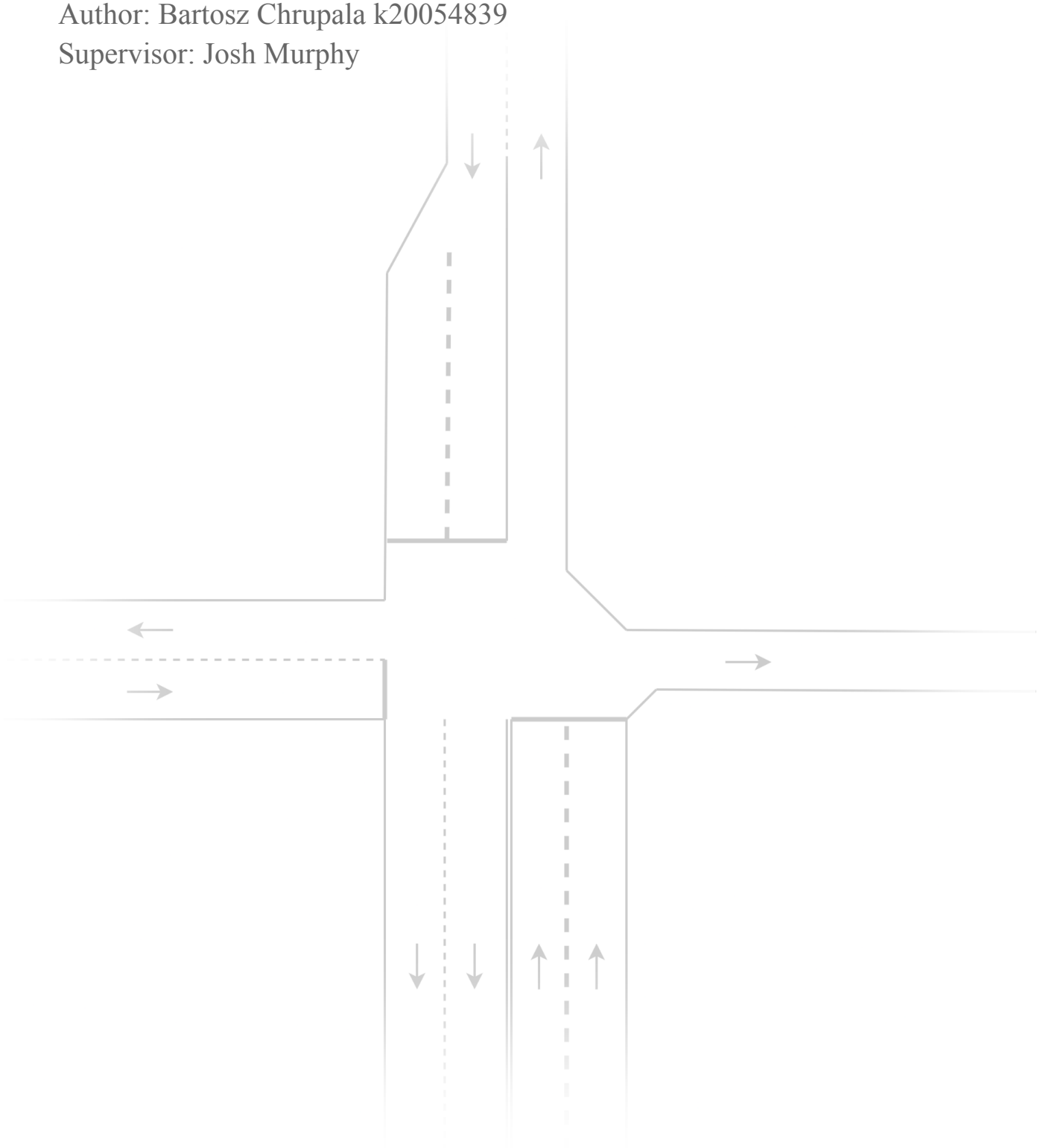
Traffic Infrastructure Flow Simulation

Year 3 Computer Science BSc Individual Project, 2023

Author: Bartosz Chrupala k20054839
Supervisor: Josh Murphy



I verify that I am the sole author of this report, except where explicitly stated to the contrary, Bartosz Chrupala 28/10/2022.

Abstract

Urban planners face the complex problem of designing and testing effective road intersection designs. The goal of this project is to create a multi agent simulation to model urban junctions. This micro-simulation will look at how the lanes on roads and traffic lights schemes can be configured to maximise traffic throughput, and minimise waiting times. Users of the model should be able to replicate real life intersections and make modifications to test for an improvement or decrease to traffic flow. The issue of congestion is one which grows everyday, with increasing population, and the ever more pressing threat of climate change. Therefore it is important that our road layouts are as optimised as possible.

Many industry standard implementations of traffic microsimulations have unintuitive interfaces, which result in a steep learning curve to their use. One of the secondary objectives of this project is to improve on the nuanced UI of other solutions, by incorporating some concepts from games.

Contents

Appendix Contents

Introduction

Congestion on our road systems is a problem that has effects much greater than simply the loss of time due to longer journeys. Vehicles idling in traffic jams and stoplights result in unnecessary emission of greenhouse gases and polluting compounds, contributing to climate change and pollution of our atmosphere. Pollution alone results in a tragic loss of life, for example it is estimated that in 2015 between 5.71 million and 7.29 million deaths can be attributed to pollution.(GBD 2015 Risk Factors Collaborators)[3] Transport is a major contributor to this. Transport networks are the backbone to every successful economy, as logistics is an important step in most industries, where inefficiencies are likely to cause knock-on effects on steps down the line. For time sensitive deliveries, if a deadline is missed or a shipment is late, there is a significant impact on the supply chain. Goods taking longer to be transported means drivers have to be paid for more time, resulting in a greater logistical overhead. Perishable goods such as refrigerated food are at higher risk of spoiling while in transport; the longer this takes, the more food waste is produced. Londoners spend an average of 2 days and 2 hours per year in traffic (HPI, 2018)[1], at the scale of a country this is an enormous loss in productivity with national repercussions. In addition to that, the time spent in traffic is not only wasted, but stressful and miserable.

The same survey by HPI found the majority (72%) of time in traffic was caused by intersections, where traffic paths must cross. Another cause is accidents on the road that result in a bottleneck. Therefore even a slight increase in the efficiency and safety of traffic intersection and interchange designs can have a significant difference in the world. A simulation which would enable users to test new and compare existing intersection layouts is an essential part in determining which design is most applicable and efficient to be built. For existing intersections the tool could be used to identify problems and help test potential changes to improve traffic flow.

Before a road network can be simulated, first it must be designed and inputted into the software. The first stage of this will be to select the positions and lengths of road segments on the grid, in addition to properties of the road such as the number of lanes in each direction. After this the user will use a tool to combine road segments into intersections, and edit its properties like which lanes are used for turning in each direction. Once an intersection is built, the user could add traffic lights to it and set up the order in which they will be illuminated.

As well as modelling the state of the network and agents internally, the simulation should display this information using a graphical user interface in a comprehensive yet aesthetically pleasing way. When creating roads the user should be able to visualise how the road network will be structured. When roads are connected into an intersection the arrangement of the road segments must be computed to work out the best way to align themselves into a connected intersection without overlapping.

The difficulty of building a traffic simulation comes from the span of abstraction levels that must be considered to consistently mimic the movement of vehicles in a complex system. This kind of simulation will have to model everything from the distance cars move between each frame, to the structure of the road network with intersections being the nodes that connect it together. Another aspect which will be is the movement of the traffic, the design should attempt to avoid collisions between vehicles in a way which mirrors human behaviour.

While this project could aim to model a single intersection, where the user can vary the number of roads and lanes a part of it and the configuration of the traffic lights, I believe that this is too limited for the scope of modern cities. Each small environment (such as an intersection) interacts with adjacent ones to create an emergent system, which acts differently than a standalone intersection. Microsimulations only consider the specific details of a system such as the timing of traffic lights or connection of lanes, where higher level simulations only track the movement of groups of vehicles though arterial roads. However the minute specifics of a design can have a huge impact on vehicle movement. For example: if the traffic light phase of a main road is just too short to get all the cars through the intersection, with each passing phase, the number of cars on the main road builds up, eventually causing a complete standstill due to grid-lock. Because of this I have chosen to build a tool which considers both the larger scale and the specific configuration of the road system, giving the user flexibility to design a network with the exact configuration of the real world.

Enabling this flexibility significantly increases the complexity and size of the codebase. This is because each component, such as road segment, lane or intersection must be self contained. A simpler application could have a tool for adding a lane to a road in a single intersection, which would add it to the array storing lanes. However with a free designer, where the user can create multiple components, the relationship between them must be managed, and calculations need to be done at runtime to work out the best arrangement of them as the network evolves.

Interchanges

An interchange (grade-separated junction), is a type of junction which aims to reduce the crossing of paths for vehicles. This is achieved with overpasses and tunnels, which avoid the problem of minimising interruption to flow by diverting the flow of traffic into the 3rd dimension. The main consideration in designing an interchange is the layout of the lanes of traffic and in what way they will lead drivers to their destination. For example the stack interchange gives every turning direction its own path resulting in almost no reduction to flow. However as this is a four level interchange, it requires an immense construction cost. The cloverleaf is another type of interchange which successfully addresses the issue of cost, as it only requires 1 bridge, for the straight on traffic. However this comes at the expense of some interruption to flow, as vehicles must weave between lanes quickly to reach their destination, possibly slowing down other traffic. Interchanges are very efficient and would ideally be used in every situation; however due to space, cost, and time constraints that is infeasible. Therefore they are usually only used on motorways between 2 high traffic flow routes. The constraints are especially true in urban environments where land is even more valuable, and important buildings are often in the way of a larger junction.

Intersections

Due to the practical drawbacks of interchanges, standard (at-grade) intersections are a much more common type of junction, where streams of traffic must be interrupted to safely cross each others' paths. Managing traffic across intersections in cities is one of the most complex problems we regularly interact with due to the number of factors involved. With 6,000+ traffic signals just in London[2], traffic lights are one of the most common ways of managing priority at an intersection. They are space and cost efficient, with only some interruption to traffic flow if managed efficiently. The scheme used to control when the phases of traffic lights change, the layout of the lanes, pedestrian crossings and bike lanes are all considerations designers must analyse when designing an efficient junction. Therefore the second major goal of the simulation is to investigate the efficiency of different designs of traffic light arrangements. This could prove useful if, for example, an urban designer is tasked with fixing grid lock in a neighbourhood. The software could demonstrate how much the installation of a sensor in the form of an induction loop in the road surface will increase the efficiency of an intersection by reducing wasted time in the phases.

Background

Traffic simulations are an important tool for urban planners as they allow a design to be tested and evaluated in the early stages, but without the cost of a real life model or actual implementation in the world. A term commonly seen in the semantic field of traffic modelling is microsimulation. This is a fine grained type of simulation where individual units such as vehicles and people are modelled in real time. This is opposed to higher level, large models which consider the overall efficiency and flow values of corridors of traffic. “Due to this modelling of individual vehicles, microsimulation is able to reproduce dynamic phenomena such as queuing behaviour and blocking back through junctions” (TfL Modelling Specification, 2021 p. 43)[2]. This type of simulation is what I plan to produce as it is most applicable to the design of singular intersections to study their throughput.

