TUGAS AKHIR – TJ141502

Pengembangan Simulasi Lalu Lintas Menggunakan Graf Berbobot Berdasarkan Waktu Tempuh

Bagus Himawan
NRP 07211440000049

Dosen Pembimbing
Dr. Supeno Mardi Susiki Nugroho, ST., MT.
Eko Pramunanto, ST., MT.

JURUSAN TEKNIK KOMPUTER
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2018

FINAL PROJECT – TJ141502

Development of Traffic Simulation using time-Weighted Graphs

Bagus Himawan
NRP 07211440000049

Advisor
Dr. Supeno Mardi Susiki Nugroho, ST., MT.
Eko Pramunanto, ST., MT.

DEPARTMENT OF COMPUTER ENGINEERING
Faculty of Electrical Technology
Sepuluh Nopember Institute of Technology
Surabaya 2018

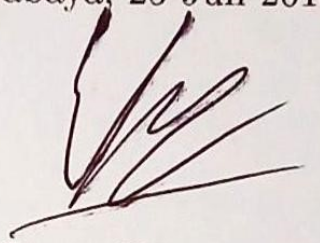
PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Pengembangan Simulasi Lalu Lintas Menggunakan Graf Berbobot Berdasarkan Waktu Tempuh**" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 25 Juli 2018



Bagus Himawan

NRP. 07211440000049

LEMBAR PENGESAHAN

Pengembangan Simulasi Lalu Lintas Menggunakan Graf Berbobot Berdasarkan Waktu Tempuh

Tugas Akhir ini disusun untuk memenuhi salah satu syarat memperoleh gelar Sarjana Teknik di Institut Teknologi Sepuluh Nopember Surabaya

Oleh: Bagus Himawan (NRP: 07211440000049)

Tanggal Ujian : 29 Juni 2018

Periode Wisuda : Maret 2019

Disetujui oleh:

(Pembimbing I)

Dr. Supeno Mardi Susiki N., ST., MT.
NIP. 197003131995120001

(Pembimbing II)

Eko Pramunanto, ST., MT.
NIP. 196612031994121001

(Penguji I)

Dr. I Ketut Eddy Purnama, ST., MT.
NIP. 196907301995121001

(Penguji II)

Dr. Surya Sumpeno, ST., M.Sc.
NIP: 196906131997021003

(Penguji III)

Dr. Eko Mulyanto Yuniarno, ST., MT.
NIP: 196806011995121009

Mengetahui
Kepala Departemen Teknik Komputer

Dr. I Ketut Eddy Purnama, ST., MT.
NIP. 196907301995121001

ABSTRAK

Nama Mahasiswa	:	Bagus Himawan
Judul Tugas Akhir	:	Pengembangan Simulasi Lalu Lintas Menggunakan Graf Berbobot Berdasarkan Waktu Tempuh
Pembimbing	:	1. Dr. Supeno Mardi Susiki Nugroho, ST., MT. 2. Eko Pramunanto, ST., MT.

Setiap tahunnya jumlah kendaraan yang ada semakin bertambah, sehingga kepadatan lalu lintas juga ikut meningkat. Harga kendaraan roda 4 saat ini yang lebih banyak menjangkau kalangan masyarakat, didukung dengan para pelaku industri otomotif yang memberikan fasilitas pembayaran yang lebih terjangkau meningkatkan jumlah kendaraan yang ada. Dengan semakin banyaknya kendaraan di jalan raya, maka kondisi lalu lintas akan semakin ramai dan padat, kemacetan pun menjadi konsumsi sehari-hari pengguna jalan terutama masyarakat perkotaan, untuk menguraikan kemacetan tersebut dapat dengan pengalihan jalan atau penutupan jalan sehingga kepadatan menjadi berkurang. Dengan menggunakan simulasi lalu lintas penutupan atau pengalihan jalan dapat lebih optimal, penggunaan simulasi dapat membantu pemilihan jalan mana yang paling optimal jika suatu jalam ditutup, hal ini sangat membantu dalam pengaturan lalu lintas di perkotaan sehingga pengguna jalan dapat terhindar dari kemacetan dan tetap melewati jalan yang tercepat.

Kata Kunci : Simulasi; Sistem Lalu Lintas; *Graph Theory*

Halaman ini sengaja dikosongkan

ABSTRACT

Name	:	Bagus Himawan
Title	:	*Development of Traffic Simulation using time-Weighted Graphs*
Advisors	:	1. Dr. Supeno Mardi Susiki Nugroho, ST., MT. 2. Eko Pramunanto, ST., MT.

Each year the number of vehicles increases, so the traffic density also increases. Current prices of four-wheeled vehicles that more addordable by the people, are supported by automotive industry players who provide more affordable payment facilities to increase the chance for people to having a car. With more vehicles on the highway, traffic conditions will become more crowded and congested, traffic congestion becomes the daily consumption of road users, especially urban communities, to decipher congestion by road diversion or road closure so density becomes reduced. By using a closing traffic simulation or road diversion can be more optimal, the use of simulation can help the most optimal selection of path if the jalam closed, very helpful in traffic management in urban areas so that road users can avoid congestion and stay through the fastest way.

Keywords : Simulation; Traffic System; Graph Theory

Halaman ini sengaja dikosongkan

KATA PENGANTAR

Puji dan syukur kehadirat Allah SWT atas segala limpahan berkah, rahmat, serta hidayah-Nya, penulis dapat menyelesaikan penelitian ini dengan judul **Pengembangan Simulasi Lalu Lintas Menggunakan Graf Berbobot Berdasarkan Waktu Tempuh**.

