Virtual Traffic Simulation

by

Jonathan Ruttle, BA, BAI

Thesis

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science

University of Dublin, Trinity College

September 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Jonathan Ruttle

September 10, 2008

Permission to Lend and/or Copy

Acknowledgments

The author would like to thank his parents for all their help and support during the implementation and writing of this project. The author would also like to thank Dr. Steven Collins for all his guidance and advice during the project. Without these people this project would not have been possible.

JONATHAN RUTTLE

University of Dublin, Trinity College
September 2008

Virtual Traffic Simulation

Jonathan Ruttle
University of Dublin, Trinity College, 2008

Supervisor: Steven Collins

This project studies and implements a comprehensive real-time traffic simulation system. The system uses the real-time physics engine Bullet and a custom made driver behaviour model to allow a vehicle agent to interact with a number of traffic situations. The simulation can be dynamically generated from four input files allowing for easy alteration to the road network, traffic light timings or individual driver behaviours. Within the project a number of sensing systems were tried and tested to allow the virtual driver to receive input from the environment. The system presents an easy way to investigate emergent behaviours on a larger traffic flow level and how these may be affected given a change in the road layout or traffic lights.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Planning authorities world-wide have been concerned for years in developing traffic plans for their city centres that will ensure proper development, ease of access for their citizens and reduction of traffic congestion and air pollution.

This work describes the use and adaptation of techniques utilised in the interactive entertainment technology industry in developing a real-time traffic simulation tool for the study of traffic flow in city centre streets and road networks. The tool employs an interactive approach between vehicles, traffic lights and the streets and enables users to visualise a real road network and the vehicles which drive on it. The technique developed can be changed to numerous street layouts due to the flexible adaptable extensible method with which it has been implemented.

In particular this project implements an agent-based traffic simulation system with the following features:

- Driver agents capable of navigating roads, t-junction, crossroads and traffic lights
- A spatial partitioning technique using roads which is used for quick reliable local space querying
- A pre-timed traffic light system to simulate traffic lights at crossroads and other junctions to control traffic flow

- A finite state machine which has been incorporated into the driver agent to allow for the easy addition to the driver capabilities
- A Bullet Physics raycast vehicle [1] to ensure realistic vehicle response to driver input
- Dynamic generation of roads, waypoints and driver behaviours from text input files to allow easier creation of new maps and drivers
- Independent driver behaviours to reveal greater realism within the simulation

Computer simulation is a powerful tool for analyzing complex and dynamic scenarios. It provides an appealing approach to study and analyze situations which would be exceedingly expensive to implement in the real world. This powerful technique helps the decision making process. Agent-based traffic simulation has emerged as an efficient tool to investigate traffic phenomenon.

The layout of this thesis is as follows: Chapter 2 reviews the current state of the art within this field including explaining the different kinds of traffic simulation systems described as, macroscopic, mesoscopic and microscopic. Chapter 3 outlines the original design and plan for developing the traffic simulation system. The implementation is detailed in Chapter 4. Chapter 5 critically analyses the implementation and discusses both the benefits and drawbacks of the system and the results it offers. Chapter 6 presents the conclusions and highlights possible future work relevant to the project.

[1]See section 4.3 for further details

Chapter 2

State of the Art

There are many different types of traffic simulation models and tools. Loosely they can be broken up into three categories macroscopic, mesoscopic and microscopic. There can also be models which combine two or more of these categories together.

2.1 Macroscopic Traffic Simulation

Macroscopic traffic simulation models tend to use fluid flow algorithms to model traffic as a continuous flow. There are numerous algorithms used to do this. Smoothed particle hydrodynamics and the Navier-Stokes equations are just some of the equations used.

Chronopoulos [1] designed and implemented a traffic-flow simulation system capable of simulating freeway traffic flow in real time. The design was mapped onto a parallel computer architecture to improve the execution time, which resulted in being able to simulate a 2 hour traffic flow simulation on an 18 mile freeway from 2.35 minutes on a single processor computer to only 5.25 seconds on a parallel traffic simulation system.

Chevallier [2] presents a dynamic macroscopic model which includes a system that introduces a traffic light delay into a stream of vehicles entering a freeway from a sideroad to accurately represent the average vehicle delay and queue length at such situations.

2.1.1 Advantages of Macroscopic Traffic Simulation

The main advantage of macroscopic traffic simulation is that it can simulate large networks of large systems particularly well. Macroscopic simulation can accurately represent traffic on motorways and dual carriageways due to the way that traffic flows along these highways.

2.1.2 Disadvantages of Macroscopic Traffic Simulation

The main disadvantage of macroscopic traffic simulation is that the details of any localised traffic is smoothed out. This means that this sort of system is not very good for modelling city centre traffic scenarios.

2.2 Mesoscopic Traffic Simulation

Mesoscopic traffic simulations contain individual agents to represent real vehicles. Unlike other systems this simulation is driven by the roads and junctions which pass these individual vehicles around like matchsticks. The vehicles themselves have no intelligence compared to the roads and junctions which use speed-flow-density relationships and queuing theories to determine how and when to pass these vehicles around.

Ben-Akiva [3] develops a mesoscopic representation of a road network based on speed-density and queuing models to capture traffic dynamics and interactions as part of a simulation system designed to perform real-time estimations of the current state of the road network and to help predict future traffic conditions so it can guide travellers in an effort to minimize travel time.

Mahut [4] also develops a mesoscopic representation of a road network to allow quick calculations to be done for testing possible route travel options to determine the best option for travel.

Yoginath [5] presents a mesoscopic traffic simulation system that can run on a parallel computer architecture. This improves the speed at which the simulation can be run.

The current implementation is limited to a single lane per direction per road segment, which cannot be used to represent multiple lane networks.

Burghout [6] shows a new mesoscopic model which contains the speed-flow-density relationship and queuing theory but also has new features such as start-up shockwaves. The paper highlights the need for high quality microscopic simulation systems to accurately represent more complicated areas of traffic networks.

2.2.1 Speed-Flow-Density Relationships

Speed, flow, and density are all related to each other. In general under uninterrupted flow conditions the relationship between the three values is easy to understand. The flow, being the number of vehicles passing a point per hour, is equal to the average speed at which all the vehicles are moving multiplied by the number of vehicles per kilometer, that is, the density of traffic. A problem occurs when the combination of speed and density reach some critical value which maximises the flow. The modern traffic jam, where the density is very high but the speed is very low results in a very low flow. Basically the relationship determines a wait delay that the driver incurs when travelling a road.

2.2.2 Queuing Theory

Queuing theory is the mathematical study of waiting lines. The theory enables mathematical analysis of several related processes, including arriving at the (back of the) queue, waiting in the queue (a traffic jam), and getting to the front of the queue. The theory uses several performance measures including the average waiting time in the queue to determine when a vehicle will get to move on. The general idea is that the junction object will use this theory to determine how much time to wait until passing the vehicle onto the next road object.

2.2.3 Advantages of Mesoscopic Traffic Simulation

An advantage of the mesoscopic traffic simulation is the ability to run the simulation at high speeds so that within a short period of time you can find out what the traffic situation for a certain route would be over a long period of time. This means this simulation is very good for determining the best route a traveller should take to get to his destination.

2.2.4 Disadvantages of Mesoscopic Traffic Simulation

A disadvantage of the mesoscopic traffic simulation is the need to determine speed-flow-density information for the roads of the network which you wish to model.

2.3 Microscopic Traffic Simulation

Microscopic models capture the behaviour of vehicles and drivers in great detail, including interactions among vehicles, lane changing, response to incidents and behaviour at merging points.

Cremer [7] demonstrates the effectiveness of a hierarchical, concurrent state machine as a framework for programming the behaviour of synthetic entities including that of autonomous vehicle behaviour in a real-time driving simulation.

Ben-Akiva [8] simulates a road network to evaluate dynamic traffic management systems. The system uses a microscopic traffic flow simulator to test how the traffic management system performs.

Ehlert [9] develops and presents a reactive driver agent to simulate a vehicle. The agent is capable of tactical-level driving and has different driving styles. The agent continuously makes control decisions in order to keep its vehicle on the road and reach its desired destination safely. The simulation tries to use realistic values for the vehicle's acceleration and turn radius but these values are only estimates and so may not accurately represent the real vehicle's capabilities.

Al-Shihabi [10] further develops the hierarchical, concurrent state machine presented by Cremer [7]. The framework presented consists of four units, the perception unit, the emotions unit, the decision making unit and the decision-implementation unit. These units use fuzzy logic to make its calculations. The paper outlines a very interesting driver framework, but does not include any implementation.

Wang [11] starts to build on the framework as laid out by Al-Shihabi [10] and creates agents capable of driving through traffic light controlled intersections. The agents have direct control over their acceleration and deceleration, which may be unrealistic. The agents are capable of determining if there is a vehicle in front of them, but the paper does not explain how this is accomplished.

Zhang [12] creates a simpler agent model than Wang's [11] which enables him to accelerate the calculations and therefore run larger scaled simulations. The simulation is currently only a single-lane traffic system.

2.3.1 Advantages of Microscopic Traffic Simulation

The main advantage of microscopic traffic simulation is that instead of using general traffic-flow models, traffic becomes an emergent property of the interaction between the agents. Another important point is that the agents can exhibit human-like behaviour ranging from slow and careful to fast and aggressive driving styles.

2.3.2 Disadvantages of Microscopic Traffic Simulation

The main problem with microscopic traffic simulation is how to reproduce realistic patterns of traffic flow at both macroscopic and microscopic levels with restricted computational resources

2.4 Other Simulation Systems

2.4.1 Hybrid Traffic Simulation Systems

A number of papers present traffic simulation systems which merge microscopic simulation with either macroscopic simulation or mesoscopic simulation. This allows both the large scale and fine detail to be done at the same time in the areas that require it.

Sahraoui [13] implements a microscopic-macroscopic traffic simulation system. The paper focuses on two main issues, the calibration and validation of the system and the ability to handle path-related issues on large networks.

Wittmann [14] also implements a microscopic-macroscopic system. This system focuses on the traffic introduced by a major sea port in Hamburg, Germany. This system has a small highly detailed simulation around the port and a large low detailed simulation surrounding that.

Burghout [15] implements a mesoscopic-microscopic traffic simulation system. Again the areas of interest are simulated using the microscopic system and the surrounding network in lesser detail with the mesoscopic model. The paper focuses on an adaptive traffic light system and compares it to a fixed-time system.

2.4.2 Driver Simulator Systems

Another area of research is that of creating a driving simulator, like a flight simulator but for a vehicle. This area of research tries to solve problems of generating a realistic world for the driver to view and interact with. Once a simulation system is built it allows for experimentation with how different information will effect a driver.

Jin [16] develops a virtual-reality based driving–traffic simulation system to study how different information presented to the traveller will effect the traveller's decision–making process. Particular effort was made to make sure that other vehicles being simulated would react to the driver's decisions.

A paper by Ono [17] is solely based on efficient and effective image generation for a virtual driving simulation system. The system works on mixing geometric model and image based model approaches to create a realistic syntheses of the driver's world.

Nanyue [18] created a driving simulation system which can present different kinds of roads, the time of day and variation in weather to the driver. The system also includes virtual vehicles that have dynamic intelligent behaviour.

2.4.3 Traffic Light Control Systems

Traffic light control systems have a huge effect on the flow of traffic and a number of traffic simulation systems have been developed with the sole purpose of improving the traffic light control system.

Hewage [19] outlines a special-purpose simulation tool for optimizing traffic signal light timings. The tool is capable of optimizing light timings at single junctions as well as networks of junctions.

2.5 Steering Behaviours

Reynolds [20] paper outlines how to construct basic motion for autonomous agents in a three level hierarchy of: action selection, steering and locomotion. The paper presents a number of common steering behaviours and how they can be implemented.