Industry Leading Examples

A notable example of a widespread tool used for traffic simulation is CORSIM (CORridor SIMulation). This software can be used to model vehicles on roads, intersections, motorways and the networks that arise from their combination. This package is well established because it can be used to replicate nuanced real life conditions through the diverse amount of aspects considered. This includes: driver reaction time, interruptions to flow such as incidents or rail crossings, and many more. It was first developed in the 70s for the US government’s Federal Highway Administration and is still in use 50 years later. The TSIS-CORSIM package is divided into FRESIM, which focuses on traffic on motorways. And NETSIM, which simulates urban traffic on intersections. Considering most traffic in London is at grade and with traffic signals, this is the aspect the model will focus on. This tool is extremely powerful and can simulate most realistic conditions, but it's notable for its steep learning curve. An average user can’t intuitively create a model they imagine due to the user interface being from decades past.

Another prominent example of a traffic model is PTV VISSIM. Similarly to CORSIM, it is a microsimulation working at the second by second interval level, which allows the simulation of a multitude of agents involved in the road environment. This includes cars, trucks, buses, pedestrians, and even rail. VISSIM was first developed by Planung Transport Verkehr AG in 1992 in Germany[7]. The two software packages are similar but have some distinct differences in their design. This was investigated by L. Bloomberg and J. Dale in 2000[8] in their paper comparing the models in a congested network. One of the differences uncovered was how the road layout is designed in the simulations. VISSIM uses a process in which the network is

built overlaid on a map of the area, and the user modifies the path followed by the road with the use of this background image without the use of nodes. In contrast, CORSIM uses an arguably simpler process through its use of nodes and links. The user adds nodes and connects them with edges to create roads. Considering my focus is on designing new road intersections and networks, the link node approach used by CORSIM is more applicable as it doesn't involve a preexisting map or network.

Another difference found in the paper[8] is the way drivers follow the vehicle ahead of them. Modelling human behaviour can be a complex task so different approaches can be applied. The naïve approach would be for the agents to maintain a constant distance between the agent in front and themselves. However, since every real driver has a different risk tolerance, perception, and driving style, this type of simulation would be too robotic. To solve this problem CORSIM requests the user to define ten 'driver types', which define the amount of headway they will maintain. These driver types are spawned in the simulation with corresponding frequencies. VISSIM takes a different approach, by using a psychological model which focuses on the perception of a driver. When a driver approaches a vehicle moving at a different speed, they try to accordingly accelerate or decelerate, however they cannot judge the difference in speed accurately. This results in over braking which is later compensated by accelerating back to the correct velocity. The driver type used by TSIS-CORSIM is more in line with the scope of this project and can be understood by the user more easily by users due to the way risk tolerance values are inputted for each type of agent.

Cities: Skylines

An element of inspiration for a traffic simulation comes from the 'Traffic Manager' plugin[5] for the city and road network building game 'Cities: Skylines'. This object of the game is to construct and manage a city with an efficient transport network. The plugin gives the user more control over how traffic behaves in the game, for example changing the yield signs at intersections, and managing traffic lights and intersection turn lanes. The lane connector tool is an intuitive way of assigning the directions vehicles can go from which lanes. The user simply connects lanes entering the intersection to the exits in the environment itself, rather than using UI which is often confusing and less efficient. This simple solution is illustrated in figure 1 below.

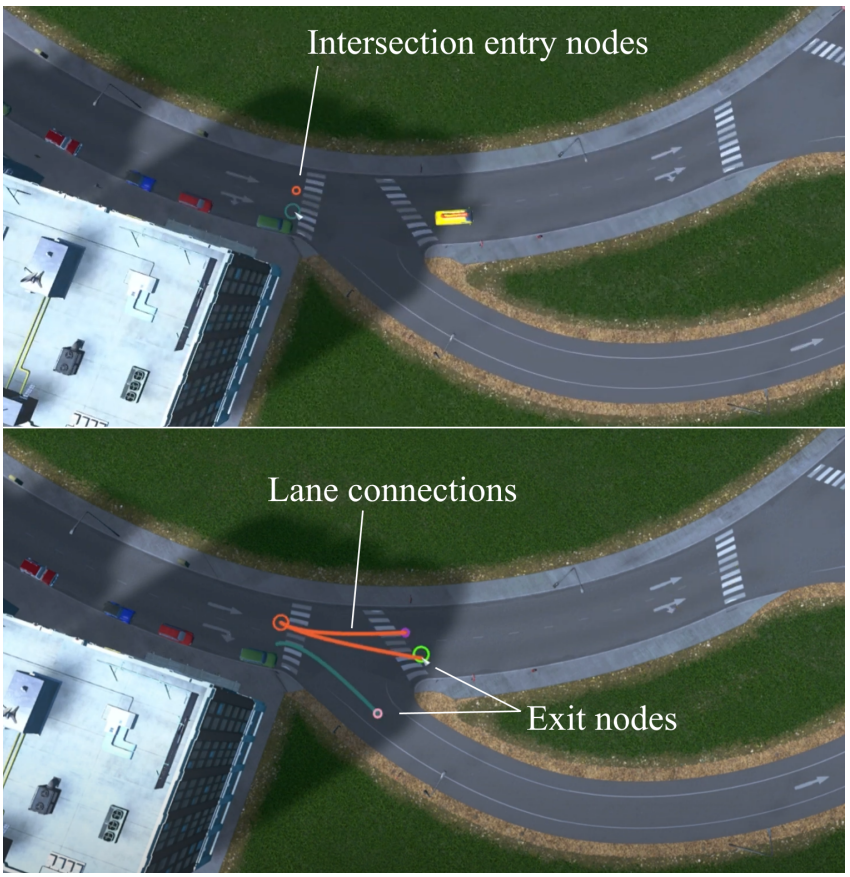


Figure 1: A user defining lane connectors using the traffic manager mod[5][6]

This traffic configuration has vehicles entering from the left in two lanes, then splits into two roads, one with two lanes, and another with only one. To define the traffic flow the user clicks on the two nodes on the left, then selects which of the three nodes on the right it should have an edge to. By default when an entry node does not have edges, vehicles can move from it to any other exit node. This arrangement means vehicles from the left and right lanes continue onto the left and right roads respectively.

Requirements

For a traffic micro-simulation to be suitable for its use case, many features are necessary. The table below explains the requirements, their importance for the simulation to work, and how this can be tested. When testing, a majority will refer to a ⅔ supermajority.

Table 1: Software requirements

#	**Priority**	**Requirement**	**Testing Questions**
1	Very High	A way for the user to input a road layout, including adding roads with different numbers of lanes.	Most users should be able to create a simple layout following the guide.
2	Very High	Vehicle spawn and despawn points. The simulation cannot be infinite and therefore must have an edge where vehicles come from.	Do vehicles spawn and despawn correctly?
3	Very High	A tool for designating which lanes connect with which other lanes in intersections, this will define what type of traffic lights appear at each entrance, for example one for straight on and left, and another for right turns.	Can users connect roads to other roads at intersections?
4	Very High	The user must be able to define the scheme of management for the traffic lights, this including timings between phase changes. The user will make adjustments to these schemes to evaluate which is the most efficient and maximises flow.	Do the correct traffic light colors appear in each phase designated by the user?
5	High	Agents which move through the road system and behave realistically, obeying the simulations traffic rules.	Vehicles must accelerate and break at a rate which can be achieved by real cars and lorries. Do vehicles go through red lights or leave the road border? Do vehicles collide or overlap?

6	High	The simulation must track fundamental statistics about the agents and environment to aid the analysis of the network design. For example: average vehicle speed, total time spent stopped at a traffic light, average agent travel time, vehicle count through an intersection, etc.	Are correct statistics about the running of the simulation returned after execution?
7	High	The user must be able to modify the parameters for the vehicles in the simulation, for example the number of cars spawning per minute, the speed limit, and max vehicle acceleration.	Does the model behave differently accordingly with the aspects that were changed?
8	Medium	For expansive networks there must be a way for the user to pan and zoom around the plane. Scroll wheel up to zoom in and down to zoom out.	Is it possible for the user to change the magnification level and view every part of the model?
9	Medium	The simulation must have a user-friendly way of displaying the current state, including the states of the traffic lights and locations of vehicles. The generated diagram may be simplistic, but must contain this relevant information.	Does the majority of users understand where vehicles are, and which vehicles are allowed to move through the intersection at a given time from the UI?
10	Medium	The execution of the simulation must be in a reasonable amount of time.	Is the simulation in real time or faster?
11	Low	The simulation should be able to accommodate multiple road users, including passenger vehicles, lorries, cyclists and pedestrians.	Are motor vehicles, bikes and pedestrians able to reach their destination?

Design Specifications

The main goal of the design user interface is to provide an intuitive yet precise way of creating a realistic model of a traffic environment to be tested by the use of agents. The simulation will consist of a top down 2D ‘engineering diagram like’ depiction of a road network, with lanes marked in dotted lines, and road borders in solid lines. This will be designed and inputted by the user in the first stage of creating the simulation. The state of the traffic lights at intersection entrances will be overlaid near to the queue. A design diagram is shown in figure 2 below.