Penelitian ini disusun dalam rangka pemenuhan bidang riset di Departemen Teknik Komputer ITS serta digunakan sebagai persyaratan menyelesaikan pendidikan S1. Penelitian ini dapat terselesaikan tidak lepas dari bantuan berbagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Keluarga, Ibu, dan Ayah yang telah memberikan dorongan spiritual dan material dalam penyelesaian buku penelitian ini.
2. Kepala Departemen Teknik Komputer ITS, Dr. I Ketut Eddy Purnama, ST., MT.
3. Dr. Supeno Mardi Susiki Nugroho, ST., MT., M.Sc. dan Eko Pramunanto, ST., MT. atas bimbingan selama mengerjakan penelitian.
4. Bapak-ibu dosen pengajar Bidang Departemen Teknik Komputer, atas pengajaran, bimbingan, dan perhatian yang diberikan kepada penulis selama ini.
5. Seluruh teman-teman *B201-crew* Laboratorium Teknik Komputer dan Telematika.

Kesempurnaan hanya milik Allah SWT, untuk itu penulis memohon segenap kritik dan saran yang membangun. Semoga penelitian ini dapat memberikan manfaat bagi kita semua. Amin.

Surabaya, Juni 2018

Penulis

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR KODE

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

1.1 Latar belakang

Sebuah simulasi ditujukan untuk menggambarkan suatu sistem atau proses dengan peragaan berupa model statistik atau pemeranan. Dalam simulasi lalu lintas maka simulasi harus bisa menggambarkan suatu keadaan sistem lalu lintas seperti kendaraan, persimpangan jalan, dan kepadatan lalu lintas ke dalam simulasi. Simulasi lalu lintas yang dapat menggambarkan suatu sistem yang kompleks dapat digunakan pada banyak hal, misalnya untuk melakukan pengalihan jalan, melihat dampak dari aturan lalu lintas yang baru, untuk mencari pengaturan waktu lampu lalu lintas yang tepat, dan lain-lain. Berdasarkan data dari GAIKINDO (Gabungan Industri Kendaraan Bermotor Indonesia) hingga Agustus 2017 terdapat lebih dari 800 ribu kendaraan baru yang meluncur ke jalanan Indonesia [1], salah satu cara inovatif untuk mengurai kemacetan adalah mengatur batas kecepatan, dan memecah gelombang lalu lintas tertentu [2], sehingga diperlukan suatu tools untuk mendapatkan suatu rekayasa lalu lintas yang tepat. Namun pada simulasi lalu lintas yang ada sekarang hanya dapat memvisualisasikan, dan merekayasa aturan lalu lintas pada satu perempatan saja, sedangkan agar bisa digunakan diperlukan simulasi dalam skala yang lebih besar, yang dapat mengatur dan menampilkan akibat dari beberapa aturan ruas jalan sekaligus, serta memungkinkan untuk memodifikasi atau mengubah aturan jalan yang biasa disebut rekayasa lalu lintas. Untuk mengetahui apakah suatu rekayasa sudah cukup efektif, diperlukan suatu pengukuran sebagai patokan selain visual, yaitu waktu tempuh pada sebuah ruas jalan, permasalahan tersebut dapat diselesaikan dengan menggunakan graf berbobot dengan nilai tempuh sebagai bobotnya, pada suatu graf satu ruas jalan sama dengan satu buah node yang menyimpan nilai kendaraan yang ada pada jalan tersebut, batas kecepatan, dan panjang dari node, parameter-parameter tersebut digunakan untuk menghitung waktu delay yang diakibatkan oleh kepadatan lalu lintas untuk digunakan sebagai nilai bobot suatu node. Pengembangan pada simulasi se-

hingga dapat menampilkan kondisi lalu lintas yang kompleks perlu dilakukan, dengan begitu simulasi lalu lintas ini bisa digunakan oleh lembaga-lembaga tertentu atau penelitian lebih lanjut.

1.2 Permasalahan

Berdasarkan kondisi awal simulasi lalu lintas ini, maka dapat dirumuskan permasalahan sebagai berikut:

1. Semakin meningkatnya jumlah kendaraan di jalan, sehingga diperlukan sebuah rekayasa lalu lintas yang dapat mengurai kepadatan secara optimal.
2. Diperlukannya suatu indikator untuk menentukan apakah suatu jalan bisa dilalui atau ditempuh dengan waktu yang optimal atau lebih lama.
3. Belum adanya program simulasi yang dapat memvisualisasikan suatu kondisi lalu lintas jika diterapkan suatu aturan tertentu.

1.3 Tujuan

Mengembangkan suatu simulasi lalu lintas yang dapat memvisualisasikan suatu kondisi lalu lintas jika diterapkan suatu rekayasa tertentu, dengan adanya simulasi ini diharapkan pengguna dapat melihat hasil kondisi kepadatan lalu lintas jika diterapkan sebuah aturan lalu lintas tertentu.

Untuk mengatasi permasalahan tersebut maka perlu dikembangkannya sebuah *traffic simulation* yang berjalan di atas sistem operasi *PC/Windows* yang mampu:

1. Menampilkan persimpangan yang terkoneksi
2. Menampilkan kepadatan jalan
3. Menampilkan informasi tentang kepadatan di suatu ruas jalan
4. Memberikan pilihan jalan tercepat yang dipengaruhi oleh kepadatan lalu lintas
5. Mengubah peraturan lampu lalu lintas
6. Memvisualisasikan Pengalihan Jalan.