2.6 Analysis

Burghout [6], [15] places importance on the need for good microscopic simulation to get the interaction between vehicles correct at dense complex road networks. Simpler solutions can be used in motorway and dual carriageway models where the interaction between vehicles is relatively simple. But finer detail is required to achieve accurate representation in city centres.

Ehlert [9], Wang [11] and Zhang [12] present very good driver behaviour models to control the vehicle agents within their systems, but they do not guarantee that the movement the vehicle performs is physically correct. Therefore even though the driver agents perform in a realistic way the vehicle is capable of very unrealistic movement. An important part of the microscopic traffic simulation is that the individual vehicles only perform what real vehicles can do otherwise the resulting interactions between the drivers may be unrealistic.

Chapter 3

Design

By modelling correctly the physics of a car and a range of driver behaviours a number of interesting traffic phenomenon will emerge.

3.1 Plan

The original plan for this system was to start with getting a simple vehicle up and running. The main requirement for this would be to utilise a real-time physics engine to do all the physics calculations of how the vehicle should react to driver input. The initial vehicle would be user controlled but the user would subsequently be replaced with a driver agent.

A system of roads or road network would need to be developed. Here consideration needed to be into how a road network might be represented and what a driver would need in terms of information about that road network.

The next step would be to develop the driver agent, giving it some ability to navigate a road, a way to decide where to head and how to get there.

Then there would be a need to increase the number of vehicles and so the number of driver agents within the system. Once this was done the driver agent would have to be given some concept of the rules of the road so that the vehicles wouldn't just crash into each other. Along with the rules of the road there would be a requirement for traffic control mechanisms like traffic lights which would have to be developed and integrated.

After that the map size would be increased to see on a larger scale how all the driver agents were interacting and to see if there was any emergent behaviour.

During the entire time, consideration would be given as to how to incorporate driver behaviour into the system. This would be required to show how aggressive drivers with road rage and passive drivers would appear within the system.

At a later stage graphics could be improved to give the system an impressive look and feel, resulting in a more realistic simulation.

The final step of the project would be integrating the system into 'Virtual Dublin' which is part of the Metropolis Project. This is a system that currently shows Dublin city centre with a large number of pedestrians walking around. This would allow the system to explore some pedestrian - vehicle traffic situations.

3.2 Requirements

The main requirement for the system was to first create a framework in which the whole system could be developed.

The requirements for the framework were that it should be:

- Robust
  - Each subsystem could have access to the necessary information it required but not information it didn't need
  - Because it isn't known at the beginning, the framework should be robust enough to withstand changes in how the system communicates
- Modular

  - Each subsystem, like a control system, would have a standard input and output
  - Each subsystem would be easy to replace if a better subsystem were conceived
  - A replacement subsystem should not affect any of the other components

3.3 Methodology

The methodology would include:

- Use of previously developed software modules utilised for the interactive entertainment industry
  - OGRE3D real-time graphics engine
  - Bullet Physics real–time physics engine
- Use of the Microsoft Visual Studio 2005 as the development environment
- SVN repository for:
  - Version control
  - Backup
  - System roll–back
  - Seamless transfer to multiple computers

A philosophy employed in the design and implementation was to use an iterative approach to move the project forward, learning and refining the plan from previous steps.

The system was implemented in C++. The main reason this was done was because C++ is a robust object oriented programming language that is fast, mature and well supported. It would facilitate future integration with Virtual Dublin which is also written on C++.

3.4 Goals

- To develop virtual vehicles, controlled by drivers with variable driving habits
- To operate and navigate in a network of streets
- To have the capability of adapting and reacting to the movement and turning of other vehicles, and
- To respond to traffic light signals

Chapter 4

Implementation

This Chapter sets out details of the implementation of the project, the issues that arose, the approach to resolving them and a description of their solutions. Issues for which acceptable solutions were not achieved or completed are also highlighted. These issues were not integrated into the final system. The entire source code plus a compiled version of the program is included on a CD at the back of the thesis.

4.1 Framework

A framework was a necessary requirement as there needed to be separation between different components in the system as not all of them would be fully developed. Each section should be 'plug and playable' and switchable at a later stage if a better solution came along. There were four main design issues that needed to be resolved;

- The world in which the vehicles operate
- The operation of the vehicles
- The rules in which the driver operated, and
- The driver control systems

The framework which evolved is set out in Figure 4.1.

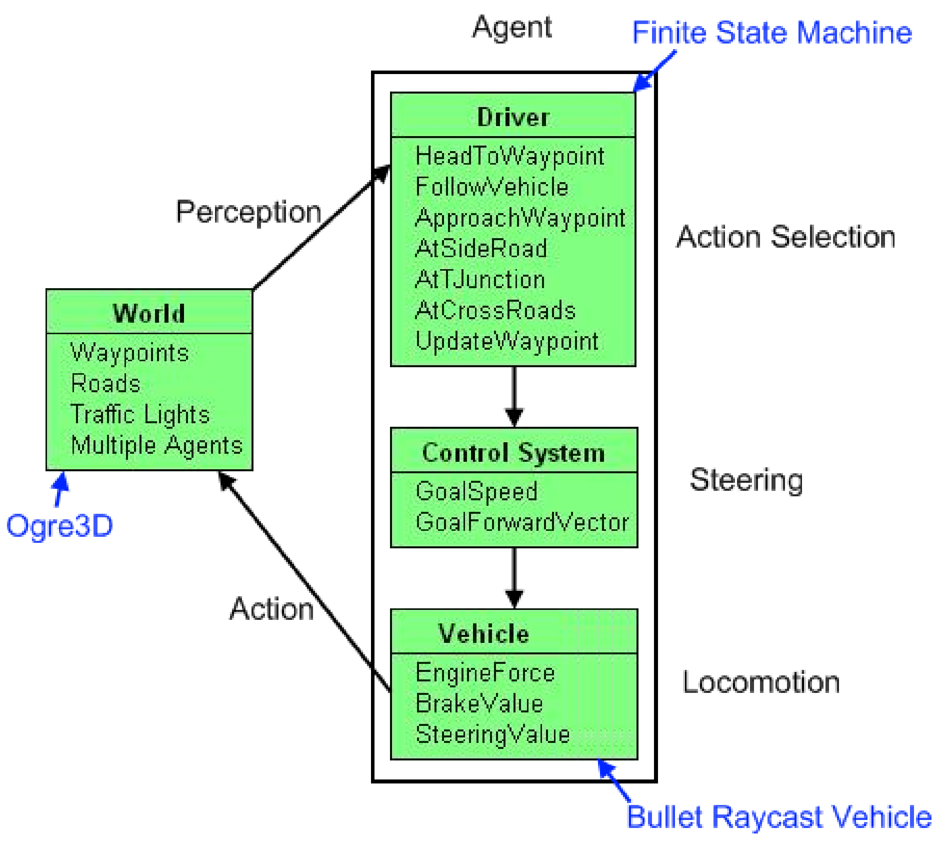


Figure 4.1: Framework of traffic simulation system

4.2 World

There were two issues that needed to be considered when deciding how to represent the world. The first was what needed to be represented, and the second was how that information is going to be used by the driver agent to navigate around the world. Reynolds [20] presents a number of common steering behaviours which use target points as positions which the agent needs to head towards. Buckland [21] confirms these steering behaviours as a good approach to how characters move. The key element that can be deduced from this is that there is a mature set of behaviours for getting a character to move towards a point. This means that a logical way to represent the world is in terms of points. Both Reynolds and Buckland outline a ‘path following’ steering behaviour which creates a steering force that moves an agent along a series of waypoints forming a path. A vehicle when navigating a road system is just moving along a path to get from its starting point to its destination. By representing a road network as a series of waypoints, where each road has a waypoint at its beginning and its end, and more waypoints along the road if the road is curved, any route can be created as a series of waypoints.

These waypoints allow for path following abilities, but do not give any information as to how all the waypoints are connected together. A roads data structure was developed. This data structure holds information relating roads to waypoints and how one road connects to another road. This information joins all the waypoints together to create a road network.

Another feature within the world is traffic lights. This traffic control mechanism dictates right-of-way at various junctions, overriding traditional right-of-way rules.

4.2.1 Waypoints

As noted above the world is made up of waypoints. Each road has a waypoint at the beginning and at the end and may have any number of waypoints in the middle if the road is curved.

Each waypoint has additional information attached to it more than just its position. This information is available to the driver to allow the driver to make decisions. The waypoint information is as follows:

- Position: This is the real world position of the waypoint.
- Orientation: In this world each road has two lanes, the vehicle should head to the left lane. To determine the left lane requires the orientation of the road.
- Speed Limit: This is the speed limit of the road the waypoint is attached to.
- Width: This is the width of the road at the point of the waypoint. It is used to determine how far left the vehicle should be of the waypoint.

- Junction Type: When exiting a road the driver needs to know what type of junction he is coming to so he can decide what he has to do.
- Traffic Light: Sometimes waypoints can have traffic lights attached to them to control who has right of way at a junction.

A sample waypoint layout can be seen in figure 4.2.

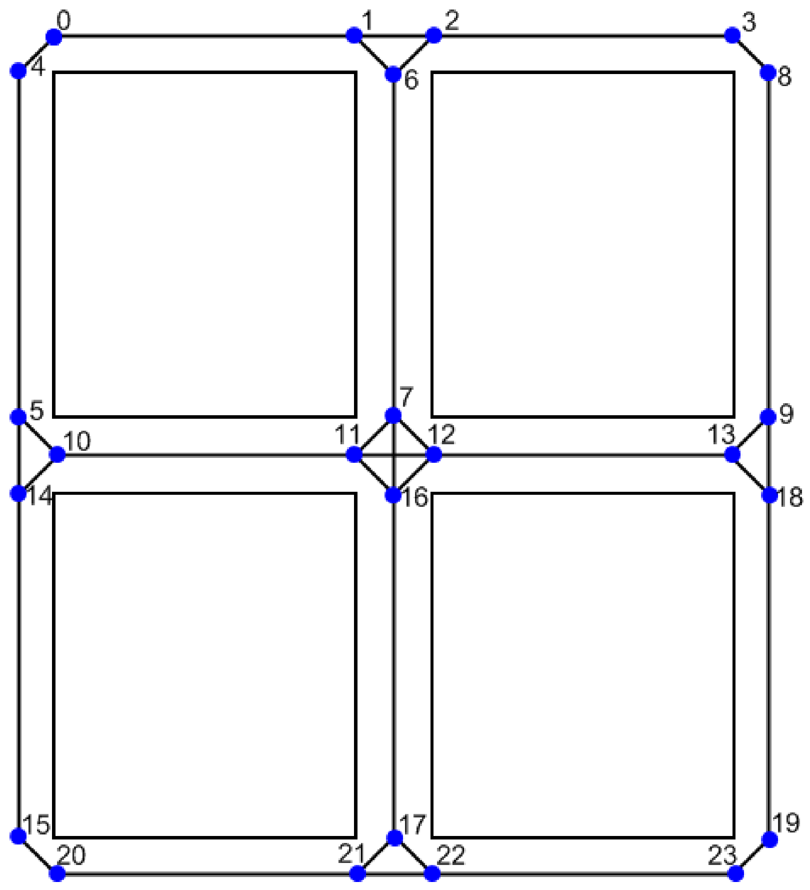


Figure 4.2: Sample world waypoint layout

4.2.2 Roads, a Spatial Partition Plan

When dealing with a multi-agent system, there is a need for each agent to know what other agents are around them. This was initially done by each driver having global knowledge over all the vehicles in the world. To reduce this down to only the agents in close proximity, a distance test was preformed against every vehicle. This turned out to be a very expensive calculation to do when the number of vehicles increased.