Lane connections

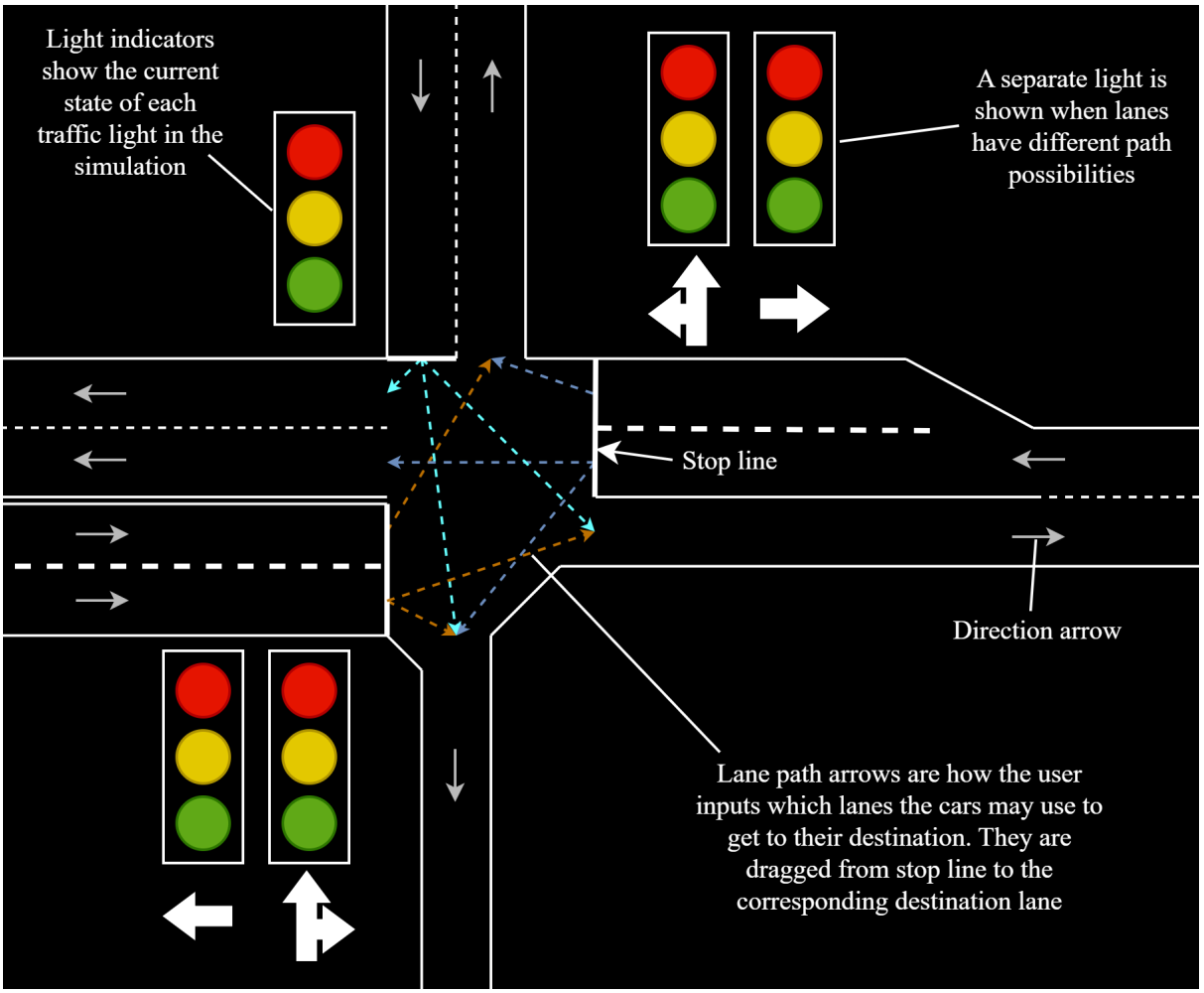


Figure 2: Early diagram of vehicle path interface

The combinations of the traffic lights are created from the nodes each lane on that road is connected to. So if a road is a single lane, and it has edges left, straight, and right, a single traffic light will be used for all vehicles. Without separated lanes, several lights have very limited utility because vehicles won’t be able to overtake the

ones waiting because they are going in a different direction. A wide road where every direction has its own lane might have three separate lights to allow for the most nuanced (potentially most efficient) combination of phases. For example the near turn lane (right turn in right-hand traffic) can often be green while other directions are because it conflicts with the lowest number of paths, this increases flow because those vehicles can go during more than one phase.

This implementation uses the concept of dragging connectors from node to node to create a connection between lanes in the intersection. This visual way of representing the paths agents will be able to take is a step in the right direction from the UI used in a lot of large scale simulation software, which remembers its days in windows XP.

Timed Traffic Light Phases

Once the user has designed an intersection layout by setting the number of lanes on every road, and designating which paths vehicles will be able to use, they must configure the traffic light management scheme.

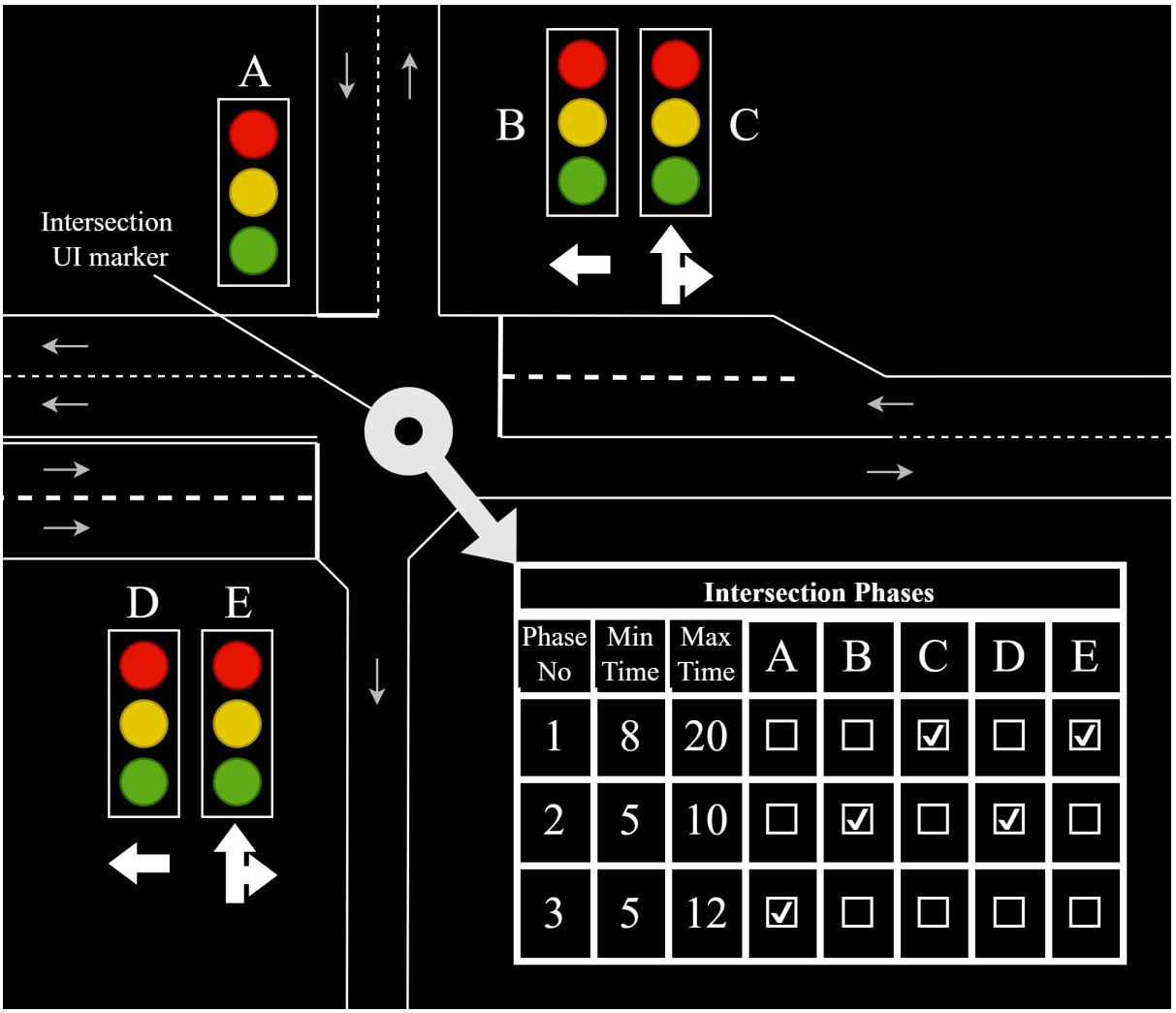


Figure 3: Traffic light phase system UI

Figure 3 shows how this is done through the UI that appears when the user hovers their mouse near an intersection and clicks on the node identifying it. A new window will open with a list of phases which the user can edit, or rearrange. Each phase represents the combination of traffic lights that are green, and red from each direction and lane. For example a common phase in traffic lights include main roads straight through with right turns, where the two higher throughput roads (C & E on the figure) will have a green light, and the rest of the smaller roads will be red. This phase is usually the longest because the largest amount of traffic flow occurs in the busier roads going straight through.

Each phase has a minimum and maximum duration configured by the user. The value depends on the expected traffic at those lights, and the number of lanes approaching the intersection, as more entering lanes allow for higher throughput, which will clear the backlog more quickly. Many modern or ‘smart’ traffic lights have sensors which detect the presence of vehicles approaching it, this is where the minimum and maximum values come in. The minimum value ensures the phase occurs for at least that duration even if no vehicles are detected waiting. This might be necessary as sensors aren't always perfect, and can report no vehicles when there are. The value can also be set to 0, meaning the phase will be completely skipped, potentially improving flow at quiet times. The maximum duration cuts a phase off once the specified time has passed to avoid causing gridlock by a phase being continually saturated. The condition for ‘vehicles present’ could also be more complex than simply cars being detected. At busy intersections a single vehicle is often insignificant for the general flow, so more restrictive conditions could be used. For example, if there are less vehicles in the queue than currently inside of the intersection, signifying the highest flow part of the phase is over, where cars are coming from every lane.

The light combinations are split into phases rather than each light because in many intersection designs there are paths which do not conflict. Two lights may be green simultaneously when they don’t need to cross to get to their destination, increasing efficiency. A four way intersection with each direction having a separate lane has 12 unique lights, but this can easily be reduced to 4 phases: 2 for the opposing road straight on with right turn, and 2 for the opposing left turns. In countries with right hand traffic, the left turn is often the most conflicting combination because it has to cut most lanes of opposing traffic, so often corresponding left turns are done as the last phase at the same time.

The progress of a single agent will be similar to the following: The vehicle will be created at one of the spawning points. Then it will continue down the road until it starts approaching an intersection where it will choose which lane to use based on the direction towards its destination. When the agent encounters a red signal, or another vehicle, it will wait. Once it is allowed to move again it will continue towards its destination. The vehicle must select the correct lane based on the connected lane edges to be able to reach its destination.

Implementation

The implementation of the application is done in the Unity game engine, which uses the C# programming language in the scripts assigned to GameObjects. I used this engine because it provides a flexible framework to visualise the simulation, this is important because the application will be very GUI oriented. C Sharp is a generally efficient high-level object-oriented programming language which enables the effective modelling of the simulation concepts, such as roads and nodes.

The general structure of the implementation is that various UI components interface with the road network object, which permanently stores the created objects, such as road segments and vehicles. When the user starts the simulation, the road network script updates the simulation to the next state ever frame, this includes the displayed state of traffic lights, spawning of cars, and statistics like traffic flow.

Creating Road Segments

The first step users will take to create a road network is to add roads, this is done by selecting the number of lanes in each direction on the road, and clicking the draw button. This will show a first point selection dot on the grid near the mouse, the user must click to select the first point on the road segment. Then a line is drawn to give an idea of how the road will look when completed. When the user clicks again, it invokes the AddNewRoad() method in the road drawing script. By default drawn elements will snap to a grid, this is to enable easier alignment of elements by the user. However if finer details need to be inputted, the snap distance can be decreased for precision.

