. Espi, and C. Kolski, "Behaviour based on decision matrices for a coordination between agents in a urban traffic simulation," *Applied Intelligence*, vol. 28, no. 2, pp. 121–138, 2008.

[4] L. Cortes-Berrueco, C. Gershenson, and C. R. Stephens, "Traffic games: Modeling freeway traffic with game theory," *PloS one*, vol. 11, no. 11, 2016.

[5] C. Wang, S. Wang, H. Xu, and H. Chen, "A P2P traffic management model based on an ISP game," *Journal of Computers*, vol. 9, no. 6, p. 1478, 2014.

[6] M. Hikmet, "From worst case gini coefficient to preemptive multi-tasking: Consideration for fairness and efficiency in intelligent transportation systems," Ph.D. dissertation, The University of Auckland, 2017.

[7] S. Eisele, I. Mardari, A. Dubey, and G. Karsai, "RIAPS: Resilient information architecture platform for decentralized smart systems," in *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, 2017, pp. 125–132.
[8] D. Krajzewicz, G. Hertkorn, C. Rssel, and P. Wagner, "SUMO (Simulation of Urban MObility)-an open-source traffic simulation," in *Proceedings of the 4th middle East Symposium on Simulation and Modelling*, 2002, pp. 183–187.
[9] M. Fellendorf, "Vissim: A microscopic simulation tool to evaluate actuated signal control including bus priority," in *64th Institute of Transportation Engineers Annual Meeting*. Springer, 1994, pp. 1–9.
[10] S. A. Boxill and L. Yu, "An evaluation of traffic simulation models for supporting ITS," *Houston, TX: Development Centre for Transportation Training and Research, Texas Southern University*, 2000.
[11] R. Andreoli, R. D. Chiara, U. Erra, and V. Scarano, "Interactive 3d environments by using videogame engines," in *Proceedings of the International Conference on Information Visualisation*, vol. 2005, 2005, pp. 515–520.
[12] E. L. Aziz, Y. Chang, S. K. Esche, and C. Chassapis, "Virtual mechanical assembly training based on a 3d game engine," *Computer-Aided Design and Applications*, vol. 12, no. 2, pp. 119–134, 2015.
[13] R. Stephens, *Urban Planning Games and Simulations: From Board Games to Artificial Environments*, ser. Simulation and Gaming in the Network Society. Singapore: Springer Singapore, 2016, pp. 253–273.
[14] P. Haahtela, "Gamification of education: Cities Skylines as an educational tool for real estate and land use planning studies," p. 13, 2015. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-201506253211
[15] B. Bereitschaft, "Gods of the city? Reflecting on city building games as an early introduction to urban systems," *Journal of Geography*, vol. 115, no. 2, pp. 51–60, 2016.
[16] V.-P. Negoescu, "Cities: Skylines - Traffic Manager: President Edition." [Online]. Available: https://github.com/VictorPhilipp/Cities-Skylines-Traffic-Manager-President-Edition
[17] joaofarias (GitHub User ID), "Cities: Skylines - Traffic++." [Online]. Available: https://github.com/Katalyst6/CSL.TransitAddonMod