Ericson [22] Chapter 7 on spatial partitioning was consulted to decide on a system to help in calculating where each vehicle was in relationship to the other vehicles. Each of the techniques found allowed any agent to be anywhere in the world. This seemed to be a bit wasteful as within a traffic system the vehicles will always be on a road. This logically leads to a spatial partition technique which used roads as a way of local space querying.

The waypoints were excellent for just heading to, but when a higher level of information was required in considering where a vehicle was and how a waypoint connected to another waypoint, the waypoint structure was not ideal for storing this information. So a road network data structure was created. This data structure holds all the information that is needed.

The extra information consists of the following:

- Name: Not really used but roads do have names.
- List of Waypoints: A list of two or more waypoints that when connected together describe the centre line of the road.
- List of Connections: This list holds the information of what roads this road is connected to and how it is connected to those other roads.
- List of Vehicles: When a vehicle is on a road it registers itself to that road and when the vehicle leaves the road it deregisters itself.

The list of waypoints and the list connections in the road data structure is static but the list of vehicles is continuously changing. When a driver is about to drive onto a road he registers himself to that road, this adds a pointer to himself in the list of vehicles. The driver also adds which waypoint he is heading to and what waypoint he will head to after that. When a driver has successfully exited a road and entered the next road he deregisters himself off the old road.

4.2.3 Initializing Waypoints and Roads

Even in a world with only 12 roads there are 24 waypoints and 44 connections between roads. This is a lot of raw data. Initially it was hard coded into the program, but this posed a problem as there was a requirement to have a number of different worlds and be able to switch easily between them. Therefore the initialization of this information was switched to a system that loaded the data from a text file. To change the layout of a road network just requires changing a raw text file. When the simulation is started the new layout will be dynamically generated.

4.2.4 Traffic Lights

In order to expand the realism of the model, traffic lights were added. Initially it was thought that a traffic light would be a very simple three state machine with green amber and red states and maybe some filter lights. Unfortunately that is a simplistic view. There is very rarely a single traffic light on its own. Generally there are three or four sets of lights each synchronised creating a network of traffic lights.

Figure 4.3 shows a sample traffic light network that is used at a crossroads. Each column represents a traffic light state and each row represents a different traffic light which is connected to a specific road at the crossroads. Each state is active for a predetermined period of time before switching to the next state. In general the traffic light network is giving each road right-of-way for a period of time before switching to another road.

The traffic light system within this simulation has been implemented as a n-state, m-traffic light system. For each traffic light n sets of traffic light colours and filter values is required. Also required is a time allocation for each state. Each traffic light is also connected to a waypoint. A driver is able to query the state of the traffic light connected to the waypoint he is heading to. This allows the driver to make the required decision to proceed or wait.

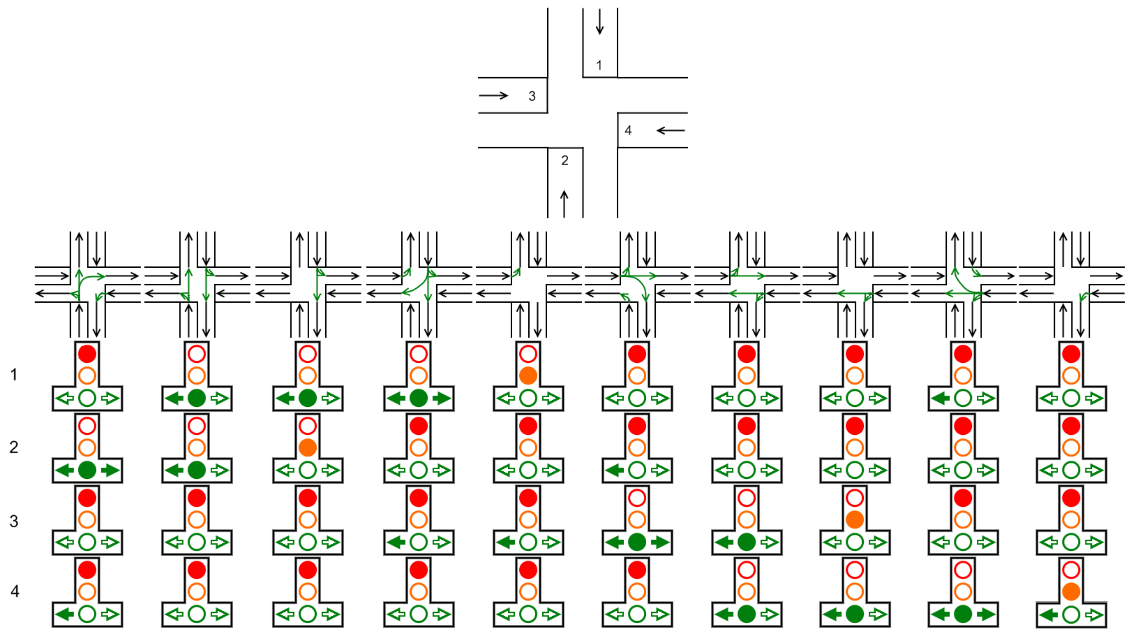


Figure 4.3: States required for a crossroads system of traffic lights

Initialization

For each traffic light network a complete list of states and timings for each light is required. Initially this was hard coded into the system but this is not an ideal situation when switching around maps. Therefore similar to the waypoint and roads initialization a text file is loaded at start up. This text file contains all the information required for all the traffic light networks in the simulation. Again to change how the traffic lights work only requires changing the raw data in a text file. This means multiple traffic light systems can be developed and switched around and tested in the system with relative ease.

Update Loop

On each update loop the traffic light network is checked to see if the current state is out of time. When the traffic light network state is out of time it is switched to the next traffic light state and the time countdown is restarted, a process which is continuously repeated until the simulation is ended.

Graphic Representation

The current graphic representation of the traffic light is a bit simple. If the traffic light is red at a waypoint a white ball is placed above the waypoint position. If the traffic light is amber or green the ball is not shown. This is not ideal as it does not show the filter lights so it sometimes appears that a vehicle is breaking a red light when it actually has a filter light.

4.3 Vehicle

As noted in Chapter 2 a number of microscopic traffic simulation systems have been developed. Each of these systems have some sort of vehicle agent that navigates around the simulation. One of the unique aspects of this implementation is that the movement of the vehicle is determined by a physics engine's reaction to driver input rather than the driver having direct control of the vehicle's speed and acceleration.

Therefore, an important part of this model is to have a physically correct vehicle. The vehicle should only be able to do what a real vehicle can do. Also the vehicle controls should be limited to modes of control only available to real vehicle drivers, for example, engine force, braking force and a steering value. To fulfil these requirements the Bullet Physics [1] raycast vehicle (btRaycastVehicle) was used.

4.3.1 Initialization

As with the waypoints, roads and traffic lights, a text file is used to specify the initial details of all the vehicles that are generated in the system. Some of this information is for the driver and will be explained later. For each vehicle specified in the text file a new instance of the bullet raycast vehicle is created.

The raycast vehicle requires a large number of settings to create each new vehicle. The following is a short list of some of the settings:

[1] Bullet Physics is an open source physics engine see "www.bulletphysics.com"

- Suspension Stiffness
- Suspension Compression
- Suspension Damping
- Max Suspension Travel Cam
- Friction Slip
- Position
- Orientation

The values for suspension stiffness, suspension compression, suspension damping, max suspension travel cam and friction slip and others are currently kept constant across all the vehicles. The values these variables are set to are the original default settings used in the example bullet physics system that displays the vehicle. These values are suspension stiffness = 5.88, suspension compression= 0.83, suspension damping= 0.88, max suspension travel cam= 500.0 and friction slip = 10.5. The input from the text file specifies the initial position and orientation for each vehicle.

4.3.2 Update

On each update loop an engine force, brake force and steering value are set for each vehicle and are then handed over to the Bullet Physics engine. The physics engine takes these values along with the amount of time that has passed since the last update loop and calculates a new position for each vehicle. The physics engine takes into account the friction the vehicle experiences, the momentum the vehicle has picked up along with a number of other variables to create as realistic movement as possible.

While this creates a very realistic simulation a number of problems can occur. If too much brake force is applied the vehicle will lurch forward and the vehicle will bounce on its front wheels. If too much engine force is applied while turning the vehicle, the vehicle will roll over leaving the vehicle on its side.

4.3.3 Representation

Figure 4.4 shows the current visual representation of the vehicle. Currently all the vehicles within the simulation look identical, although with additional car models this would be very easy to change.

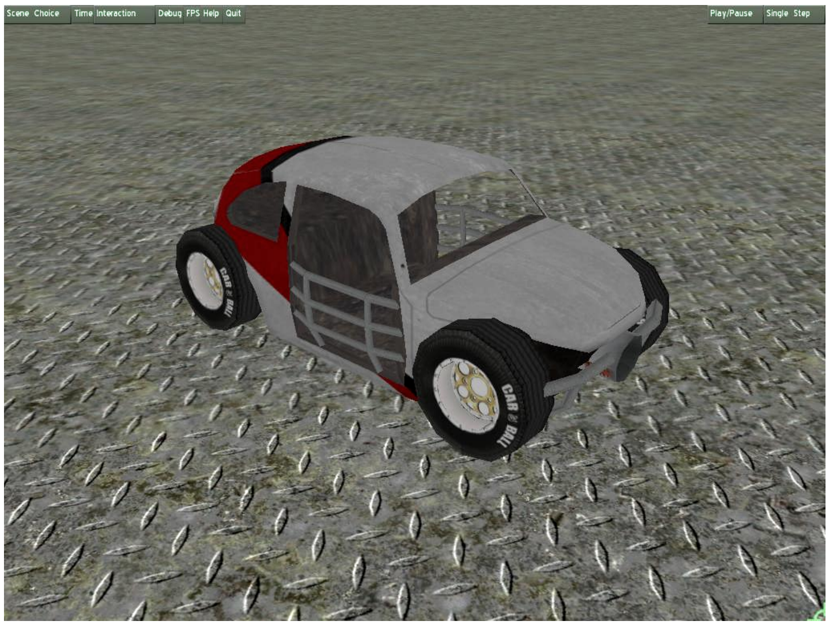


Figure 4.4: Model used to display vehicles

4.4 Control System

The purpose of the control system is to bridge the gap between the driver and the vehicle. As discussed above the Bullet raycast vehicle requires three values, an engine force, a brake force and a steering value. Unfortunately a driving agent does not care about the engine force but with the speed of the car and not the angle of the wheels but the direction that the vehicle is heading in. The main point of the control system is to convert the goal speed and direction that the driver will set and to manipulate the engine force, brake force and steering value to achieve this goal.

Currently the control system is broken up into two sections, one section is concerned with the speed at which the vehicle is moving and the other section is concerned with the direction in which the vehicle is pointing. The current system is very simple, and at the moment there is no interaction between these two sections. This may present problems in the future such as if the vehicle is reversing and is trying to change direction. Any steering that is done will have the opposite effect than is intended.

4.4.1 Speed

The speed section of the control system has a goal speed that it wishes to achieve and it also has the speed at which the vehicle is currently travelling. The objective of this system is to calculate an engine force or brake force at each time step that will result in the vehicle quickly accelerating or decelerating to the desired speed. Once the system obtains that speed it must then maintain that speed.

There have been two attempts to create a system to resolve this problem. Both are outlined below. The first utilizes a simple linear relationship which was found between the speed and the engine force. The second tries to implement a proportional-integral-derivative (PID) controller, unfortunately the PID controller did not work as expected. The first system works and is used in the current implementation of the overall system.