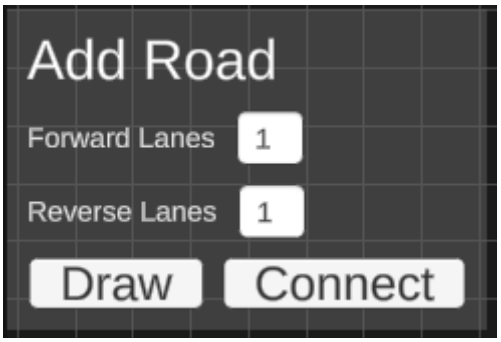


Figure 4: Left UI panel for creating and joining roads

This script calculates the corresponding parallel lines to the one selected to draw road edges, lane dividers, and median. It also creates a new RoadSegment object, which is the highest level road component of the architecture. RoadSegments are composed of LaneSegments, which are composed of Lines, which are composed of Vector2s

(points). When a parallel line is required anywhere in the code, the static function of the Line class I made is used. It takes a line, and offset and returns the line which is the offset distance away, with the same gradient.

```
// Returns a parallel line to the given one, offset by the float given
public static Line ParallelLine(Line l, float offset) {
    float newx1 = l.x1 + offset * (l.y2 - l.y1) / l.length;
    float newx2 = l.x2 + offset * (l.y2 - l.y1) / l.length;
    float newy1 = l.y1 - offset * (l.x2 - l.x1) / l.length;
    float newy2 = l.y2 - offset * (l.x2 - l.x1) / l.length;

    return new Line(newx1, newy1, newx2, newy2);
}
```

Figure 5: The ParallelLine function returns a parallel line at a distance determined by offset

There are many literal values required in the application to define the properties of objects, for example the width of lanes, which side of the road cars drive on (left or right hand) and, car acceleration. To make configuration of the simulation straightforward all of these values are stored in the Settings.cs file, where they can be easily tweaked. This is superior for maintainability because literals are not used in code and all configs are in one easy to find location with helpful descriptions.

```
// The rate at which cars will increase in speed
public const float CAR_ACCELERATION = 1f;
// The rate at which cars will decrease in speed
public const float CAR_DECELERATION = 2 * CAR_ACCELERATION;
// Cars will not increase speed if this value is reached
public const float MAX_CAR_SPEED = 5f;
// The distance at which a car is considered to have 'reached' a node
public const float DISTANCE_THRESHOLD = 0.1f;
// The distance away from an obstacle that cars will aim for
public const float STOPPING_OFFSET = 2f;
// How far the car will look for other cars to avoid collisions
public const float CAR_SIGHT_RANGE = 100f;
// Used to move raycasts to the front of cars so that they do not hit themslves
public const float CAR_LENGTH = 1.2f;
```

Figure 6: Part of settings file storing the global constants

Connecting Roads

Once some road segments have been added to the canvas, they can be combined into an intersection with the connect tool. The user clicks on markers on the ends of the road segments to select them, and then presses ‘make connection’ to create an intersection. This creates a new intersection object and switches to the lane connecting mode. This is one of the biggest advantages to this implementation, the user has the precise control of which lanes entering an intersection can go to which lanes exiting the intersection. In real life this is analogous to lane arrow markings, informing drivers where they may turn from a given lane.

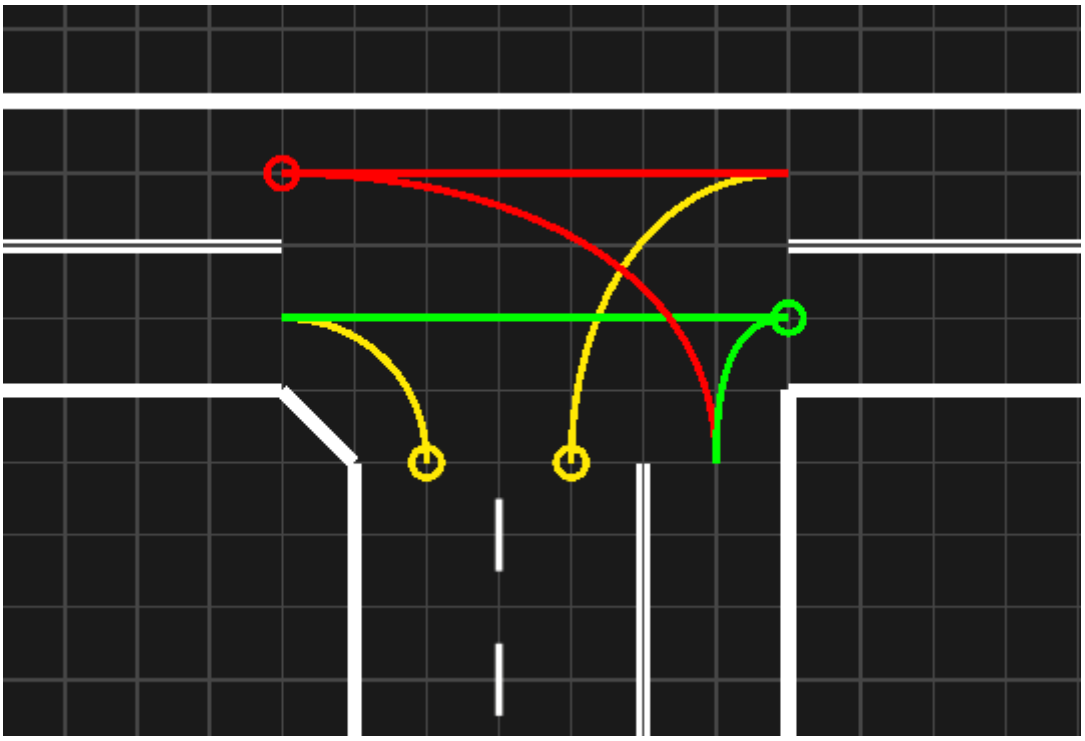


Figure 7: Example of small intersection lane configuration

Figure 7 shows an intersection which has been wired for each road to be connected to each other road, however as the bottom road has 2 lanes going into the intersection, the left one only turns left, and the right lane only allows right turns. The lines connecting lanes are cubic Bézier curves, which are a type of smooth, continuous curve ideal for connecting 2 lines without sharp corners. The cubic variant of Bézier curves have 2 control points, which act like tangents, which the line moves along in the beginning, but then fluidly deviates to the other defined point. Bézier curves are the best method for connecting lanes partially because they are visually clear and appealing, but more importantly because the vehicles in the simulation will move

along these curves when making a turn. This is a more realistic model of how a vehicle would manoeuvre, than for example sharp turns or straight lines.

In my first implementation, I used quadratic Bézier curves, this is a curve with only one control point, other than the start and end points. However this approach presented the problem of defining this control point: if a point was calculated halfway between the start and end point, it would simply result in a straight line. This means the point would have to be displaced from the direct line between the lanes towards the centre of the intersection, however this would create inconsistency in how curved the line is. In addition to that, if two lines to be connected were parallel, the control point would need to be infinitely far away, as parallel lines do not cross, by definition. Cubic Béziers are better as they fix these issues, and allow for lines with two direction changes, which is required in some intersection configurations.

The cubic Béziers curve is defined by the following equation:[4]

$$B(t) = (1 - t)^3 P_0 + 3(1 - t)^2 t P_1 + 3(1 - t) t^2 P_2 + t^3 P_3$$

Where:

- t is a float between 0 and 1 representing a ratio of a point on the line
- P_0 and P_3 represent the start and end vectors
- P_1 and P_2 are the control point points
- B(t) is the resulting vector, a point on the curve

Configuring Traffic Lights

Once lane connections have been set up, the user may enable traffic lights on the intersection. When traffic lights are enabled the user must configure them by pressing the ‘Edit Scheme’ button, this switches the user to a new mode. When configuring traffic lights, the user can add traffic phases, which are a specific configuration of traffic lights in a given moment. Phases have a duration, after which, the system switches to the next phase, this repeats in a cycle.

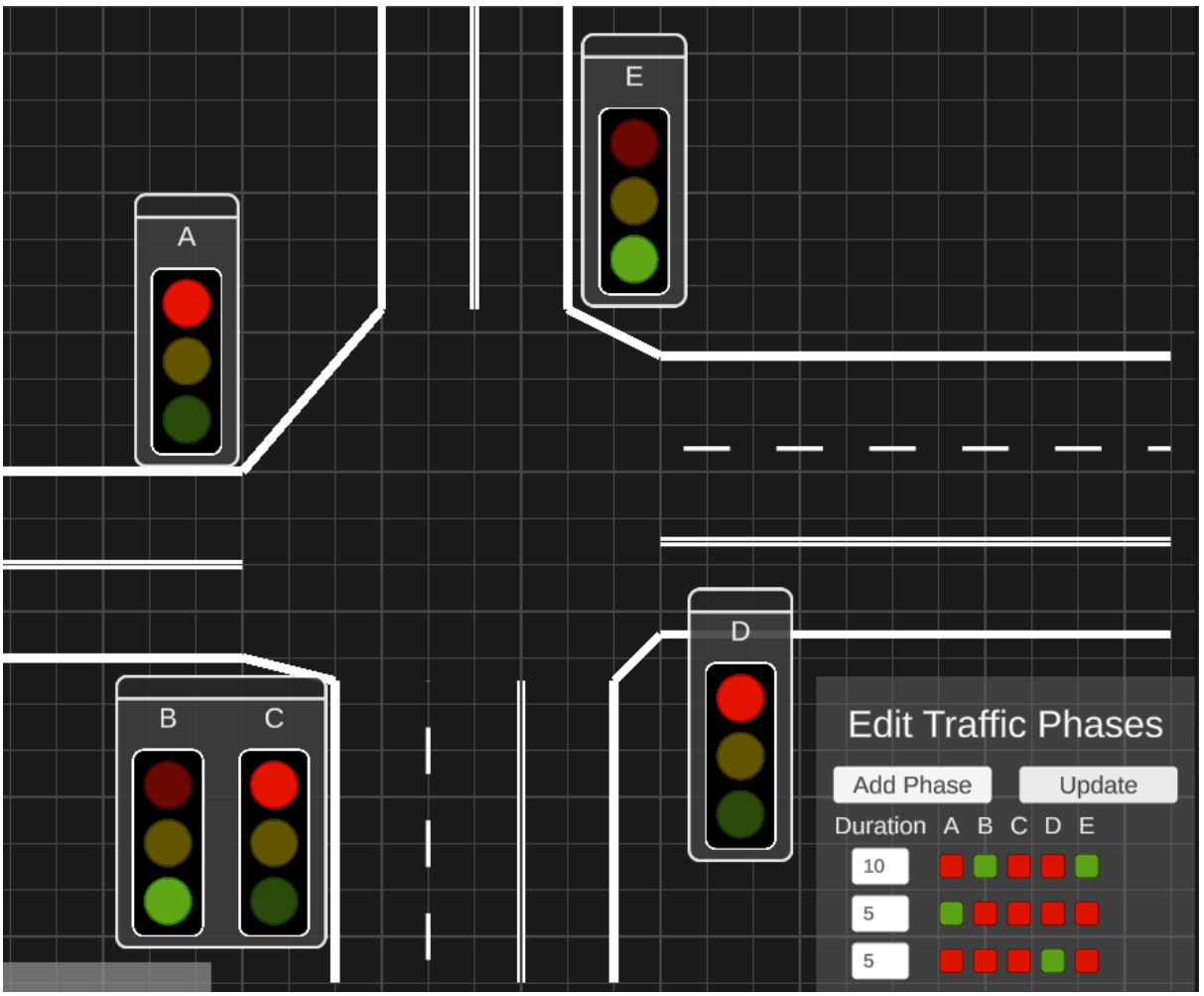


Figure 7: Example intersection light scheme configuration

Figure 7 shows an intersection with its lane connections already set up, the traffic lights are configured in the bottom left in 3 phases, lasting 10s, 5s and 5s respectively. First the traffic lights B and E are green (shown on screen), then the remaining phases are cycled through. When the ‘Update’ button is clicked the configuration of the traffic scheme is saved in the ‘Intersection’ object created earlier as a list of boolean arrays.

When configuring traffic lights the user selects between the states of green and red, however in practice, between each phase is a short transition period, the length of which is defined in the settings file. In the logic of the program, if a light is changing from red to green, the red and yellow lights are shown. When a light is changing from green to red, only the yellow light is shown, to differentiate the state transitions, like in the real world.