1.4 Batasan masalah

Untuk memfokuskan permasalahan yang akan diangkat maka dilakukan pembatasan masalah. Batasan-batasan masalah tersebut diantaranya adalah:

1. Simulasi yang akan dikembangkan pada tugas akhir ini hanya mencakup pada level makro saja, yang meliputi aturan dari sebuah jalan misalnya arah jalan, batas kecepatan, dan waktu lampu lalu lintas.
2. Visualisasi simulasi akan menggunakan Unity.
3. Kendaraan Sepeda Motor tidak termasuk dalam simulasi ini.
4. Jalan putar balik belum diterapkan pada simulasi ini.
5. Lalu lintas yang akan disimulasikan akan menampilkan hasil rekayasa lalu lintas di wilayah sidoarjo kota dan surabaya dari Merr hingga Galaxy mall.
6. Algoritma pathfinding yang digunakan adalah A*.
7. Faktor simulasi mikro tidak diperhitungkan dalam kalkulasi weight atau delay seperti sifat pengemudi dan sebagainya.

1.5 Sistematika Penulisan

Laporan penelitian Tugas akhir ini tersusun dalam sistematika dan terstruktur sehingga mudah dipahami dan dipelajari oleh pembaca maupun seseorang yang ingin melanjutkan penelitian ini. Alur sistematika penulisan laporan penelitian ini yaitu :

1. BAB I Pendahuluan
   Bab ini berisi uraian tentang latar belakang permasalahan, penegasan dan alasan pemilihan judul, sistematika laporan, tujuan dan metodologi penelitian.
2. BAB II Dasar Teori
   Pada bab ini berisi tentang uraian secara sistematis teori-teori yang berhubungan dengan permasalahan yang dibahas pada penelitian ini. Teori-teori ini digunakan sebagai dasar dalam penelitian, yaitu informasi terkait pajak reklame, formula perhitungan jarak dan teori-teori penunjang lainya.
3. BAB III Perancangan Sistem dan Implementasi
   Bab ini berisi tentang penjelasan-penjelasan terkait perencanaan sistem yang akan dilakukan dan fitur-fitur pada aplikasi. Guna mendukung itu digunakanlah blok diagram atau *work flow* agar sistem yang akan dibuat dapat terlihat dan mudah dibaca untuk implentasi pada pelaksanaan tugas akhir, pada bab ini juga menjelaskan tentang hasil implementasi dari rancangan aplikasi yang sudah dibuat sebelumnya. Setiap fitur

yang sudah dibuat akan ditunjukkan pada bab ini..

4. BAB IV Pengujian dan Analisa
   Bab ini menjelaskan tentang pengujian penelitian yang sudah dilakukan terhadap aplikasinya. Setiap fitur akan ditunjukkan hasilnya pada bab ini dan dilakukan analisa.
5. BAB V Penutup
   Bab ini merupakan penutup yang berisi kesimpulan yang diambil dari penelitian dan pengujian yang telah dilakukan. Penelitian lanjutan merupakan saran dan keritik yang membangun untuk pengembangan lebih lanjut juga ditulis pada bab ini.

BAB 2
TINJAUAN PUSTAKA

Demi mendukung penelitian ini, dibutuhkan beberapa teori penunjang sebagai bahan acuan dan refrensi. Dengan demikian penelitian ini menjadi lebih terarah.

2.1 OpenStreetMap

OpenStreetMap merupakan sebuah peta *Open Data*, data tersebut diperoleh dari komunitas *Mapper* atau pemeta yang berkontribusi mengatur data tentang lokasi yang ada seperti jalan raya, tempat makan, stasiun kereta apa, dan lain-lain. Beberapa kelebihan *OpenStreetMap* diantaranya :

2.1.1 Berbasis Komunitas

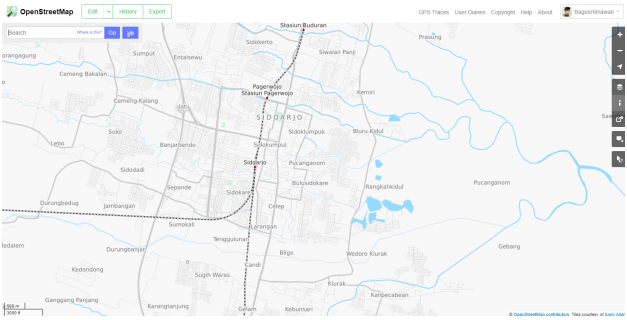
Komunitas *OpenStreetMap* terus berkembang dan semakin besar setiap harinya, pengguna *OpenStreetMap* mulai dari *mapper* sebagai pemeta jalan dan lain-lain, *GIS Progessionals* menggunakan data pada *OpenStreetMap* untuk memetakan informasi geografis, *Engineer* yang menggunakan data dari *OpenStreetMap, para relawan untuk menentukan atau memetakan area yang terkena bencana, dan lain sebagainya.*

2.1.2 *Open Data*

Data yang tersimpan oleh *OpenStreetMap* merupakan *Open Data*, layaknya *Open Source* pada sebuah software, *OpenStreetMap* memperbolehkan menggunakan data yang dimilikinya selama pengguna memberikan *credit* kepada *OpenStreetMap.*

2.1.3 Multifungsi

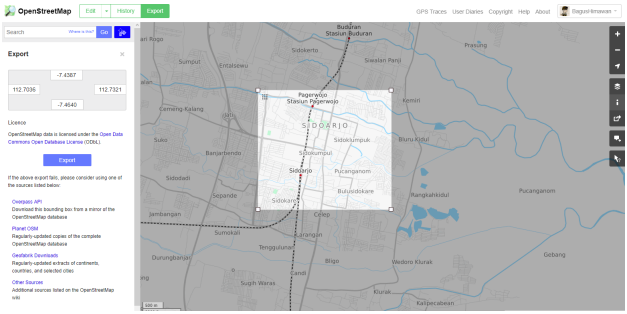
OpenStreetMap telah digunakan oleh ribuan *website*, aplikasi *mobile*, dan berbagai perangkat keras lainnya, sehingga *OpenStreetMap* dapat digunakan pada tugas akhir ini karena sudah teruji.