Linear Relationship

After a number of tests, there was found to be a linear relationship between the speed of the vehicle and the engine force applied to that vehicle. Basically if an engine force is applied to a vehicle that vehicle will reach a constant speed after a period of time. This constant speed is a result of the engine force achieving the critical speed of the vehicle that cancels out the friction and resistance the vehicle is experiencing. A simple multiplication of the goal speed by a constant speed-engine force ratio will give the required engine force that over a period of time will result in that speed.

This resulted in the vehicle eventually reaching its required speed but it took a very long period of time to do so. To solve this problem a very quick and simple solution was applied. By doubling the engine force the vehicle would accelerate much faster thus arriving at its goal speed quicker. Once the goal speed was achieved the engine force would be reset to the original engine force thus maintaining its goal speed.

Another problem with this system was that it did not slow down fast enough. Although generally the only time this was a problem was when the vehicle was required to stop, that is, when the goal speed was zero. The solution then used to quickly achieve this goal speed of zero was done by applying a large braking force. This resulted in a very crude jerky stopping motion where the vehicle would lurch forward and bounce on its front wheels. Overall this is not very realistic in terms of what a driver would do but is a working solution.

PID Controller

To improve the smoothness of the speeding up and slowing down an attempt was made to use a PID control system to modulate the engine force to achieve the correct speed and also a suitable acceleration and deceleration. A PID control system is a proportional-integral-derivative controller which is a generic control loop feedback mechanism widely used in industrial control systems. As the name suggests the controller involves three separate parameters each affecting the system in a different way:

- The proportional value determines the reaction to the current error
- The integral determines the reaction based on the sum of recent errors
- The derivative determines the reaction to the rate at which the error has been changing

An attempt was made to implement and integrate the controller into the system. Unfortunately the PID controller system was a complete failure. The vehicles would start off slowly accelerating but would then go too fast, then they would start reversing for an unknown reason. After a while the vehicles would realise they were going in the wrong direction and would then hugely accelerate forward causing virtually all the vehicles to crash in a very amusing way. Due to time constraints there was not enough time to work out why the system failed or how to get the system to work.

4.4.2 Direction

There is a goal direction/vector to which the driver wants the vehicle to be pointing and an actual direction/vector in which the vehicle is pointing. The requirement of this control system is to calculate a steering value which over time will result in these two vectors pointing in the same direction.

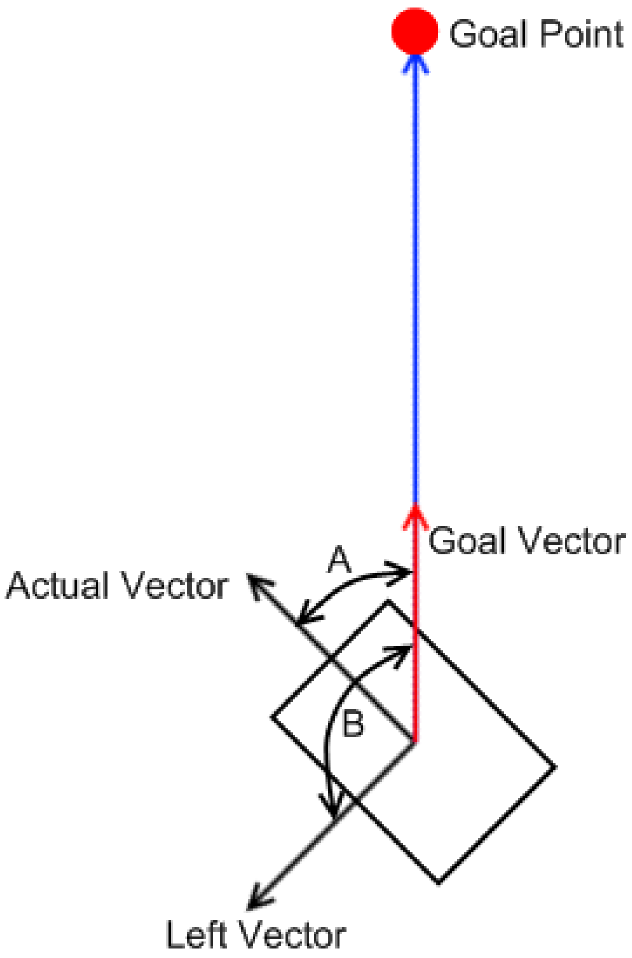


Figure 4.5: Direction component of the control system

Figure 4.5 gives a graphical representation, showing the goal vector and actual vector of the vehicle. Angle A can be calculated by finding the inverse cosine of the the dot product of the goal vector and the actual vector [2]. This angle A is actually an angle of error and it is used to specify the magnitude of the steering value. For example the bigger the angle between these two vectors the bigger the steering value being applied will be.

There is a major problem with this solution so far. The angle of error has no direction associated with it. It could be A degrees left or A degrees right. The dot product calculation will give the same solution either way. To solve this problem another vector is required. This vector is the left vector of the vehicle. The left vector is a vector which is rotated exactly $90\,^\circ$ left of the actual vector of the vehicle. A similar calculation can be done as before to calculate the angle B. If the angle B is more than $90\,^\circ$ then the required direction of steering value is right, if it is less than $90\,^\circ$ then the direction is left.

4.5 Driver

The driver component of the system is where the actual decisions are made. This component is the 'Action Selection' i.e. 'What to do now?' At a most basic level the driver perceives the world around him and from those perceptions makes a decision to head in a certain direction and go at a certain speed.

An integral part of the driver system is his ability to 'see' other vehicles around him. This ability was developed and implemented in two ways, the first using a series of cascading classifiers and the second using a spatial partitioning technique. The first proved to be exponentially expensive when the number of vehicles was increased and so was abandoned. The second system was adopted. This solution proved very useful and was found to be linearly proportional to the number of vehicles introduced into the simulation system.

[2]This method only works if both the vectors are normalised, which in this case they are.

Currently the goal of the driver agent is to following a predetermined circular path around the road network. While achieving this goal the driver must follow the rules of the road. This means that the driver must drive on the left-hand side of the road and must give right-of-way to other vehicles at certain junctions. The driver must also obey any traffic light signals and only proceed when he has a green light or green filter light. If the driver ends up behind another vehicle travelling at a slower speed he must slow down and avoid hitting the vehicle in front.

Intuitively this list of requirements can be broken down into a number of states. Each state handles a certain type of traffic situation:

- Drive along a road
- Navigate junction
- Change to new road
- Obey traffic lights
- Follow vehicle

Logically this can be mapped to a finite state machine (FSM). Overall this allows the driver to be very flexible. If the system needs to handle a new traffic situation all that is needed is to add an additional state to the FSM. After a number of iterations the diagram shown in figure 4.6 is the current implementation of the FSM. Each state is described in detail in a number of sections later in this chapter.

4.5.1 Initialization

As described in section 4.3.1 there is an input file that supplies all the data for the vehicles/drivers in the current simulation. For each entry in the input file a driver agent is created and attached to the vehicle agent and control system already created. The input file specifies the circular route which the driver is meant to travel around the road network. The file also specifies at what point along the route the driver is currently at, along with a number of other variables.

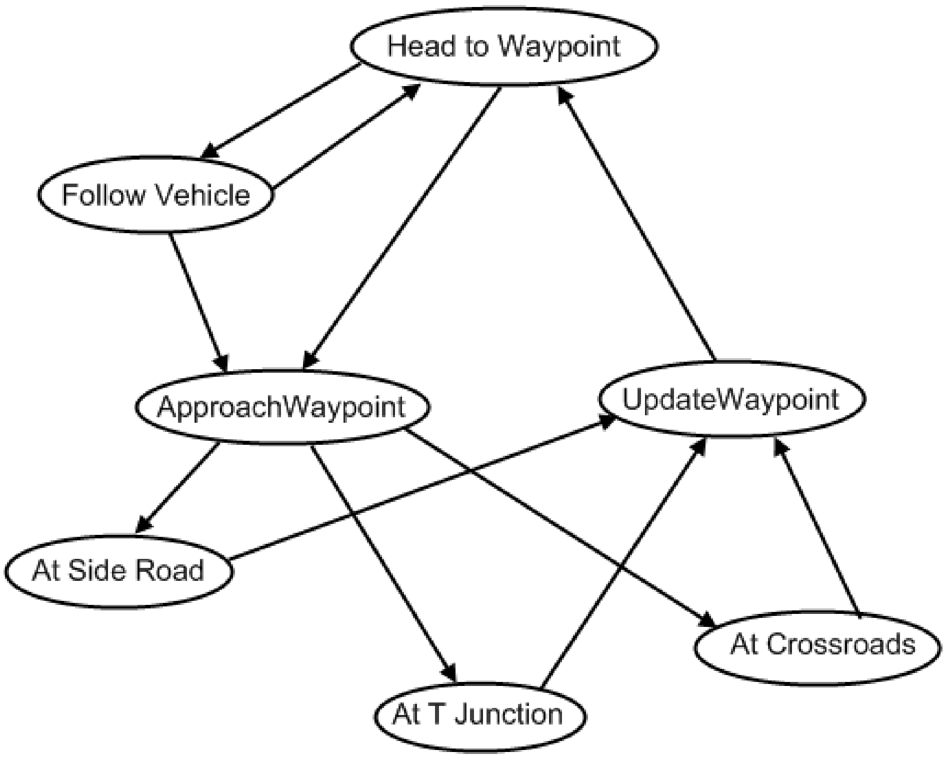


Figure 4.6: Finite state machine behind the driver

As the driver is being created a number of variables are calculated and set. These variables include: the road the driver is currently on, the waypoint the driver is heading towards, the waypoint the driver will head towards after that. The vehicle will also be registered to the road it is on.

4.5.2 Tools

To allow the driver to understand what he is doing and what is in the world around him a number of functions were developed. A simple concept of moving left, right or straight ahead is required in an abstract world of waypoints. A set of functions were developed to give the distance of the nearest approaching vehicle in a certain direction. In addition a function was developed to determine if there is a vehicle ahead of the driver and how close the driver is to that vehicle. These functions are all described in the following sections.

Heading

This function calculates whether moving from one waypoint to another is a left turn, right turn or straight ahead. This function creates two vectors a past vector and a future vector. The past vector is the vector from the last waypoint to the current waypoint. The future vector is the vector from the current waypoint to the future waypoint. As explained in section 4.4.2 the angle and direction of two vectors can be found with a number of dot product calculations.

Approaching Vehicle Distance

The point of this function is to return the distance between the driver and the nearest vehicle approaching the driver from a particular heading. An example situation when this function is required is when a driver arrives at a T-Junction and wishes to turn left. The situation is illustrated in figure 4.7. Because the driver is joining a road he must give right of way to vehicles already on the road. The driver must make sure there is enough of a gap in vehicles approaching from the right for him to be able to make the turn and accelerate without disrupting the main flow of vehicles on that main road. Therefore, the distance between the driver and the nearest vehicle approaching from the right is required. Once this value is great enough the driver can pull out and make the turn.

This function was implemented in two different ways first without the road spatial partitioning technique and then a second time with the road spatial partitioning technique.