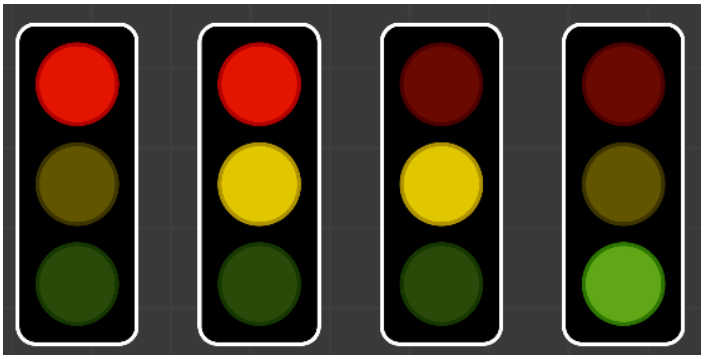


Figure 8: The four possible traffic light states

Vehicle Simulation

Upon completion of the desired road setup, the simulation is ready to be run, to find the various traffic flow properties of the system. The user may press the play icon at the top of the screen to start spawning vehicles and measuring the statistics. During the playback of the simulation ‘Car’ objects are created, and set a heading node. The roads and lanes defined by the user are made up of nodes which are connected with lines. Cars move between these nodes in a straight line, except on intersections, where they follow the points on the Bézier curves.

The distance to move a car in a given frame is defined by the equation $s = v \times \Delta t$. With v being the current velocity and Δt the amount of time elapsed since the last frame. This ensures a consistent smooth motion of the vehicle, regardless of the the performance of the hardware the simulation is running on. If the frame rate is lower, the frame times are higher, meaning the car will move further each frame.

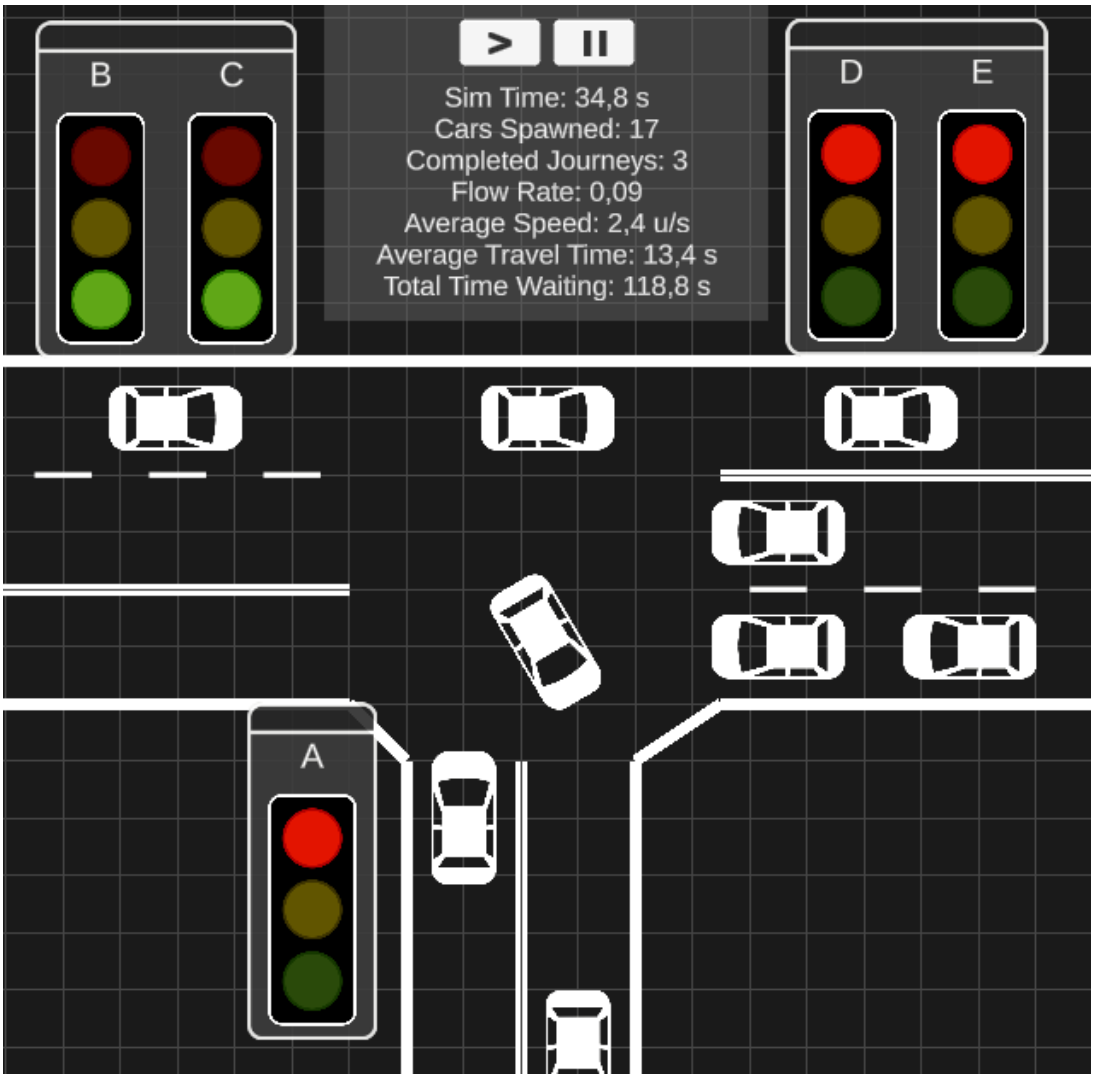


Figure 9: Simulation running on a T junction

Figure 9 shows a simple three way intersection with dedicated turning lanes on the main road. Vehicles spawn on points at the edge of the simulation off-screen, and manoeuvre through the intersection, choosing their destination randomly. The following table describes the parameters being tracked during execution.

Table 2: Explanation of tracked variables

Statistic Name	**Description**	**Network Efficiency**
Simulation Time	Time passed since the play button was pressed, starting the simulation.	N/A
Cars Spawned	The number of vehicles which have been added to the world in this instance of the simulation.	N/A
Completed Journeys	Number of vehicles which have been despawned, representing completed journeys	Higher is better
Flow Rate	Number of completed journeys divided by the time passed. Effectively journeys/second.	Higher is better
Average Speed	Average of speed values of vehicles currently in the simulation	Higher is better
Average Travel Time	The average journey length.	Lower is better
Total Time Waiting	Total time spent travelling at < 0.1u/s by all vehicles, including completed journeys.	Lower is better

Flow rate and total time waiting are the main sim results which should be monitored. By aiming to reduce total time waiting, it results in a system in which agents spend less time waiting for traffic lights to change or congestion to clear, and more time travelling to their destination. Flow rate represents the throughput of the intersection or larger traffic system. Peak time traffic jams are caused by the infrastructure not supporting the rate of cars flowing into the system vs the cars leaving the system, a higher flow rate means a higher ceiling for this condition.

The main car logic is shown in figure 10 below, it is the update function of the ‘Car’ GameObject prefab, which executes in every frame of the simulation. A prefab in Unity is a saved instance of a GameObject which can be instantiated in a script to create a copy of it in the hierarchy.

```
// Updates the cars parameters every frame
void Update()
{
    // If car is passing traffic light (entering intersection)
    if (hasReachedPoint(headingNode.GetPosition())) {
        updateHeadingNode();
        turning = true;
    }

    alterSpeed();
    // If the car is in an intersection, turning on a bezier curve
    if (turning) {
        moveCarAlongBezier();
    } else { // Car is traveling straight
        Vector2 direction = headingNode.GetPosition() - currentPosition;
        float distanceToMove = speed * Time.deltaTime;
        moveCarStraightTowards(currentPosition, direction, distanceToMove);
    }
}
```

Figure 10: Car object update function, implementing vehicle logic

Car objects first check if they have reached the road’s end node, if they have, it means they must now select a direction to turn from the lanes connected to the lane the vehicle is leaving. The turning variable is then set to true to flag that the car must move along the Bézier curve stored in ‘bezierArray’. The car must then decide whether it should accelerate or decelerate, depending on the state of the light in front of it, and on if there are any obstacles in the way. The ‘alterSpeed()’ method uses the ‘findClosestObstacle()’ to find this, then calculates the vehicle's stopping distance to determine if it can stop in time for the obstacle. If it cannot, it continues forward, this applies when a traffic light has just turned yellow, and the car is about to enter the intersection, so it should not slow down.

The following rearranged equation of motion is used to calculate stopping distance:

$$r_1 - r_0 = u^2 / 2a$$

Where:

r_1 is the stopping displacement
r_0 is the current displacement
u is the current velocity
a is the maximum rate of deceleration of the car

After the speed of the vehicle has been altered accordingly to the surrounding environment, the car must move. If the turning flag has been set to true, a point on the Bézier curve is selected using the 'moveCarAlongBezier()' method. This point is one which is closest to the distance which the vehicle must move that frame using. If the car is moving down a straight road, the moveCarStraightTowards()' method is used to set the car's new position.

Multiple Intersections

One of the most notable aspects of the software implementation is its ability to model multiple connected intersections simultaneously, mirroring a real life neighbourhood.

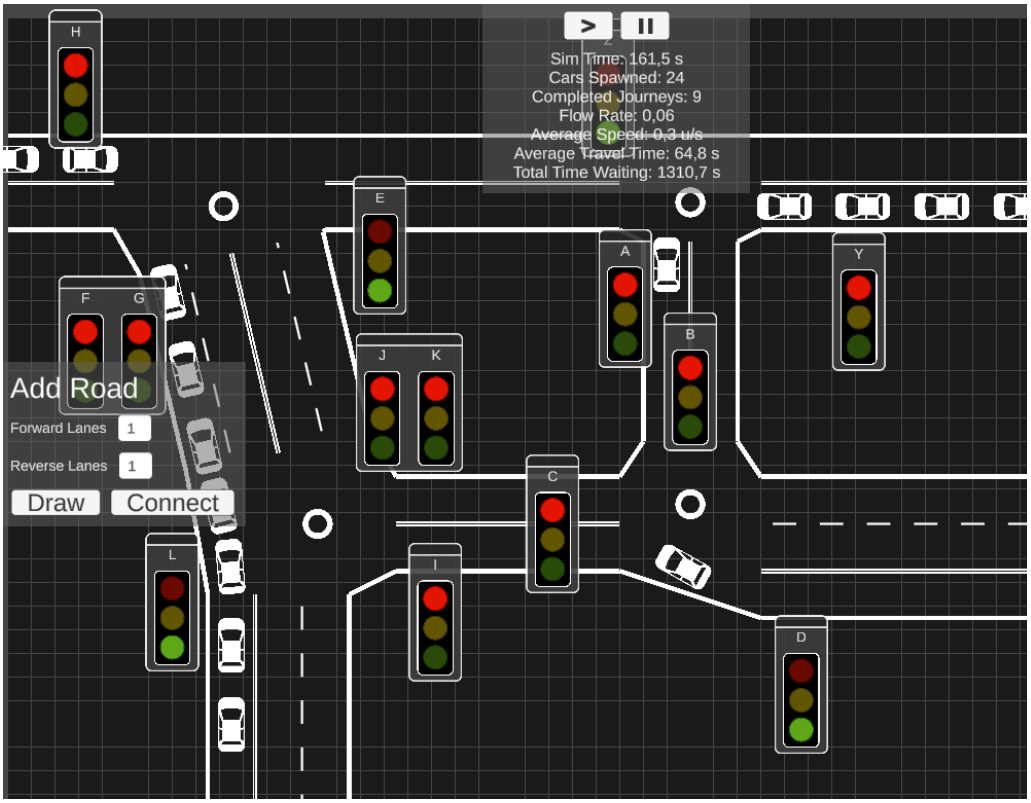


Figure 11: A zoomed out view of a traffic simulation of a road network with 4 intersections.