Gambar 2.1: Tampilan *OpenStreetMap*

2.2 *Export Data*

Karena bersifat *open data* mengambil data dari *OpenStreetMap* pun cukup mudah, cukup dengan memilih pilihan *export* pada website *OpenStreetMap*, area yang ingin diambil datanya juga bisa ditentukan dengan mudah



Gambar 2.2: Tampilan fitur export *OpenStreetMap*

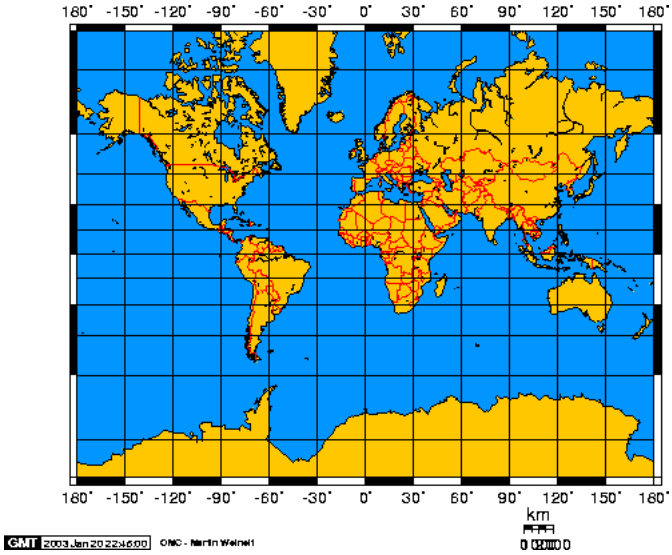
Data yang diperoleh merupakan hasil seluruh *mapping* yang dilakukan oleh *mapper*, data tersebut berisi data jalan, sungai, rambu-rambu lalu lintas, dan relasi antar jalan.

Setelah melakukan *export* pada area pilihan, *OpenStreetMap* akan memberikan kita sebuah file dengan format .OSM, yang jika dibuka oleh text editor isinya adalah tulisan dalam format xml

Gambar 2.3: Data yang didapat dari hasil export *OpenStreetMap*

2.3 *Mercator Projection*

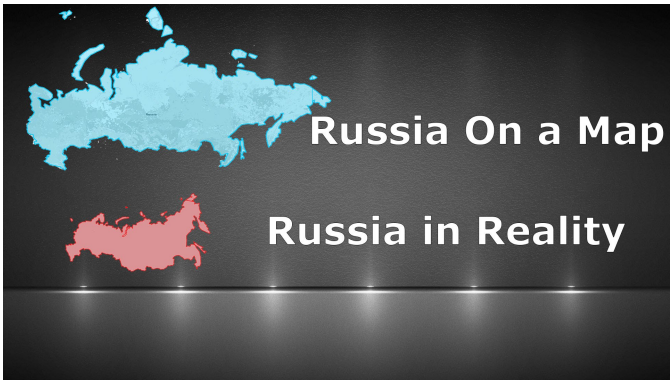
Peta yang ditampilkan oleh *OpenStreetMap* merupakan peta bumi kita dalam bentuk datar, sehingga terdapat perbedaan bentuk antara bentuk bumi yang asli dibandingkan dengan bentuk bumi pada peta.



Gambar 2.4: Tampilan Peta denga proyeksi *mercator*

Akibat dari penggunaan proyeksi *mercator* juga berpengaruh

terhadap ukuran dari suatu area, maka dari itu data yang diperoleh dari *OpenStreetMap* diharuskan untuk di projeksikan terlebih dahulu sebelum divisualisasikan pada program



Gambar 2.5: Perbedaan ukuran antara peta dan aslinya

2.4 Simulasi Makro

Simulasi pada level makro,pemodelan lalu lintas pada level makro menampilkan deskripsi lalu lintas secara global, pemodelan makro biasanya digunakan untuk pengaturan strategi atau rencana, serta mengatur arus lalu lintas dan regulasinya. Simulasi makro lebih ideal untuk menampilkan jaringan dalam skala besar, dalam simulasi lalu lintas dapat digunakan untuk memvisualisasikan sebuah jaringan jalan yang terkoneksi antara satu dengan yang lainnya. [3].

2.5 A* Pathfinding

Salah satu algoritma pathfinding untuk menghitung jarak terpendek, A* merupakan algoritma Dijkstra yang ditambahkan perhitungan heuristic untuk menghitung jarak ke tempat tujuan. Pada ilmu komputer algoritma ini banyak digunakan untuk memecahkan masalah pathfinding dan *graph traversal*, proses plotting node yang berhubungan tergolong cepat, sehingga banyak digunakan, selain itu akurasi dari perhitungan yang dihasilkan juga tergolong baik.

Algoritma A* merupakan salah satu pathfinding BFS atau

best-first search, yang berarti A* akan memproses dari suatu jalan yang sudah ditemukan dengan nilai terkecil terlebih dahulu, pada iterasi pertama A* akan mendapatkan semua jalan atau *edge* yang terhubung dengan *node*, setelah itu edge yang paling pendek akan dipilih untuk memproses *node* selanjutnya, pada perhitungan *node* selanjutnya hasil perhitungan sebelumnya akan digunakan untuk menghitung jarak total yang ditempuh. Rumus perhitungan jarak A* :

$$f(n) = g(n) + h(n) \tag{2.1}$$

g = jarak yang sudah ditempuh h = perhitungan heuristik

2.5.1 Sifat A*

1. Selalu menemukan solusi jika ada.
2. Dapat menggunakan metode *heuristic* untuk memperoleh hasil yang lebih cepat.
3. Mendukung *weighted graph*.
4. Dapat mencari jalan ke segala arah.

2.6 Graf Berbobot

Graf adalah diagram yang digunakan untuk menggambarkan berbagai struktur yang ada, pada simulasi lalu lintas bobot pada graf dapat digunakan untuk mengetahui nilai waktu delay dari kondisi lalu lintas yang ada pada node tersebut.