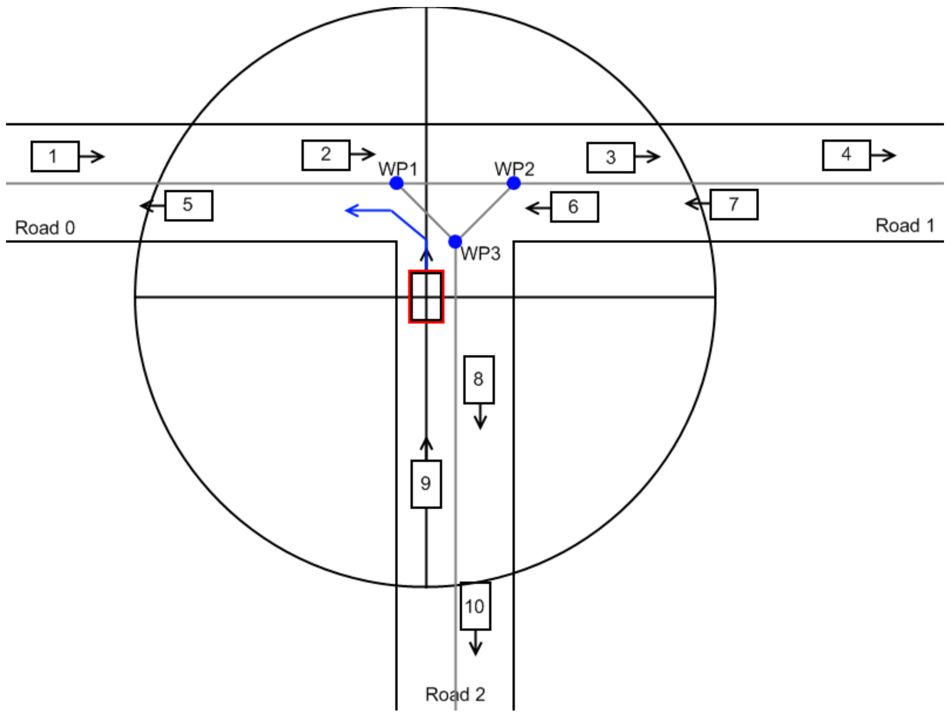


Figure 4.7: Example of old space querying system

The first way this function was implemented was by a series of cascading classifiers. The function would start off with global knowledge of all the vehicles in the system and then would start discounting vehicles until it was reduced down to only a few which would hopefully be the vehicles approaching from the direction in question. The first is a simple squared distance test. The squared distance is very quick and easy to find compared to the actual distance because it does not require the expensive square root operation. Using the diagram 4.7 this test reduces the vehicles down to only the vehicles in the radius i.e. vehicles 2, 3, 5, 6, 8 and 9. In this example the idea is that the driver wants to turn left unto road 0, but to do this he first has to check to see if there are any vehicles on road 1 and if there is, is the distance to the first vehicle large enough to allow the driver to pull out and get going on road 0. The next classifier is done by calculating the normalised vector to each of the other vehicles by subtracting the positions of the driver from the position of the other drivers and then normalising that vector. Using the dot product of the direction of the driver and this new vector gives a value. If that value is greater than zero then the other vehicle is in front of the driver. If the value is negative then the vehicle is behind the driver. Using this vehicles 8 and 9 can be removed from consideration. By rotating the direction of the vehicle around 90 degrees to the right and calculating the dot product of this new right direction with the vector to the other vehicles, the vehicles which are on the left and which are on the right can be found. This reduces the list of vehicles to 3 and 6. Now vehicles heading left relative to the direction of the vehicle are required. By creating a left direction vector and calculating the dot product of this with the direction the other vehicles are travelling in, reveals that 6 is the only vehicle needing consideration. Now the distance between the driver and all the vehicles left in the list can be calculated and the function can return the shortest distance.

The second way this function was implemented was by utilising the road spatial partitioning technique. The function queries what road the vehicle is on and what waypoint it is heading to. In the diagram 4.7 this returns Road 2 and Waypoint 3. Again it is necessary to get a list of vehicles approaching from the right so a decision can be made if it is safe to turn left. The function then queries road 2 for a list of roads connected to waypoint 3 in a right turn. This returns road 1 via waypoint 2. Road 1 is then queried for a list of vehicles on road 1 heading to waypoint 2. This returns vehicles 6 and 7. Knowing the future information of where the vehicle will head to next is like looking at the indicator of the vehicle. If the vehicle is heading to waypoint 3 after waypoint 2 then it is not counted. Finally a list of vehicles on road 1 heading to road 0 remain. Again a small number of distance tests can be done and the shortest distance is returned to the driver to determine whether he has room to proceed or if he needs to wait.

Follow Vehicle Distance

The reason for this function is to determine what vehicle, if any, are on the current road, heading in the same direction and in front of the driver. The function was developed and implemented in a similar way to the Approaching Vehicle Distance function in the section before. Again a series of cascading classifiers was used but like the above implementation proved too slow for practical use.

With the spatial partitioning technique the situation is very easy. The driver queries the road he is on for a list of vehicles on the road that are heading to the same waypoint the driver is heading to. By calculating a vector to the other vehicles and performing a dot product calculation with the direction of the vehicle, it can easily be determined which vehicles are in front of the driver and which are behind. The function calculates the distance to all the vehicles in front of the driver and returns the distance to the nearest one.

4.5.3 Finite State Machine

Buckland [21] in chapter 2 presents a template for a comprehensive Finite State Machine (FSM). The FSM presented was fairly intuitive and flexible, appeared quick and easy to code and debug, and had very little computational overhead. For those reasons Buckland's template was used to create the FSM for the driver. The following sections describe all the states, the logic that is performed in those states and the switching conditions for changing states.

The general order of states for the FSM is: *head to waypoint*, *approach waypoint*, *a junction state*, *update waypoint*, back to *head to waypoint*. The junction state is a set of conditions that state whether a driver can proceed or should stop and wait at a given junction when the driver wants to preform a certain junction crossing. Once the condition to proceed is satisfied then the system updates the waypoint and continues.

Head To Waypoint

Head to waypoint is the initial state for all the drivers. Basically the state is just for driving down a road when no one is in front of the driver. The goal speed of the vehicle is set to the speed of the road. A goal vector is calculated by subtracting the current position from the waypoint position. This goal vector is then also set. On every loop the distance to the waypoint is calculated. Also the follow vehicle distance function is used to see if any vehicles are in front of the driver and if so how close.

There are currently two exit conditions for this state. If the driver gets too close to the vehicle in front the state is switched to *follow vehicle*. The other condition is when the driver gets close to the goal waypoint, the state is then switched to *approach waypoint*.

Follow Vehicle

When the driver is in this state it is because he got too close to the vehicle in front of him. The general reason for this happening is when the vehicle in front of the driver is travelling slower. To avoid hitting the vehicle in front the driver must slow down. The goal vector is left alone but the goal speed is set to match the vehicle in front. Again the distance to the waypoint and the distance to the vehicle in front are continuously measured.

There are currently two exit conditions for this state. If the distance to the vehicle in front gets larger again, then the state is switched back to the *head to waypoint* state. The other condition is when the driver gets close to the goal waypoint, the state is then switched to *approach waypoint*.

Approach Waypoint

Once a driver approaches a waypoint he is transferred to this state. This state looks at what type of waypoint the driver has approached. The state is then immediately switched to a state which deals with that type of waypoint. For example if the waypoint is a side road waypoint then the state is switched to a *at side road* state.

To expand the capacities of the road network to include other road junctions, this is the only state that would need to be modified. A new state would be created to handle that new junction, for example a roundabout. This state would just need an extra condition to check to see if the waypoint was a roundabout waypoint and if it was, then switch to that new state that deals with a roundabout.

At Side Road

There are two options a driver has when he approaches a side road. He can continue on past the side road which requires no checking for any other vehicles. The other option is for him to turn onto the side road. This can happen in one of two ways depending on what side of the road the side road is on. If it is a left turn then the driver does not need to give right of way to anyone else. However if the turn is a right one then he must give right of way to vehicles approaching from straight ahead, as the driver will need to cross that lane to complete the right turn. If the driver is not required to check for vehicles or if the way is clear then the state is immediately switched to *update waypoint* state. Figure 4.8 shows what directions a driver has to give way to at a side road or t-junction.

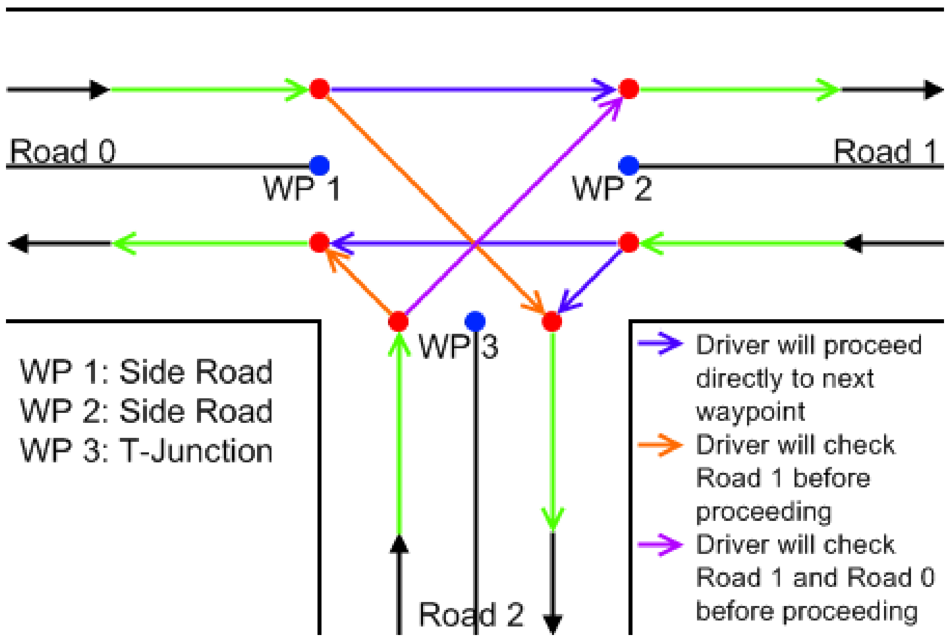

Road 0
Road 1
WP 1
WP 2
WP 3
Road 2
WP 1: Side Road
WP 2: Side Road
WP 3: T-Junction
Driver will proceed directly to next waypoint
Driver will check Road 1 before proceeding
Driver will check Road 1 and Road 0 before proceeding

Figure 4.8: Right–of–way diagram for T–Junction

At T Junction

When a driver is at a T Junction he has the option to turn left or right. If the driver turns left he first checks to see if there are any vehicles coming from the right and going straight. If the approaching vehicle is turning left it is discounted. If the driver wants to turn right he has to check for vehicles approaching from both left and right. Once either of these conditions is meet the state is switched to the *update waypoint* state.

At Crossroads

A number of options were considered as to how to deal with a crossroads.

- Each direction waits for other directions.
- Nearest vehicle to the centre of the crossroads goes first.
- A priority queue.
- Traffic Lights.

The first situation was that if a driver wanted to turn left at a crossroads he had to check for vehicles coming from the right. If the driver wanted to go straight on he had to check for vehicles coming from right and left, and if the driver wanted to turn right he had to check for vehicles coming from left, right and straight ahead. This did not work very well as vehicles ended up either crashing, or waiting for an indefinable period for another vehicle which was actually waiting on them.

The second attempt had a calculation that the nearest vehicle to the centre of the crossroads went first. This time no matter what direction the driver wanted to go the driver checked to see if there were any vehicles approaching from the left right and straight ahead. If there was the other vehicles position relative to the centre of the crossroads was compared to the drivers and if there was no other vehicles closer to the crossroads then the driver proceeded. This was meant to create a first come first served situation but there were two problems. When two vehicles were the same distance to the centre of the crossroads, that is neither was closer, then both drivers would go and most likely crash. When the logic was reversed and the check was that the driver had to be closer to the crossroads neither driver would go and it would be gridlock. The second was if a driver was following another vehicle and that vehicle got through by momentum, this meant that a string of vehicles would get through from one direction with unfair advantage.

A third option was considered but never implemented. There would be some sort of queue placed at each crossroads. As a driver approached the crossroads he would add himself to the queue. He would then see if he was at the front of the queue. If he was he would move through the crossroads and once through would remove himself from the queue allowing the next driver through. If the driver was not at the front of the queue he would wait until he was. In theory this method should work quite well with no potential for crashing, but might not have been the most efficient in terms of traffic flow for high demand crossroads.