Maintainability

An example of a programming technique used to improve the maintainability of the software is the LightState enum. This type is used to define that state of a traffic light, which is a widely used state throughout the application.

```
public enum LightState {
    Red = 0,
    RedToGreen = 1,
    Green = 2,
    GreenToRed = 3}
```

Figure 12: Definition of LightState Enum

The use of an enum provides a clear and concise way to define and use the different states that a traffic light could be. Apart from improving drastically readability of code, it is type safe, so debugging is made easier if the compiler were to catch a value that is not part of this defined enum. The alternative to this approach would be to store the state only as an integer. In the code this would be confusing, as the reader would need to refer to the defined standard for which numbers refer to which states. Finally, using an enum can also make the code more modular and therefore, easier to maintain. By defining a set of related values in an enum, it becomes easier to reuse that code across different parts of the software. This helps to reduce duplication of code and makes it easier to make changes to the code in the future, as changes only need to be made in one place.

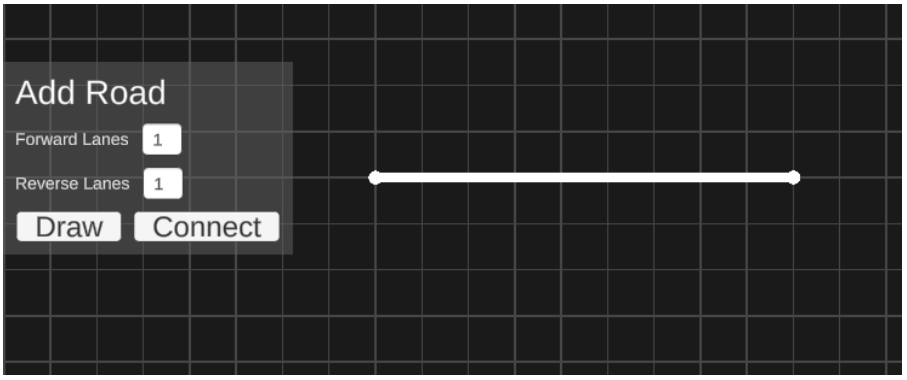
Another example of a pattern used to improve the maintainability and future extensions of the code is the use of a centralised script to manage the transition between modes and features in the program. The ‘UIFlowManager’ ensures that only one mode of edit is enabled at once. This streamlines the process of switching to different tools when creating and simulating the road network. The alternative to this approach would be to enable and disable different scripts from each feature. However this approach is not extendable, as with more features, there will be more combinations of modes to switch between, requiring hard coded and exact micromanagement of UI components. With the implemented approach, adding a new tool to the software is as simple as adding a new function to the ‘UIFlowManager’ which switches to that view. Then invoking this method from the interface components designed to switch to it.

Throughout the development of the simulation I used the version control software Git to incrementally manage the evolution of the software. Using version control correctly is an essential skill for any programmer to have. Version control software automatically creates backups of code changes, allowing programmers to revert to an earlier version of their code if something goes wrong. In addition to this, Git helps organise code by providing a central location for the source code to be downloaded from, on any machine.

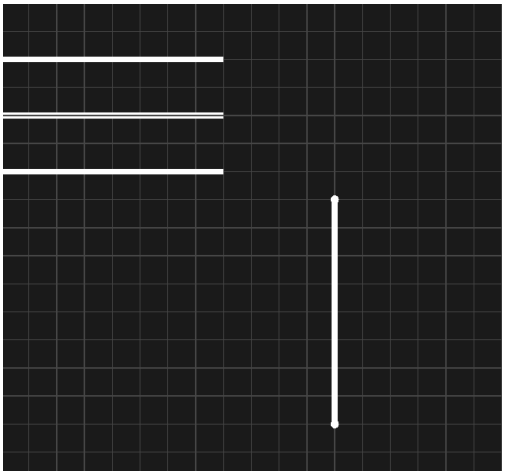
User Guide

The following procedure could be used to set up a functioning simulation environment. At any point the **arrow keys** are used to pan around and the **scroll wheel** or Q / E to zoom.
This guide is also available as a **video** demonstration: https://youtu.be/FFVYhXtyirI

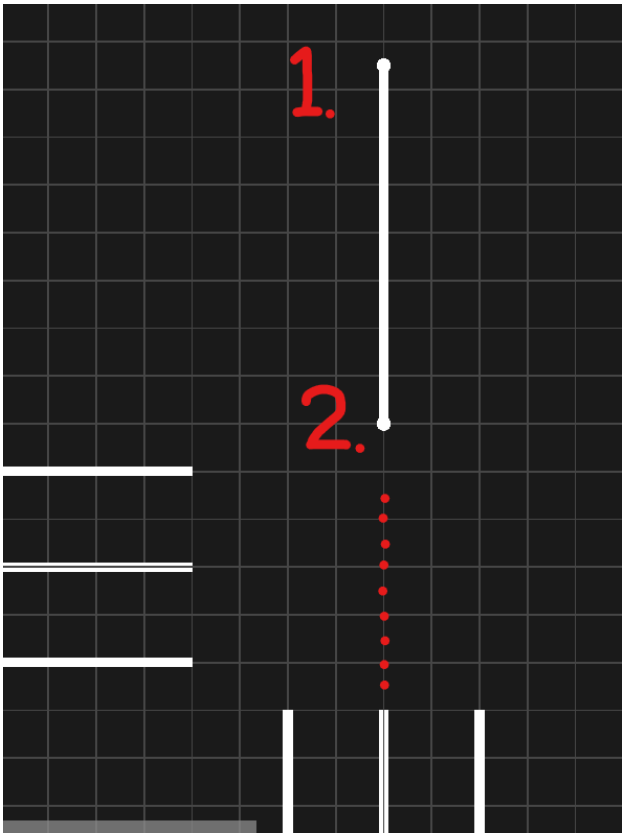
1. Open Traffic Simulation.exe.
2. Click '**Draw**' on the left of the screen.
3. Click on the grid to select two points horizontally.



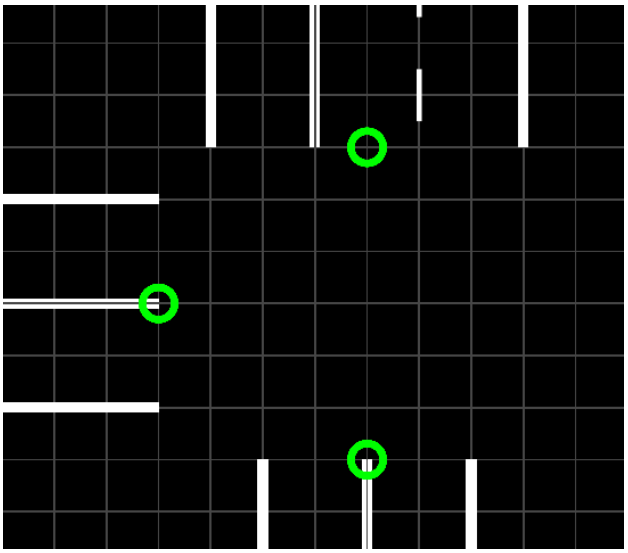
4. Click on the **draw** button again.
5. Draw another line to the right of the existing road, starting just below and to the right, going downwards. (Making sure to leave enough room for the width of the road)



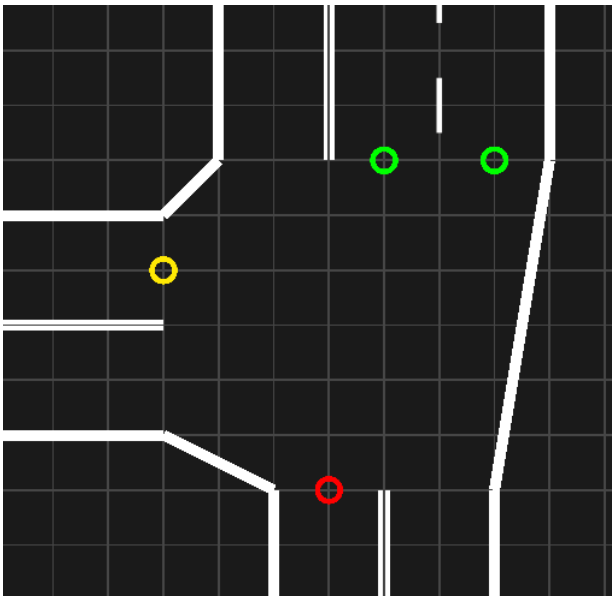
6. Change the '**Forward Lanes**' value in the left panel to 2.
7. Click the **draw** button.
8. Draw another road from above, starting **far** and finishing close to the intersection.



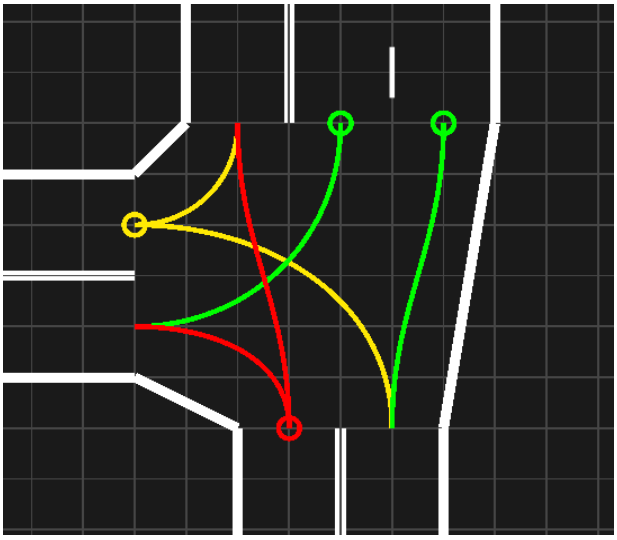
9. Click on the '**Connect**' button in the left panel.
10. Click on each of the **three nodes** making up the intersection, so they are green.



11. Click '**Make Connection**' on the left (this may take a second), the intersection should look like this:

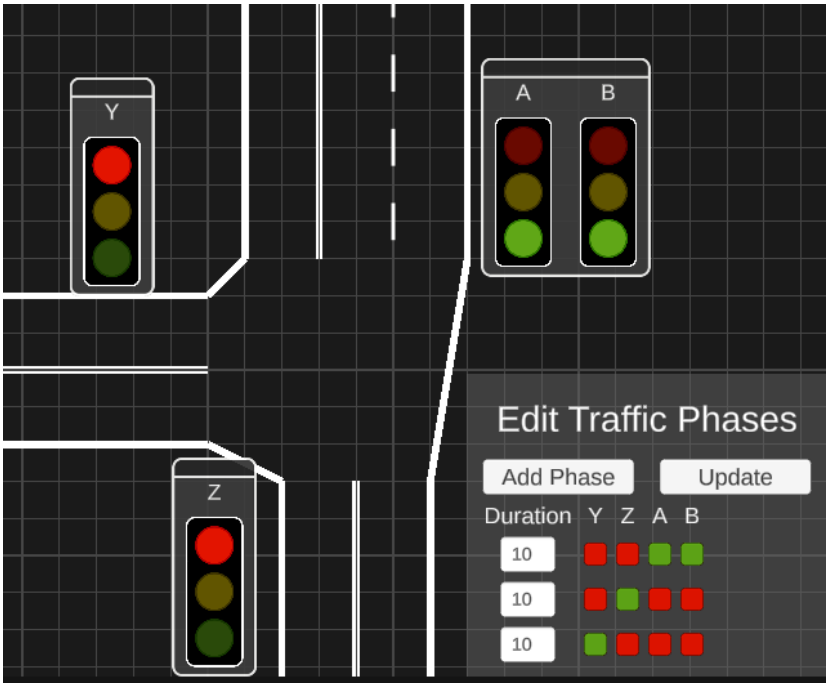


12. Click on the intersection entry markers, and connect each to at least one exit marker. **Right click** once to complete a selection.