Biasanya bobot pada suatu graf merupakan jarak antara 2 node, namun pada kali ini sedikit berbeda, bobot yang digunakan merupakan satuan waktu untuk menuju suatu node dari node sebelumnya

2.7 Heuristic Method

Penggunaan hasil kalkulasi metode heuristik pada algoritma A* merupakan pembeda dari algoritm sejenis seperti Dijkstra, dengan penggunaan metode heuristik dapat memungkinkan algoritma berjalan lebih cepat, pada umumnya metode heuristik yang digunakan menghitung jarak lurus dari daerah asal ke tujuan, namun perhitungan heuristik yang digunakan dapat diubah sesuai dengan kebutuhan Rumus metode heuristik yang akan digunakan pada tu-

gas akhir ini :

$$h(n) = H + T \tag{2.2}$$

h(n) = Fungsi Heuristik H = Panjang garis lurus antara titik awal dan tujuan
T = Delay waktu pada suatu ruas jalan

2.7.1 Perhitungan Waktu Delay

Pada dunia nyata estimasi waktu delay dipengaruhi oleh kondisi yang bisa diketahui dan tidak bisa diketahui. Namun untuk mengetahui delay waktu yang ada pada suatu ruas jalan perlu adanya sesuatu yang bisa diukur, salah satunya adalah kecepatan rata-rata suatu kendaraan ketika memasuki jalan tersebut. Untuk menghitung delay waktu menggunakan kecepatan rata-rata dan panjang dari suatu ruas jalan dapat menggunakan rumus berikut :

$$T = \frac{L}{V} L/V[4] \tag{2.3}$$

L = Panjang dari ruas jalan V = Kecepatan Rata-Rata kendaraan

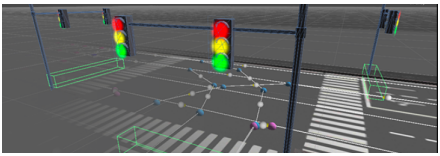
2.8 Wired Development Road and Traffic Framework

Sebuah plugin untuk unity yang memungkinkan untuk membuat secara cepat dan mensimulasikan sebuah jaringan jalan dalam skala besar maupun kecil dan memiliki berbagai jenis macam jalan dan lampu lalu lintas.

System ini dapat dengan mudah dimodifikasi, sehingga memungkinkan untuk menambahkan jenis jalan yang ada, ataupun menambahkan kendaraan baru.

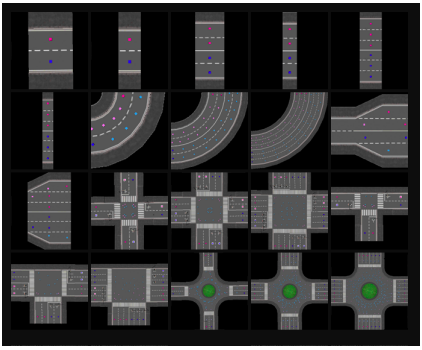
Fitur-Fitur yang disediakan :

1. Lampu lalu lintas



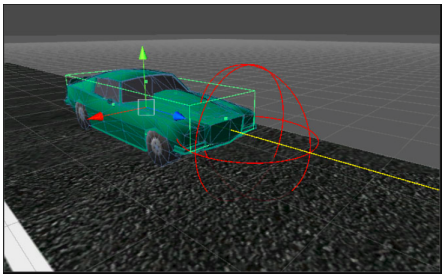
Gambar 2.6: Lampu lalu lintas

2. Asset jalan
   2 Lajur
   4 Lajur
   6 Lajur



Gambar 2.7: Aset Bagian Jalan

3. Kendaraan

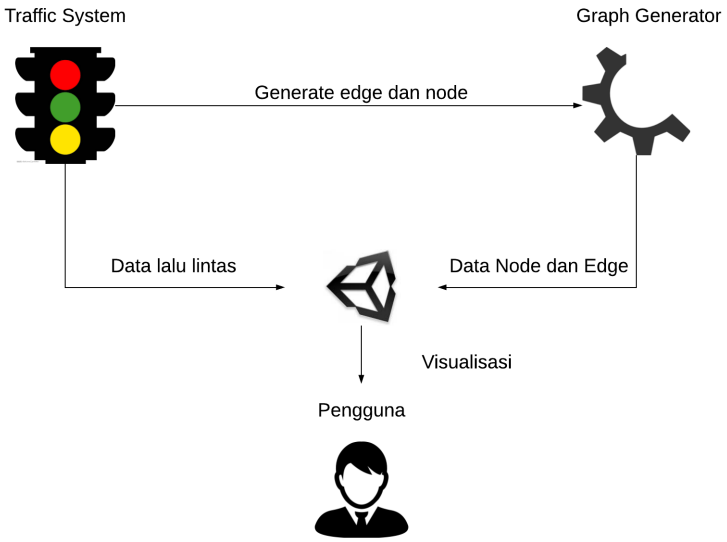


Gambar 2.8: Aset Kendaraan dan AI nya

Halaman ini sengaja dikosongkan

BAB 3
PERANCANGAN SISTEM DAN IMPLEMENTASI

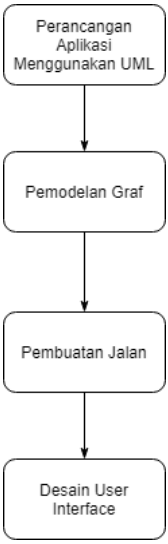
Penelitian ini bertujuan untuk mengembangkan sebuah simulasi lalu lintas yang dapat memvisualisasikan suatu kondisi lalu lintas jika diterapkan suatu rekayasa tertentu, dengan adanya simulasi ini diharapkan pengguna dapat melihat hasil kondisi kepadatan lalu lintas jika diterapkan sebuah aturan lalu lintas tertentu, kepadatan suatu lalu lintas ditandai dengan semakin lamanya waktu untuk menempuh atau melewati jalan tersebut.