The forth and final attempt which was actually implemented in the final version of the system was the traffic light system. If a driver got a red or amber light he would stop and wait. Once a green light was on the driver could proceed left or straight. If the driver wished to turn right he had to check for a right filter light or that no cars were approaching from ahead. On the whole this works very well, but required some tuning to make sure that the roads with the highest demand got more time on the traffic lights than the other roads, otherwise that road would clog up.

Update Waypoint

In this state the waypoint information is updated. This requires some calculation. The current driver route is queried to find out what the new waypoint is, but this is not where the driver actually needs to head. The driver needs to head to the left of the waypoint so that the driver drives on the left side of the road. Once the calculation has been complete the state is switched back to *head to waypoint* state.

4.5.4 Behaviour

At the beginning of this project, this was an area where there was a lot of potential, unfortunately time limitations meant that not much work was done on it.

Currently there are two areas in the driver model which can be altered to give variations in the drivers behaviour. The first area is in speed. Each driver has a multiplier which is multiplied by the speed limit of the road to determine the speed the driver prefers to travel at if he can. The second is the distance at which the driver likes to leave between the vehicle in front and himself.

Chapter 5

Evaluation and Discussion

This chapter outlines how the system performed when used. As explained in chapter 4, a number of input files are required to initialize the simulation and these files can be altered to change the simulation. For the purposes of evaluating the system three sets of input files were created called maps, effectively road networks. Tests were then performed on the maps to analyse the long and short term performance. To get a feel for the breakdown of the performance of each section of the simulation a number of scenarios were run which tested different aspects of the system. Overall the system simulated the desired map in the expected way. Over time certain traffic phenomenon did start to appear such as traffic jams. These traffic jams could then be altered by attempting different configurations of traffic light times. The ability to immediately see the results of such an alteration allowed for a very quick iterative process to determine the best traffic light time solution.

5.1 The Simulation System

Once a set of input files are created and the simulation system is started, the system initializes according to the input data and displays the world. The world is composed of a flat plane that represents the ground, a number of boxes that visually outline the roads, a few floating white balls that represent red traffic lights and a number of car models that represent the vehicles. The user is able to move the camera around and study how the vehicles are interacting with the environment. For example the user can

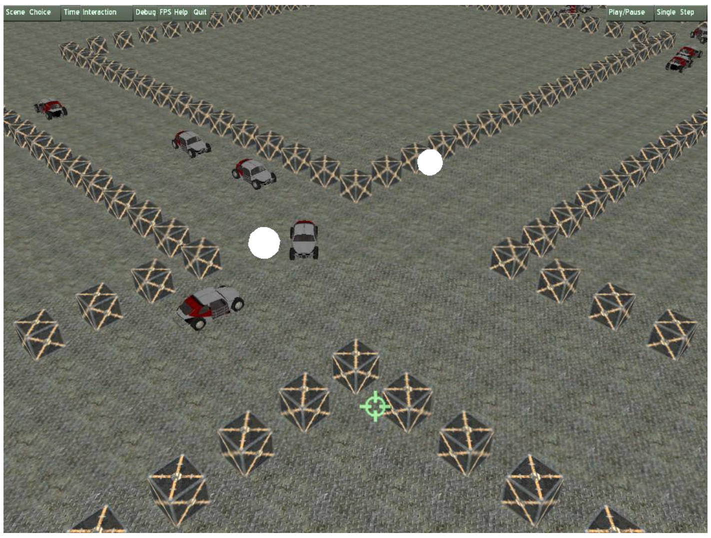


Figure 5.1: Screenshot A of a traffic situation at a crossroads

zoom into a crossroads as seen in figures 5.1, 5.2 and 5.3 and watch the interaction of the vehicles with the traffic lights. The user can move over to a t-junction and observe how the right–of–way operation is affecting the operation of the vehicle. This is shown in figures 5.4 and 5.5. It is also possible to latch onto a vehicle and just follow that vehicle around and see how it reacts to the world around it, as illustrated in figures 5.6 and 5.7. Another option is to just watch the system from above looking down on the entire map as seen in figures 5.9, 5.11 and 5.13.

5.2 Maps

To confirm the flexible nature of the system three different maps were created. The first map as seen in figure 5.8 is a road network with 12 roads in the shape of a standard New York city block. Each of the t-junctions do not have any traffic lights and are navigated by right-of-way. The crossroads in the centre has a traffic light system which currently gives equal time to each of the roads. This map has been tested with up to 28 vehicles and has been left running for over 5 hours with no memory leaks in the program or vehicles crashing in the simulation. A screenshot of map 1 is shown in figure 5.9.

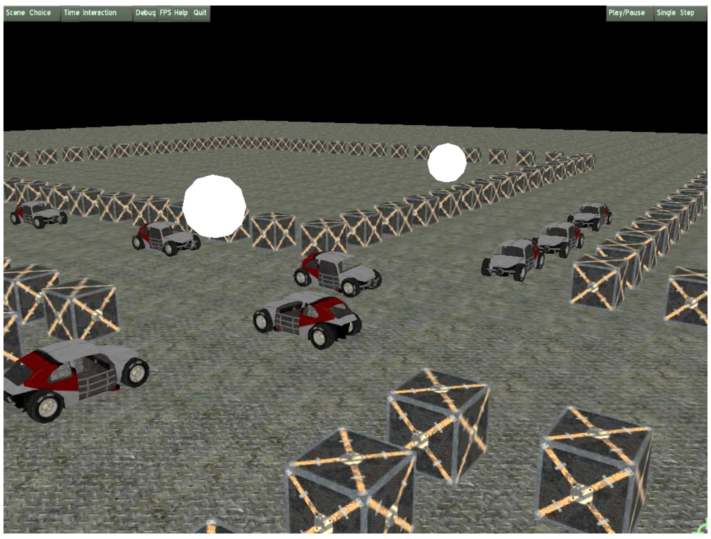


Figure 5.2: Screenshot B of a traffic situation at a crossroads

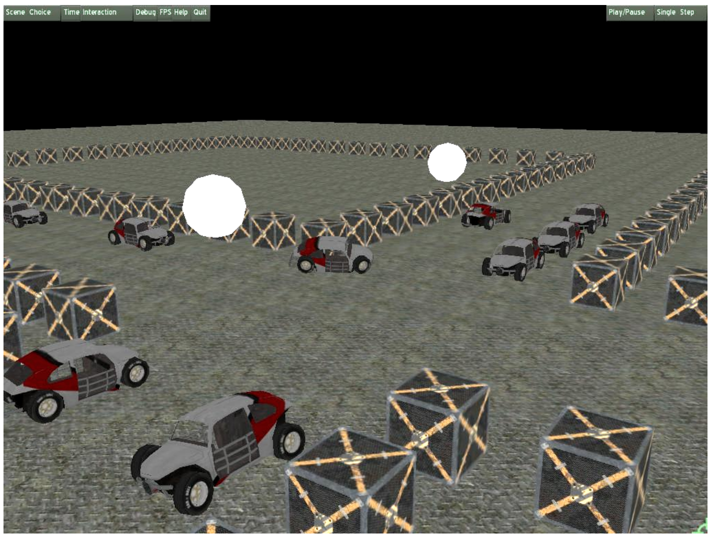
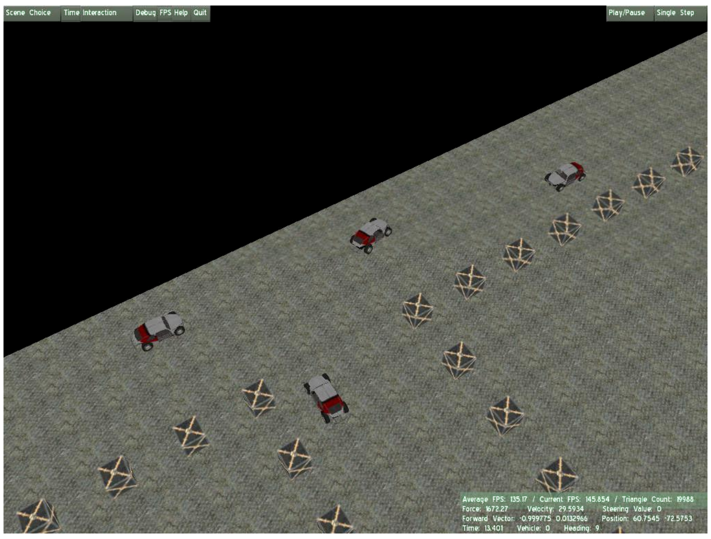
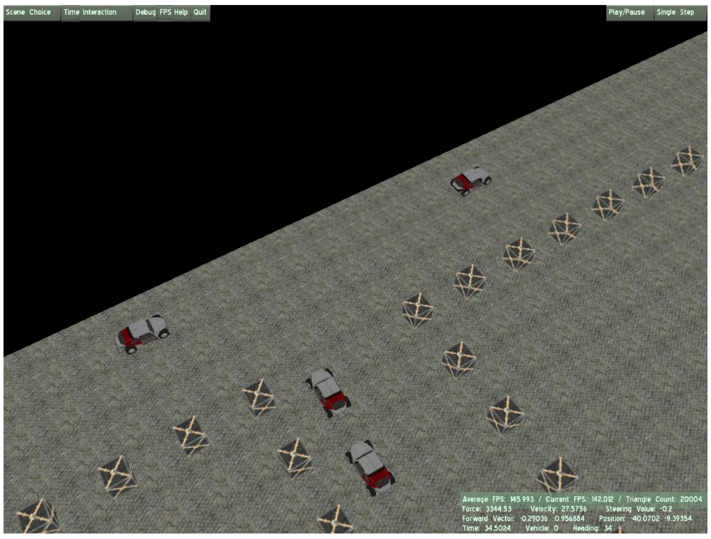


Figure 5.3: Screenshot C of a traffic situation at a crossroads



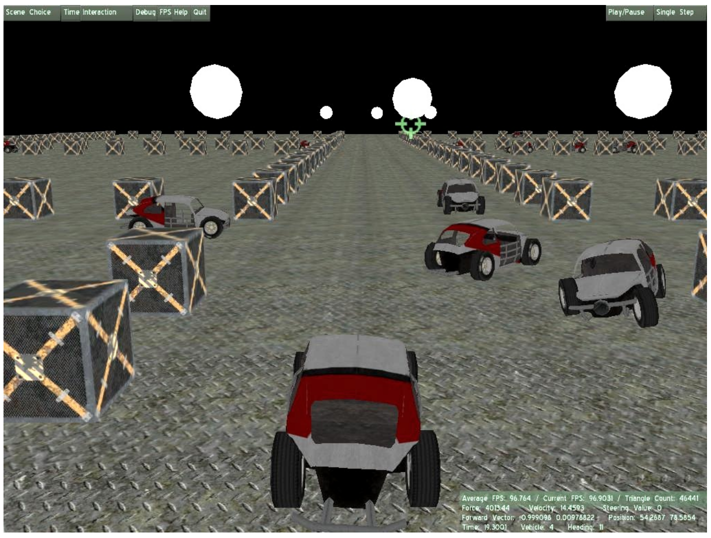
```
Average FPS: 135.17 / Current FPS: 145.854 / Triangle Count: 19988
Force: 1672.27        Velocity: 29.5934        Steering Value: 0
Forward Vector: -0.999775 0.0132966     Position: 60.7545 -72.5753
Time: 13.401     Vehicle: 0     Heading: 9
```