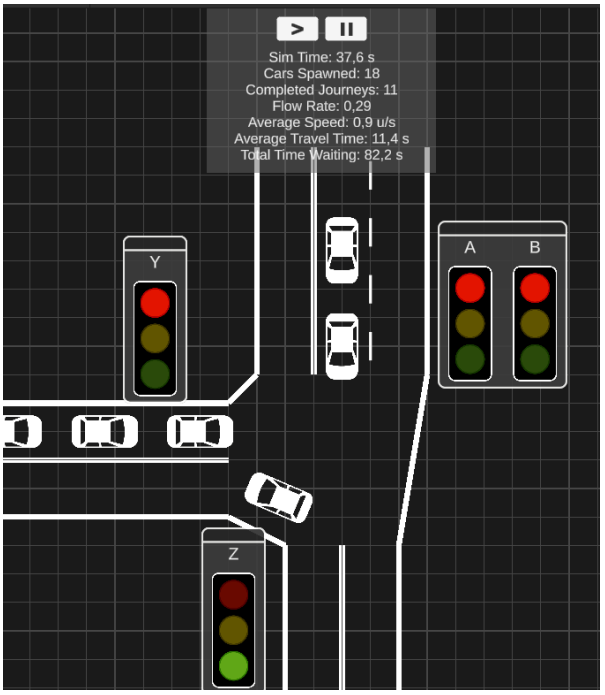


13. Check the '**Traffic Lights Enabled**' box, then click '**Edit Scheme**'.

14. Click '**Add Phase**' **twice** so there are 3 light phases. Then click on the **red boxes** so that one incoming road is green for each phase. In the example below, lights A and B are on the same road, so they can be green at the same time, in the first phase here.



15. Click '**Update**' in the bottom right. (Traffic lights can be moved by dragging)
16. Click the **play button** at the top of the screen to start the simulation.



17. (**Optional**) Click pause at the top of the screen to stop the simulation, and see the results.

Evaluation

Overall the created software successfully achieves the goals set out in the beginning of its design phase. The application makes extensive use of sophisticated concepts from euclidean geometry such as splines (e.g. cubic Bézier curves), to provide realistic trajectories for vehicles. These trajectories are designated by the user in an expressive method through the use of coloured node markers.. To facilitate multi-tool use, the software implements separate views corresponding to the function of each tool. For instance, the "Draw" tool invokes a view layer which only renders roads, whereas the "Connect" tool causes additional elements such as control points to be shown. This visual segregation helps in maintaining a more clutter-free user experience. The system is implemented using a ‘UI flow manager’ which controls which scripts are able to display elements at any given time, this generic implementation allows new tools to be added without requiring significant changes to the underlying architecture.

Requirements Testing

In the design stage of development 11 main requirements were set out to aid the creation of an effective traffic simulation, after implementation these requirements were tested. The following table shows the results of the testing conducted. 10 of the 14 requirements were successfully implemented. All of the requirements with very high priority passed the tests. The requirements which were not passed are analysed below.

Table 3: Requirements testing

#	**Priority**	**Testing Questions**	**Testing Conducted**	**Pass/ Fail**
1	Very High	Most users should be able to create a simple layout following the guide.	I sent the compiled software to two peers who have never seen it before, and following the user guide, they constructed a working simulation.	Pass
2	Very High	Do vehicles spawn and despawn correctly?	I started the software, built a road, then pressed play, cars spawned correctly, then moved across the road and despawned once they reached the end.	Pass

3	Very High	Can users connect roads to other roads at intersections?	I built 3 road segments, then connected them into an intersection. After this I was able to click on entry nodes and connect them to exit nodes. When the intersection view was closed, and reopened, the connections remained.	Pass
4	Very High	Do the correct traffic light colors appear in each phase designated by the user?	Using the intersection from the previous test, I enabled traffic lights and set them up in a configuration. After clicking update, they correctly changed colors between the phases.	Pass
5a 5b 5c 5d	High	Vehicles must accelerate and break at a rate which can be achieved by real cars and lorries. Do vehicles go through red lights or leave the road border? Do vehicles collide or overlap?	I created a few intersections and ran a few simulations. The vehicle's speed behaved realistically, and did not suddenly change. No vehicles went through red lights. In some situations where angels in the road were too sharp, the cars would leave the road border, unable to turn quickly enough. When traffic lights were configured in a way which enabled collisions, the vehicles would sometimes overlap.	Pass Pass Fail Fail
6	High	Are correct statistics about the running of the simulation returned after execution?	I ran a simulation for 30 seconds and the following correct statistics were returned: Sim Time: 30,0 s Cars Spawned: 8 Completed Journeys: 2 Flow Rate: 0,07 Average Speed: 0,6 u/s Average Travel Time: 8,0 s Total Time Waiting: 30,3 s	Pass

7	High	Does the model behave differently accordingly with the aspects that were changed?	These parameters can be edited from the Settings.cs file, however when the software is compiled, this file is not accessible. A better solution would be to have a GUI in the software to change these, or a .txt config file.	Fail
8	Medium	Is it possible for the user to change the magnification level and view every part of the model?	After opening the software I can use WASD to move around and scroll wheel to zoom.	Pass
9	Medium	Does the majority of users understand where vehicles are, and which vehicles are allowed to move through the intersection at a given time from the UI?	Both of the two test users agreed they understood the structure of the road network and positions of the cars.	Pass
10	Medium	Is the simulation in real time or faster?	I ran the simulation and it executed 30 seconds of simulation time in 30 seconds of running time.	Pass
11	Low	Are motor vehicles, bikes and pedestrians able to reach their destination if a junction is designed with them?	Only cars are implemented in the simulation, and they reach their destinations.	Fail

Road Connecting Algorithm

While implementing the software, the challenge of deciding which road edges to connect was raised. This problem turned out harder than anticipated, and I could not come up with a perfect algorithm for solving it. Despite this being purely an aesthetic GUI issue, it is a partial failure of requirement 5. The premise of the challenge is as follows:

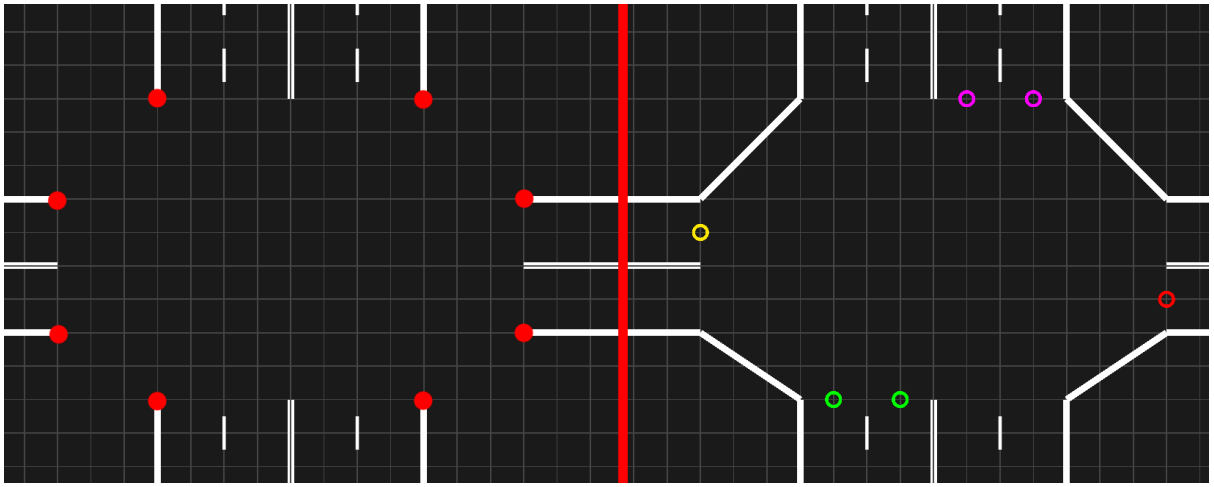


Figure 13: Diagram showing points which the algorithm must connect

A function must be devised which takes in the 8 points shown on the left side of figure 13 in red, and pairs them up to create lines in the desired arrangement shown on the right. The method used in the final version of the software is a heuristic one, which is not ideal. For each point, it finds the closest other point, and connects it to that one. This solution works in most reasonable intersection arrangements, like the one shown in figure 13, but can result in undesirable results in some situations. This is demonstrated in figure 14.

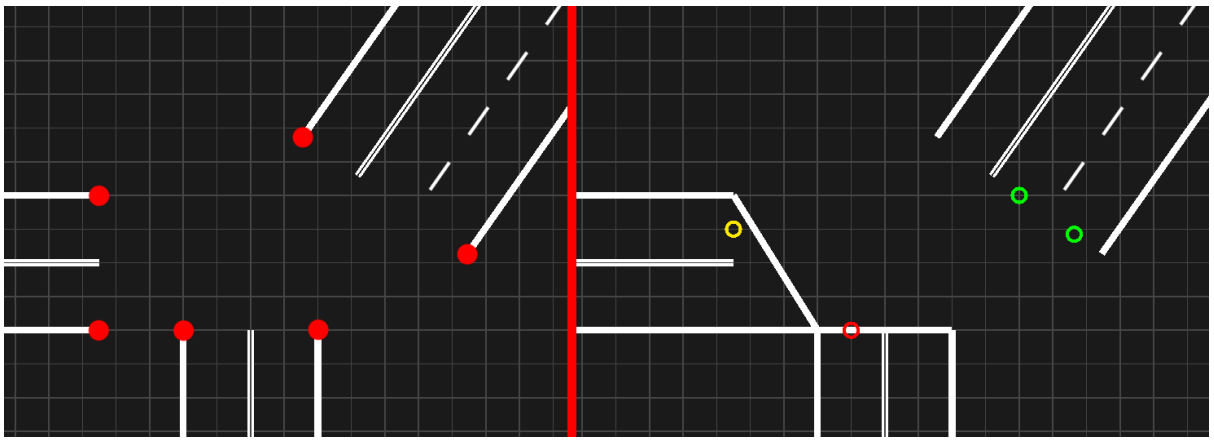


Figure 14: Incorrect intersection connection, due to non ideal alignment

A potentially more reliable solution to this connection problem could be one involving ray casting. For each point to be connected, a ray could be shot backwards, into the road, this would then iterate changing the angle of the ray away from the road, until the ray hits one of the points. Because points are infinitely small, a ray would never 'hit' actually hit the points, this could be solved by making a zone around the points which would count as hit. Unfortunately due to time constraints, I was not able to implement this.

Traffic Sensors

One of the planned features of the traffic scheme was to have the possibility of vehicle detection in front of the traffic lights, to skip phases when no cars are approaching. This type of advanced traffic light is sometimes used in real life to decrease the average waiting times at intersections. However, once again due to time constraints, I could not implement this additional feature.