Gambar 3.1: Gambaran umum sistem aplikasi

Gambar 3.1 Menjelaskan bagaimana sistem simulasi bekerja, untuk menghasilkan graph sistem lalu lintas dari *framework traffic system* milik WiredDevelopment, diperlukan adanya generator

graph untuk menentukan mana bagian node, edge, dan hubungan antar node beserta edge yang bersangkutan. Setelah graph sistem lalu lintas sudah selesai, maka perhitungan weight akan dilakukan, hasil dari perhitungan weight akan ditampilkan ke pengguna, pengguna dapat melihat informasi tentang kepadatan jalan, dan jalan tercepat dari sebuah node ke node lainnya



Gambar 3.2: Diagram tahapan pengembangan

Terdapat empat tahap utama dalam merancang dan mengimplementasikan desain sistem aplikasi sehingga dapat digunakan sebagai luaran akhir seperti pada Gambar 3.2. Berikut merupakan tiga tahap tersebut :

1. **Perancangan Aplikasi Menggunakan UML:** Merupakan tahap mendesain bagaimana simulasi akan berkerja, dan fitur apa saja yang akan dihadirkan pada simulasi.
2. **Pemodelan Graf:** Dalam tahap ini graf dimodelkan dari data yang sudah ada yang disimpan dalam bentuk .xml
3. **Pembuatan Jalan:** Setelah graf sudah dimodelkan, model jalan pun digenerate berdasarkan data yang ada pada graf

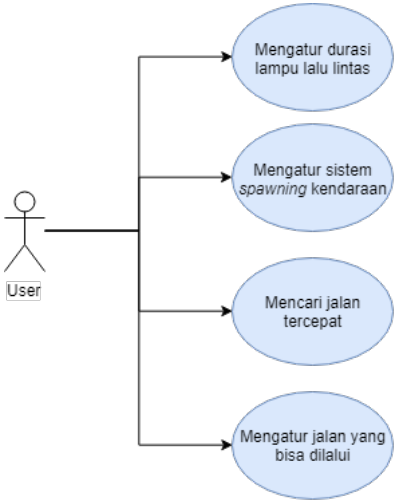
sebelumnya, pada tahap ini juga akan terlihat bagian mana saja data graf yang kurang akurat sehingga perlu diperbaiki.

4. **Desain *User Interface*:** Tahap ini berfungsi untuk merancang antar muka dari aplikasi yang nantinya akan dipakai sebagai acuan pembuatan aplikasi.

3.1 Desain Aplikasi Menggunakan *UML*

Desain sistem berdasarkan konsep berorientasi objek pada tugas akhir ini akan dijelaskan dalam beberapa diagram berikut.

3.1.1 *Use Case Diagram*

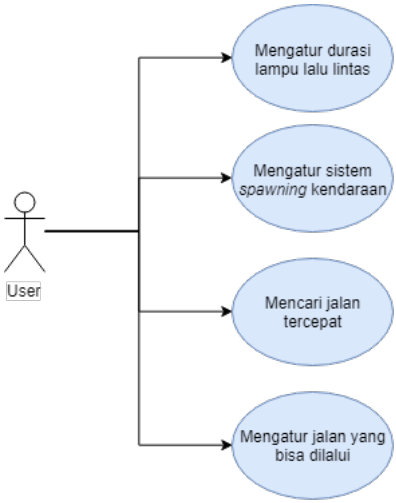


Gambar 3.3: Diagram *Use case*

Pada gambar 3.3 menjelaskan tentang interaksi yang dapat dilakukan oleh pengguna, diantaranya

1. Mengatur durasi lampu lalu lintas
2. Mengatur sistem *spawning* kendaraan
3. Mencari jalur tercepat antara 2 *node*
4. Mengatur jalan yang bisa dilalui.

3.1.2 *Activity Diagram*



Gambar 3.4: Diagram *Use case*

Gambar 3.4 menjelaskan tentang *activity diagram* dari sistem lalu lintas.

1. Saat program pertama dijalankan program akan memetakan graf berdasarkan sistem jalan yang sudah terhubung
2. setelah itu sistem akan menghubungkan tiap node yang ada dengan sebuah edge
3. setelah *graf* berhasil terbentuk sistem akan memunculkan kendaraan sesuai dengan aturan yang dibuat oleh pengguna
4. sistem sudah siap digunakan dan siap menerima input dari pengguna.
5. program terhenti jika pengguna telah selesai menggunakan program.

3.2 Pemodelan Graph Lalu Lintas

Pemodelan graf merupakan tahap yang penting sebelum simulasi digunakan, dengan terbentuknya graf kondisi simulasi lalu lintas dapat lebih terukur, dan hubungan antar bagian jalan lebih tertata. Untuk mendapatkan graf dari sistem lalu lintas yang sudah

diatur, perlu digunakan generator graf dan jalan, generator ini bertugas untuk menentukan jalan bagian mana yang merupakan node, bagaimana node terhubung dengan node lainnya, dan jalan bagian mana yang bukan merupakan node, generator ini juga menentukan *weight* awal pada seluruh edge yang ada, dengan begitu jaringan graf yang terbentuk bisa digunakan untuk *pathfinding*. Dengan sudah terbentuknya graf perhitungan delay akibat lalu lintas yang ada pun menjadi bisa dilakukan sehingga selain *weight* awal.