Figure 5.4: Screenshot A of a traffic situation at a t–junction



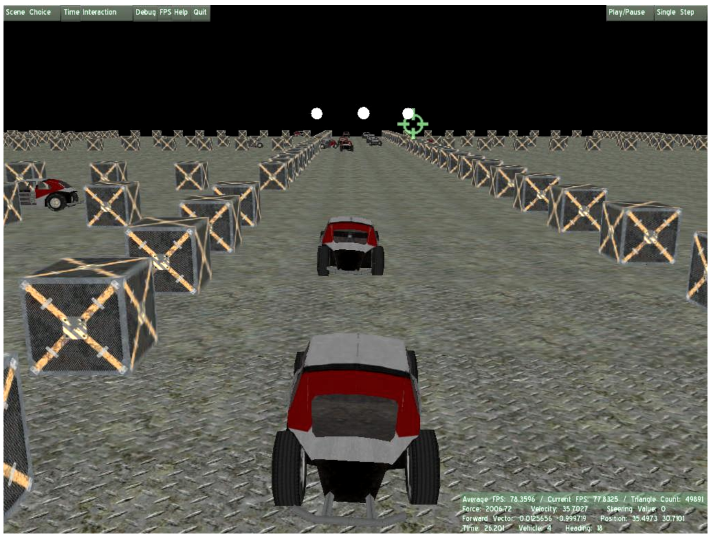
```
Average FPS: 145.993 / Current FPS: 142.012 / Triangle Count: 20004
Force: 3344.53        Velocity: 27.5756        Steering Value: -0.2
Forward Vector: -0.29036 -0.956884     Position: -40.0702 -9.39354
Time: 34.5024     Vehicle: 0     Heading: 34
```

Figure 5.5: Screenshot B of a traffic situation at a t–junction



```
Average FPS: 96.764 / Current FPS: 96.9031 / Triangle Count: 46441
Force: 4013.44      Velocity: 14.4593      Steering Value: 0
Forward Vector: -0.999098 0.00978822      Position: 54.2687 78.5854
Time: 19.3001     Vehicle: 4     Heading: 11
```

Figure 5.6: Screenshot A created while following a vehicle



```
Average FPS: 78.3596 / Current FPS: 77.8325 / Triangle Count: 49891
Force: 2006.72      Velocity: 35.7027      Steering Value: 0
Forward Vector: 0.0125656 -0.999719      Position: 35.4973 30.7101
Time: 26.201     Vehicle: 4     Heading: 18
```

Figure 5.7: Screenshot B created while following a vehicle

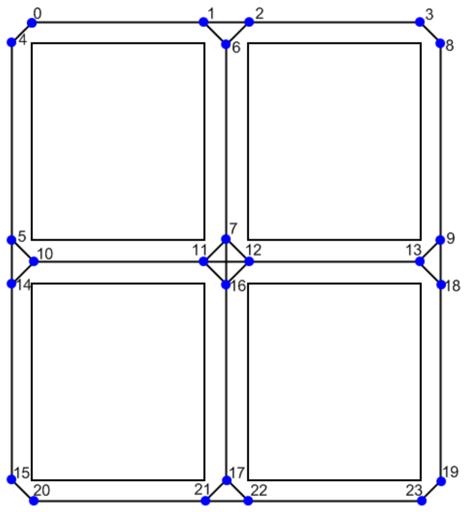


Figure 5.8: Map 1: 12 road map with 28 vehicles

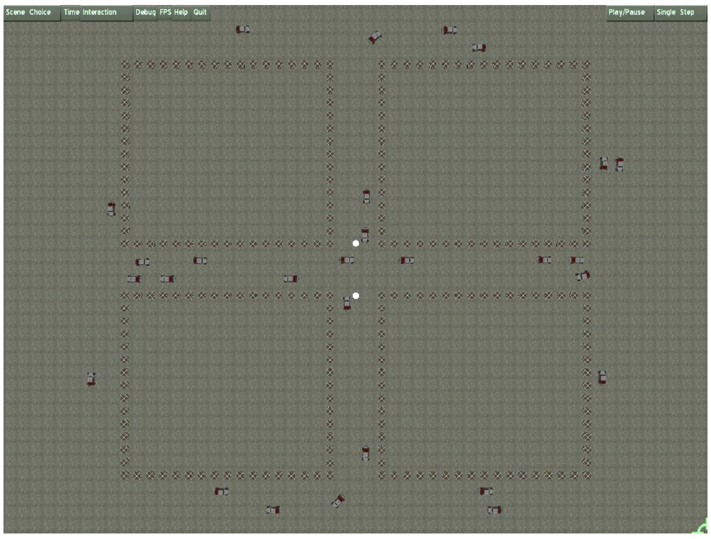


Figure 5.9: Screenshot of map 1

The second map is illustrated in figure 5.10 and visually shown in figure 5.11. The map has now been expanded to 27 roads. Again the t-junctions are operated by right-of-way and the four crossroads are operated by traffic lights. This time the traffic lights for each crossroads have been customised to suit the demand on each road to minimize traffic that was building up in the system. The map has been tested with up to 44 vehicles and again been left running for over 5 hours with no memory leaks in the program or vehicles crashing in the simulation.

The third map was an experiment to see how the system could handle diagonal roads. Figure 5.12 shows that the map is really just the first map with three extra roads added to the end. A screenshot is shown in figure 5.13. Again the t-junctions operate by right-of-way and the 2 crossroads by traffic lights. The map was tested with 28 vehicles. The diagonal roads did not adversely affect the system in any way and were handled like any other road with no problems.

5.3 Breakdown of Performance

To determine how each section of the system was performing the average loop time for a number of scenarios was recorded. The results are shown in table 5.1. Each scenario is tested on each of the maps to confirm the results. The first scenario is the standard simulation of the map with the camera looking down showing the entire map for each simulation. Scenario 2 is identical to scenario 1 except that the camera is pointed away from the simulation, such that nothing is seen on the screen. This illustrates that 78.9 %, 72.6 % and 80.2 % of the loop time of maps 1, 2 and 3 respectively is just rendering.

Scenario 3 again is identical to scenario 1 but this time the boxes which visually represent the outline of the roads are removed. These boxes are continuously checked by the Bullet physics engine to see if the vehicles have hit any of them and represent a large computational overhead in the physics engine. There are 240 boxes in map 1 representing 69.2 % of the loop time. 81.8 % of the loop time is spent on the 444 boxes in map 2. Map 3 has 312 boxes that takes up 76.9 % of the loop time. There must also be a substantial rendering overhead for these boxes.

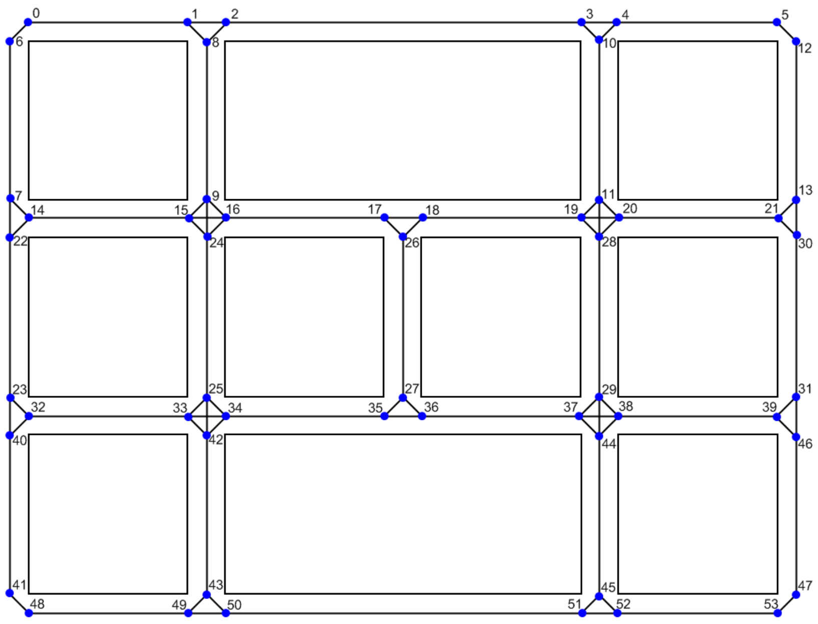


Figure 5.10: Map 2: 27 road map with 44 vehicles

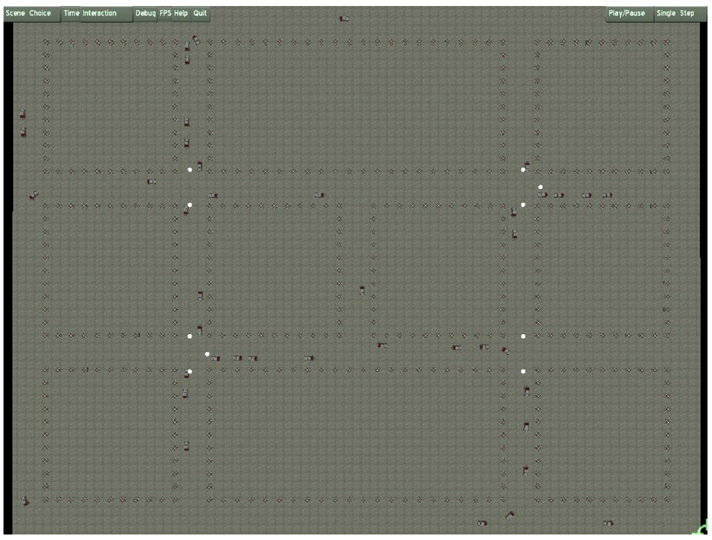


Figure 5.11: Screenshot of map 2

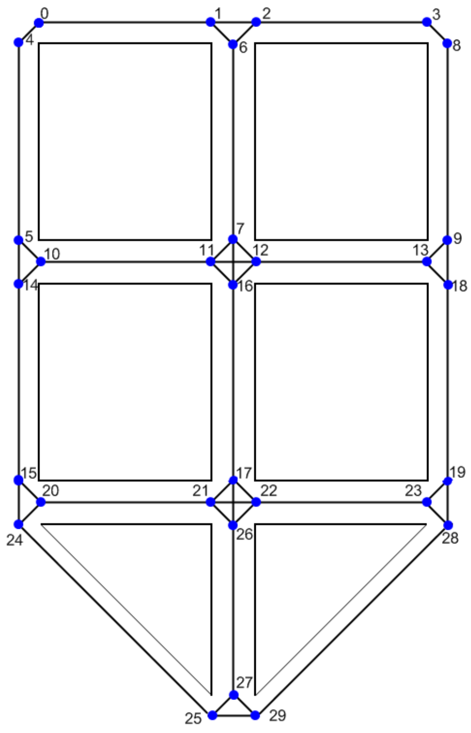


Figure 5.12: Map 3: 15 road map with 28 vehicles including diagonal roads

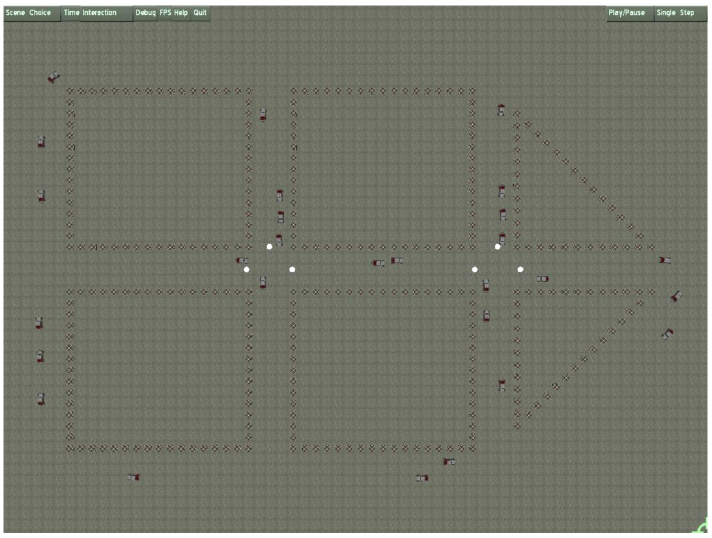


Figure 5.13: Screenshot of map 3

	Average Loop Time (ms)		
	Map 1	Map 2	Map 3
Scenario 1	6.485404	19.75432	8.589997
Scenario 2	1.368379	5.399069	1.705105
Scenario 3	1.997162	3.598933	1.986892

Table 5.1: Average loop time for a number of scenarios

No Of Vehicles	Average Loop Time (ms)
0	2.970919
1	3.191800
2	3.302952
4	3.497552
8	3.767038
12	4.296671
16	4.665826
20	5.296505
28	6.485404

Table 5.2: Relationship between number of vehicles and average loop time for map 1

Another performance factor of the system is how does the system scale with the number of vehicles. The first map was used to measure the average loop time for different numbers of vehicles ranging from 0 vehicle to 28 vehicles. The results of the measurements can be seen in table 5.2 and the results are graphed in figure 5.14. A basic conclusion that can be drawn from this information is that the system scales linearly with the number of vehicles within the system. On average each vehicle costs roughly 0.11 ms per loop. All the performance tests were done on an Intel Core 2 CPU 6400 @ 2.13GHz with 2 GB of RAM with a GeForce 7900 GTO graphics card.