Stability of Software

Due to the size of the software and amount of UI elements, it is not uncommon to run into visual glitches or inconsistencies. It's regrettable that some of these bugs remain unresolved, however through the development process, I have gained valuable experience and knowledge which will help me design more consistent interfaces in the future.

Overpasses

An extension of the system which would make it even more realistic is to implement dimensionality. Allowing the user to add overpasses to the simulation would provide a more realistic representation of traffic flow in situations where multiple levels of traffic are present, such as in complex road networks and motorways. This would enhance the software's usefulness for real-world traffic management and planning. The absence of bridges is an area that can be noted as an area for future development.

Nielsen's Heuristics

Jakob Nielsen's heuristics are a common technique for evaluating the effectiveness of the design of an interface. The following table outlines how well the implemented software achieves these heuristics. The guideline descriptions are taken from the set published in 2005 (Nielsen, 2005)[9].

Table 4: Nielsen heuristics in developed software interface

Heuristic Name	**Description**[9]	**Evaluation**
Visibility of system status	The system should always keep users informed about what is going on.	The implementation uses a status bar at the bottom of the screen to inform the user about the mode they are in, and what they can do.
Match between system and the real world	The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.	The system mostly follows this heuristic. The language used in the simulation is quite fundamental to road designing, such as ‘Intersection’ or ‘Traffic configuration’. Some of the interface is user specific, such as the connect button, for connecting roads, this is a simulation requirement, and doesn't have an obvious real life analogue.
User control and freedom	Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue.	In most situations in the software, if the user wants to cancel the current action they can use the right click button, a common action for deselection in software. A clear area for improvement would be to implement undo and redo functionality. It is often instinctive to press Ctrl + Z after making a mistake, so the omission of this feature could impact the user friendliness of the application at scale.
Consistency and standards	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.	Commonly accepted traffic symbols are used in the simulation, such as the three colour traffic lights, which users are likely already familiar with. Due to the fairly focused target audience, there are not as many platform conventions in this field.
Error prevention	Even better than good error messages is a careful design which prevents a problem from occurring in the first place.	There are no error messages in the application, rather invalid states of the interface are prevented. When inputting the duration of a phase, the input is parsed into a float and applied if it’s valid. If the user types letters into the field, it reverts to the last valid state. When selecting the number of forward lanes in a new road, and a number less than 1 is given, the same method is used.

Recognition rather than recall	Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.	As the entire construction and execution of the simulation occurs on one unified canvas, the user has the network layed out in front of them, and they do not have to remember what previous windows looked like. UI panels are used rather than overlapping pop ups, so the objects are clearly visible. This includes the crucial point in the design of a network when the user is configuring traffic lights. The lane connections are displayed as well as the traffic lights, allowing the user to refer back to this to create the optimal setup.
Flexibility and efficiency of use	Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.	Due to the nature of the application, it is not specifically tailored to novices, while I believe it is logically structured, I wouldn't expect an ordinary person to be familiar with the specifics of traffic light configuration schemes. A potential future improvement and design accelerator is for the system to automatically detect which intersections are likely to be connected, and do this automatically.
Aesthetic and minimalist design	Dialogues should not contain information which is irrelevant or rarely needed.	The information in the status panel is very concise and to the point, it is rarely more than a few words. The information displayed is also very contextually relevant, stating what the user can do in a given moment.
Help users recognize, diagnose, and recover from errors	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.	Due to the structured input system into the application (such as tick boxes for booleans), it is rare for invalid input states to occur, so there are no error codes displayed. However invalid road placements are possible, and these are not detected.
Help and documentation	Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation.	A user guide is provided which helps the user build their first fully functioning intersection, and to simulate traffic on it.

Legal, Social, Ethical and Professional Issues

Software such as automated medical diagnosis, stock market prediction models, self-driving car controllers or anything which stores sensitive user data could have implications related to the project due to risks of harm or privacy infringement. However, developing a traffic simulation tool does not inherently have many serious legal, social or ethical implications. Nonetheless, it is important to ensure that the simulation accurately reflects real-world scenarios and does not advocate any dangerous driving behaviours such as speeding. The implementation does not make use of any proprietary material requiring paid licensing, so copyright is not an issue. The only material which was not created by me is the icon used for a car, used under the 'Flaticon licence'[10]. A potential ethical consideration is if the software was used to design a road, and the inaccuracies of the simulation lead to mistakes in the real life deployment, and create a dangerous road environment. The simulation's goal is to measure efficiency of flow of traffic, so when actually implementing a design in the world, all of the safety compliance regulations should be considered separately, to avoid injury or death on the road.

Conclusion

The traffic simulation software has been successfully developed to meet the goals and requirements set out at the beginning of the project. It provides an effective and user-friendly tool for testing and analysing traffic flow patterns in a controlled environment, designed by the user. The software offers a range of features that allow the users to customise and manipulate various aspects of the network, such as road layouts, traffic signals, and vehicle behaviour. Overall, the traffic simulation represents a significant contribution to the field of traffic engineering and has the potential to be used in urban planning. I have learned, through the use of the software, that increasing the number of lanes incoming into an intersection, decreases congestion because it allows the flow of traffic to be divided into a more granular division of vehicle clusters. Therefore allowing for green lights to appear in a larger proportion of the traffic phases.

Further development and optimisation could result in a more accurate and detailed simulation, as well as a better user experience, providing even greater value. The software could be turned into a marketable product if more user experience features were implemented, such as undo, redo, and helpful hints for the first time user. Dedicated direction traffic lights on traffic lanes and overpasses could help the software become a tool used in the industry to improve real intersections and small road networks.

References

[1] HPI. (2018, July 16). Survey Reveals How Long Drivers Spend in Traffic | HPI Check. HPI Blog. https://www.hpi.co.uk/content/campaigns/how-long-drivers-spend-in-rush-hour/
[2] Beeston, L., Blewitt, R., Bulmer, S., & Wilson, J. (2021). TfL Traffic Modelling Guidelines Traffic Modelling Guidelines. https://content.tfl.gov.uk/traffic-modelling-guidelines.pdf
[3] Table 4 Forouzanfar, M.H., Afshin, A., Alexander, L.T., Anderson, H.R., Bhutta, Z.A., Biryukov, S., Brauer, M., Burnett, R., Cercy, K., Charlson, F.J., Cohen, A.J., Dandona, L., Estep, K., Ferrari, A.J., Frostad, J.J., Fullman, N., Gething, P.W., Godwin, W.W., Griswold, M. and Hay, S.I. (2016). Global, regional, and national comparative risk assessment of 79 behavioural, environmental and occupational, and metabolic risks or clusters of risks, 1990–2015: a systematic analysis for the Global Burden of Disease Study 2015. The Lancet, [online] 388(10053), pp.1659–1724. doi:https://doi.org/10.1016/s0140-6736(16)31679-8.
[4] Cubic Bezier equation used from 'TheAppGuruz'. (n.d.). How To Work with Bezier Curves In Games with Unity. [online] Available at: https://www.theappguruz.com/blog/bezier-curve-in-games [Accessed 3 Apr. 2023].
[5] Krzychu1245. (2019, January 27). Steam Workshop::TM:PE 11.7.3.0 STABLE (Traffic Manager: President Edition). Steamcommunity.com. https://steamcommunity.com/sharedfiles/filedetails/?id=1637663252
[6] Traffic Manager demonstration https://youtu.be/qCWSRAHvIyc
[7] PTV VISSIM. (2020, September 25). Wikipedia. https://en.wikipedia.org/wiki/PTV_VISSIM
[8] Bloomberg, L., & Dale, J. (2000). Comparison of VISSIM and CORSIM Traffic Simulation Models on a Congested Network. Transportation Research Record: Journal of the Transportation Research Board, 1727(1), 52–60. https://doi.org/10.3141/1727-07
[9] Nielsen, J. (2005). Heuristic Evaluation Ten Usability Heuristics. [online] Available at: https://pdfs.semanticscholar.org/5f03/b251093aee730ab9772db2e1a8a7eb8522cb.pdf.
[10] Vehicle icon used in project under free 'Flaticon' licence: https://www.flaticon.com/free-icon/car-top-view_798?term=top+down+car&page=1&position=1&origin=tag&related_id=798

Program Listings

Bold scripts are fundamental to the softwares functionality. The ordering is alphabetical.

Model

- **Intersection**: The class defining intersection objects, these are created whenever the user connects road segments. It is responsible for updating the traffic lights contained within it.
- LaneNode: Lanes are defined between two LaneNode objects.
- **LaneSegment**: Represents the data for the path vehicles will move down.
- **Line**: Defines a simple line between 2 points.
- RoadNode: Roads are built between 2 road nodes (points).
- **RoadSegment**: A collection of LaneSegments makes up a road. This script does a lot of calculations for the positions of lanes, lane divider lines and road corners.

Road Connecting

- ConnectButtonCustodian: Toggles connecting state.
- **ConnectLanes**: Responsible for the connecting lanes view. Shows and hides lane end markers accordingly, draws bezier curves between them, and connects lanes in the internal model.
- **ConnectRoadSegments**: Creates intersections upon user selection of appropriate markers. Draws lines between selected road segments to connect them.
- ItscMarkerManager: Intersection marker manager, manages the white rings used to select an intersection for editing.
- LaneMarkerManager: Similar to above, but for lane end markers. An animation is shown when the mouse is hovered.
- RoadEndMarkerManager: Controls the markers displayed when connecting roads, these toggle colour when clicked.
- **SelectIntersectionTool**: This manages the highest level view in the application, where the user can select intersections in the entire network to edit.

Traffic Lights

- EditLightsButtonManager: Implements on click actions of traffic lights enabled toggle and edit config button.
- **EditTrafficSchemeView**: Central script for editing traffic lights mode. Creates traffic light panels and enables the configuration panel.

- **LightDisplayPanel**: Creates a panel for storing traffic lights, which can be dragged.
- LightPhaseToggle: Small script which manages the state of ‘Toggle’ UI elements when configuring light phases.
- **TrafficLight**: Manages which traffic light asset is displayed inside of traffic light display panels. Also hosts the LightState enum, which is used all over the application.
- **TrafficPhaseRow**: Represents traffic phase configuration, controls toggle states and phase duration.
- **TrafficSchemePanel**: Main script for configuring traffic lights. Manages the configuration panel. Reads in and sets configurations into internal models using the ui components. Creates TrafficPhaseRows.

Main Folder

- BackgroundGridBuilder: The first script that executes in the program, draws the background canvas from scratch.
- **BezierCurveDrawer**: Does all calculations for Bezier curves, given start and end points.
- CameraManager: Moves the camera based on inputs.
- **Car**: Represents the internal model of the cars. Controls all vehicle movement behaviours, such as calculating stopping distance. Stores car properties like speed.
- **LineDrawer**: Script for drawing lines. Attached to line prefabs.
- **NewRoadDrawManager**: After a road selection has been made, this creates the visual representation of that road, and uses the internal model to find points to draw lines between.
- NewRoadPanelManager: Validation for number of lanes in new road panel.
- **NewRoadSelectionTool**: Main script for the new road drawing tool. Creates lines and dots to help users select points to make a new road between.
- **RoadNetworkManager**: One of the most central scripts. Stores the road network and vehicles on it. Orchestrates traffic lights and vehicle updates. Handles simulation statistics.
- StatusBarManager: Changes text displayed in status bar.
- **UIFlowManager**: Controls modes of application. Scripts call methods from this to switch to different tools and modes.