Jalan yang akan dimodelkan pada simulasi akan memiliki berbagai jenis jalan yang unik, misalnya jalan satu arah, persimpangan dengan lampu lalu lintas, persimpangan tanpa lampu lalu lintas, persimpangan dengan tiga jalan atau biasa disebut pertigaan, dan perempatan

Gambar 3.5: Area yang dimodelkan

Pada gambar 3.5 menunjukan area yang akan dimodelkan, pada area tersebut terdapat beberapa persimpangan, beberapa jalan

satu arah, dan beberapa jalan dengan dua arah.

3.3 Pembuatan Jalan

Untuk menghasilkan suatu graf yang memiliki node, dan edge beserta model jalan yang sudah terhubung diperlukan suatu *generator* yang dapat memodelkan data jalan dari *OpenStreetMap* dan memetakannya sehingga terbentuk hubungan antar node sehingga terbentuklah sebuah graf suatu jalan.

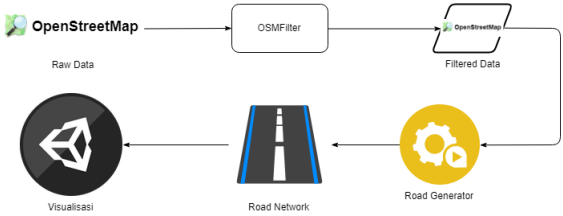
3.3.1 Pengolahan Data *OpenStreetMap*

Tidak semua data yang diperoleh dari *OpenStreetMap* akan digunakan, banyak sekali data-data yang tidak perlu seperti jalur sungai dan jalan-jalan yang terlalu kecil seperti jalan yang hanya bisa dilalui oleh motor, tipe data yang dibutuhkan untuk pemodelan jalan adalah sebagai berikut :

1. **Highway:** Merupakan data yang menyimpan data tentang jalan, tidak semua jenis jalan yang akan disimpan, beberapa diantaranya adalah :
   (a) Motorway
   (b) Trunk
   (c) Primary
   (d) Secondary
   (e) Tertiary
   (f) Unclassified
   (g) Motorway link
   (h) Trunk link
   (i) Primary link
   (j) Secondary link
2. **Relation:** Menyimpan segala informasi tentang bagaimana setiap data jalan terhubung satu dengan lainnya, selain menyimpan informasi bagaimana tiap jalan terhubung *relation* juga menyimpan tentang larangan-larangan yang ada pada suatu jalan, misalnya dilarang belok kiri dan sebagainya.

3.4 *Road Generator*

Dengan sudah didapatkannya data dari *OpenStreetMap* yang sudah di-*export* dan difilter sehingga akan lebih mudah dimodelkan. Maka *graf* jalan akan dapat terbentuk.



Gambar 3.6: Diagram Tahapan Pembuatan Graf

Dijelaskan pada gambar 3.6 bagaimana data yang didapat dari *OpenStreetMap* dapat diubah menjadi objek jalan yang saling terhubung satu sama lain :

1. Mengambil data dari *OpenStreetMap*
2. Memfilter data yang diperoleh sebelumnya menggunakan alat OSMFilter dengan *command* sebagai berikut :

Kode 3.1: Memfilter data *OpenStreetMap*

```
osmfilter.exe Sidoarjo.osm
--keep="highway=motorway =trunk =primary
=secondary =tertiary =unclassified
=motorway_link =trunk_link =primary_link
=secondary_link =traffic_signals
and visible=true "
--keep-relations="restriction"
> SidoarjoWays.osm
```

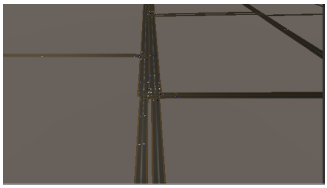
3. Data yang sudah terfilter akan berkurang ukuran filenya dan data yang tidak dibutuhkan sudah tidak ada, sehingga sudah bisa digunakan untuk membuat jaringan jalan yang diinginkan.
4. Road generator akan menterjemahkan data yang sudah difilter ke dalam format yang bisa ditampilkan oleh unity, seperti mengubah nilai latitude dan longitude ke dalam bentuk vektor tiga dimensi yang biasa unity gunakan.

5. Setelah jaringan jalan berhasi digenerasi secara otomatis, diperlukan beberapa koreksi secara manual agar jaringan jalan yang ditampilkan sesuai aslinya, hal ini dikarenakan dari *mapper* yang memetakan jalan pada area terpilih tidak lengkap dalam mengisi informasi jalan tersebut.
6. Jika data jaringan jalan sudah diperbaiki secara manual, maka jaringan jalan pun sudah siap untuk divisualisasikan dengan sistem lalu lintas yang sudah ada.

3.5 Perancangan Antar Muka

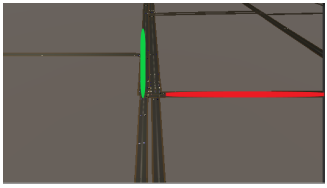
Rancangan antar muka pada simulasi lalu lintas ini adalah sebagai berikut :

3.5.1 Tampilan awal



Gambar 3.7: Tampilan Awal Simulasi

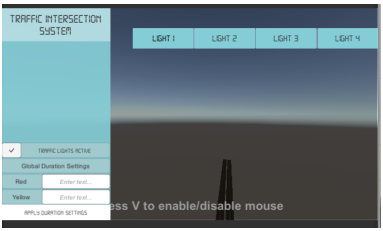
3.7 pada simulasi ini hanya menunjukan saat pertama kali program dijalankan.