5.4 Framework

Ultimately the original framework did evolve from the beginning of the project, although the overall structure of the framework remained the same. An improvement to this framework would be at the interface to the control system. The improvement would include a variable that would represent a time allocation in which the goal

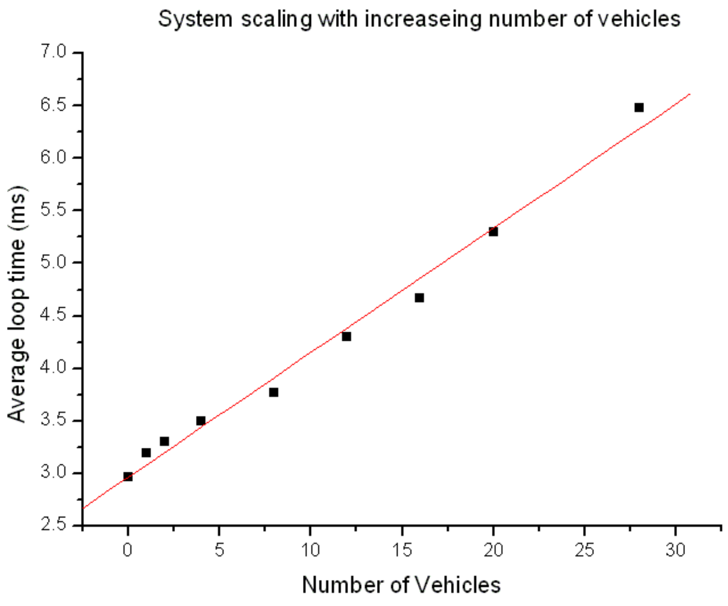


Figure 5.14: Average loop time vs number of vehicles

speed or vector of the vehicle should be reached. This would then affect how quickly the vehicle needed to respond to a change in goal speed or vector.

5.5 Validity of Simulation

An area which due to time constraints was not carried out was analysing the validity of the simulation. The original idea was to integrate the simulation system into the 'Virtual Dublin' project. This would allow the simulation to be compared to actual traffic flow occurring in the real Dublin city centre.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis outlines a flexible microscopic traffic simulation system. The system allows dynamic generation of an entire road network based on an input file. The system also allows dynamic generation of driver agents capable of navigating roads, t-junctions, crossroads and traffic lights, based on a second input file. The system uses a spatial partitioning technique using roads for quick reliable local space querying which allows the system to be linearly scaled with the number of vehicles. A traffic light network can be incorporated at any number of junctions within the system and easily changed to optimise traffic flow. The finite state machine within the driver agent allows for quick expansion of abilities of the driver to deal with new traffic situations. Overall the system is flexible enough to be easily adapted to any traffic simulation requirement.

The system presents a viable way to study emergent traffic flow levels and see how they would react to changes in the world. Adjustments to road layout can very quickly be made and the system run again to see how the changes made impact on the traffic flows. The ability to easily change the timings on the traffic lights or see what it would be like to add a traffic light to a location which didn't have one before enables investigation of traffic congestion areas. Overall this agent–based traffic simulation system creates an environment where emergent traffic phenomenon can be investigated.

6.2 Future Work

6.2.1 Route Planning

Currently the driver follows a pre-determined route which is circular. This means that all the vehicles will stay on the map and will perform certain manoeuvres. An original plan was to have an A* search algorithm to find a route for the driver to follow. This algorithm would take account of many pieces of information along with the static map of the world. These extra pieces of information would include past experiences on the roads travelled and maybe an AA roadwatch system that might report on areas of heavy traffic that certain drivers will be listening to and wish to avoid. These pieces of information could then be incorporated into the weighting of each road in the determination of the best route to take.

6.2.2 Global State and Messaging

The FSM sample given by Buckland [21] includes a global state and a messaging facility. It had been planned to use these facilities. The plan for the global state was that the driver would always be on the lookout for a hazard in front or around the car and if that were to occur the global state would hijack the current state to a state that would deal with that situation. An example would be if a pedestrian were to step out on to the road the driver would have to stop or try to swerve to avoid hitting the pedestrian.

Messaging between drivers or between drivers and traffic lights or pedestrians could be very useful in allowing communication of information between people. An example would be if a person wants to cross a road they could send out a message before stepping out on to the road and the driver can decide weather to slow down and let the person cross or continue on and ignore the person trying to cross the road.

These new features could be developed for future versions of the system.

6.2.3 Virtual Dublin

As mentioned before, it would be desirable to ingrate this system into the 'Virtual Dublin' project. The two main steps required to do this would be to develop a road network with all the waypoint information that matches Dublin city centre. The other main step required would be to change how the system renders the vehicles and the world around it, due to the fact that 'Virtual Dublin' has systems in place to do that. The traffic light representation would also have to be improved. An actual traffic light model could be used with the correct lights and arrows illuminating at the correct times.

6.2.4 Behaviour

To improve the realistic nature of the driver's behaviour, it would be desirable to expand and reframe the current behaviour aspects of the driver. The goal would be to add a large number of variations to the drivers abilities more then just speed and distance. Then by linking the variations together under concepts of aggression, patience, competency and urgency and maybe using fuzzy logic algorithms an even greater level of realism might be achieved.

6.2.5 Crashing

An interesting thing is the idea of vehicles crashing. The original plan was that if two reckless drivers were to approach a junction that there was a chance they could crash. When the implementation was started, when two drivers approached each other, they would crash because they were stupid and had no concept of other vehicles. So a lot of effort was placed into the system to make drivers intelligent so that vehicles would not crash. It would be interesting to introduce a random rogue not so intelligent driver into the system to see if and how often crashes would occur. Would this then make all the drivers look stupid or would it be an acceptable almost expected part of a real simulation!

Bibliography

[1] A. Chronopoulos and C. Johnston, "A real-time traffic simulation system," *Vehicular Technology, IEEE Transactions on*, vol. 47, pp. 321–331, Feb 1998.

[2] E. Chevallier and L. Leclercq, "A macroscopic theory for unsignalized intersections," *Transportation Research Part B: Methodological*, vol. 41, pp. 1139–1150, Dec 2007.

[3] M. Ben-Akiva, M. Bierlaire, H. Koutsopoulos, and R. Mishalani, "Real-time simulation of traffic demand-supply interactions within DynaMIT," in *Transportation and network analysis: current trends. Miscellenea in honor of Michael Florian*, pp. 19–36, Kluwer, 2002.

[4] M. Mahut, M. Florian, and N. Tremblay, "Application of a simulation-based dynamic traffic assignment model," *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on*, pp. 439–444, 2002.

[5] S. B. Yoginath and K. S. Perumalla, "Parallel vehicular traffic simulation using reverse computation-based optimistic execution," in *PADS '08: Proceedings of the 2008 22nd Workshop on Principles of Advanced and Distributed Simulation*, (Washington, DC, USA), pp. 33–42, IEEE Computer Society, 2008.

[6] W. Burghout, H. Koutsopoulos, and I. Andreasson, "A discrete-event mesoscopic traffic simulation model for hybrid traffic simulation," *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, pp. 1102–1107, 2006.

[7] J. Cremer, J. Kearney, and Y. Papelis, "HCSM: a framework for behavior and scenario control in virtual environments," *ACM Trans. Model. Comput. Simul.*, vol. 5, no. 3, pp. 242–267, 1995.

[8] M. E. Ben-Akiva, H. N. Koutsopoulos, R. G. Mishalani, and Q. Yang, “Simulation laboratory for evaluating dynamic traffic management systems,” *Journal of Transportation Engineering*, vol. 123, no. 4, pp. 283–289, 1997.

[9] P. Ehlert and L. Rothkrantz, “Microscopic traffic simulation with reactive driving agents,” *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pp. 860–865, 2001.

[10] T. Al-Shihabi and R. R. Mourant, “A framework for modeling human-like driving behaviors for autonomous vehicles in driving simulators,” in *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, (New York, NY, USA), pp. 286–291, ACM, 2001.

[11] H. Wang, J. Kearney, J. Cremer, and P. Willemsen, “Steering autonomous driving agents through intersections in virtual urban environments,” *International Conference on Modeling, Simulation and Visualization Methods, 2004*, 2004.

[12] F. Zhang, J. Li, and Q. Zhao, “Single-lane traffic simulation with multi-agent system,” *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pp. 56–60, Sept. 2005.

[13] A.-E.-K. Sahraoui and R. Jayakrishnan, “Microscopic-macroscopic models systems integration: A simulation case study for atmis,” *Simulation*, vol. 81, no. 5, pp. 353–363, 2005.

[14] J. Wittmann, J. Göbel, D. Möller, and B. Schroer, “Refinement of the virtual intermodal transportation system (VITS) and adoption for metropolitan area traffic simulation,” in *SCSC: Proceedings of the 2007 summer computer simulation conference*, (San Diego, CA, USA), pp. 1–5, Society for Computer Simulation International, 2007.

[15] W. Burghout and J. Wahlstedt, “Hybrid traffic simulation with adaptive signal control,” *Transportation Research Record: Journal of the Transportation Research Board*, pp. 191–197, 2007.

[16] M. Jin and S.-H. Lam, “A virtual-reality based integrated driving-traffic simulation system to study the impacts of intelligent transportation system (ITS),”

Cyberworlds, 2003. Proceedings. 2003 International Conference on, pp. 158–165, Dec. 2003.

[17] S. Ono, K. Ogawara, M. Kagesawa, H. Kawasaki, M. Onuki, K. Honda, and K. Ikeuchi, "Driving view simulation synthesizing virtual geometry and real images in an experimental mixed-reality traffic space," *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, pp. 214–215, Oct. 2005.

[18] A. Nanyue, B. Hanwu, C. Yan, and D. Yongming, "Modeling of virtual traffic environment for driving simulator," *Computer-Aided Industrial Design and Conceptual Design, 2006. CAIDCD '06. 7th International Conference on*, pp. 1–5, Nov. 2006.

[19] K. N. Hewage and J. Y. Ruwanpura, "Optimization of traffic signal light timing using simulation," in *WSC '04: Proceedings of the 36th conference on Winter simulation*, pp. 1428–1436, Winter Simulation Conference, 2004.

[20] C. Reynolds, "Steering behaviors for autonomous characters," in *Game Developers Conference 1999*, 1999.

[21] M. Buckland, *Programming Game AI by Example.* Wordware Publishing, Inc., 2005.

[22] C. Ericson, *Real–Time Collision Detection.* The Morgan Kaufmann Series in Interactive 3D Technology, Elsevier, 2005.