Gambar 3.8: Setelah Simulasi Berjalan

Ketika simulasi sudah berjalan sehingga kepadatan kendaraan mulai ada maka tiap jalan akan memberi indikator seperti di gambar3.8

3.5.2 Tampilan pengaturan *traffic lights*



Gambar 3.9: Setelah Simulasi Berjalan

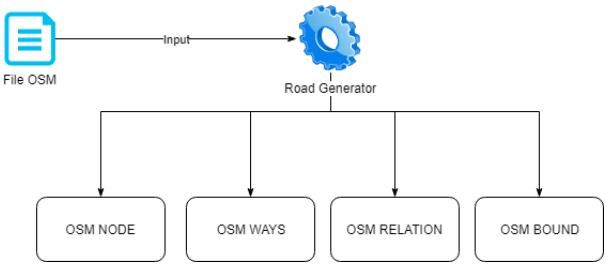
Pada gambar 3.9 merupakan tampilan pengaturan lampu lalu lintas, yang dapat mengatur tiap lampu lalu lintas dan aturan keseluruhan.

Jika suatu persimpangan lampu merah di pilih oleh pengguna, maka pengguna dapat mengubah pengaturan yang ada pada lampu lalu lintas tersebut.

3.6 Implementasi Sistem Lalu Lintas

Implementasi sistem lalu lintas menggunakan Unity sebagai *toolkit* pembuatan simulasi dan membaca file dalam format .xml untuk data yang diperoleh dari *OpenStreetMap*, diperlukan beberapa tools yang harus dibuat untuk membaca file tersebut.

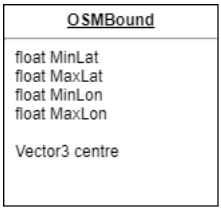
3.6.1 *Road Network Generator*



Gambar 3.10: Cara Kerja Road Generator

Road Generator bertugas untuk membaca informasi yang diberikan oleh sebuah file, file tersebut merupakan file .osm yang sudah di filter terlebih dahulu, seperti pada 3.10 tahapan pemetaan adalah sebagai berikut :

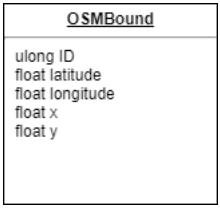
1. *Road Generator* meneripa input file yang berisikan data *OpenStreetMap*
2. Data yang ada akan dibagi menjadi 3 bagian yaitu :
   (a) ***OSM BOUND***



Gambar 3.11: Struktur Data OSMBound

Data *bound* yang berarti batas, merupakan data yang menyimpan batas daerah yang akan disimulasikan, dengan adanya data ini, konversi nilai *latitude,longitude* ke nilai posisi vektor 3 dapat dilakukan dengan mudah.

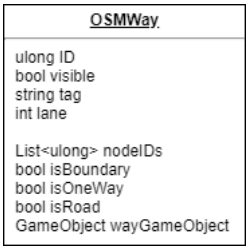
   (b) ***OSM NODE***



Gambar 3.12: Struktur Data OSMNode

Data *node* meruapakan titik-titik penting yang ada pada peta, pada titik ini bisa terdapat tikungan, rambu-rambu, dan sebagainya.

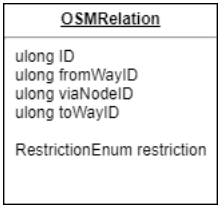
   (c) ***OSM WAY***



Gambar 3.13: Struktur Data OSMNode

Jalan yang tersimpan pada data akan termasuk dalam kategori *OSMWay*, data ini akan menyimpan titik-titik atau *OSMNode* mana saja yang termasuk bagian dari jalan ini, selain itu data ini juga menyimpan informasi yang lain seperti jumlah lajur, maupun keterangan satu arah atau tidak.

(d) ***OSM RELATION***

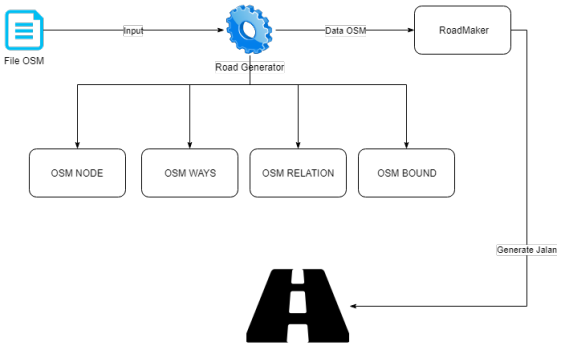


Gambar 3.14: Struktur Data OSMRelation

Informasi hubungan antar jalan disimpan dalam OSM-Relation, tanpa adanya data ini hubungan antar jalan tidak akan ada dan graf tidak bisa terbentuk, namun disayangkan data yang ada sering kurang sehingga perlu dilakukan penyesuaian manual.

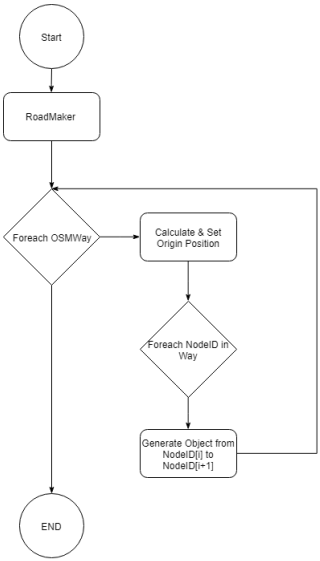
Jika proses pembacaan data sudah selesai, data yang sudah siap digunakan tersebut akan dibaca oleh script *RoadMaker* yang bertugas untuk membuat objek jalan sesuai dengan data yang ada.

3.6.2 *Road Maker*



Gambar 3.15: Cara Kerja *RoadMaker*

Pada 3.15 data yang sudah diproses sebelumnya akan diolah lagi oleh *RoadMaker* untuk membuat objek dan model jalan di Unity



Gambar 3.16: *RoadMaker Flow Chart*

Bagaimana cara kerja *RoadMaker* yang lebih detail dapat dilihat pada *Flow Chart*, dimana tahapannya adalah sebagai berikut :

1. **Membaca keseluruhan data:** Semua data jalan yang sudah diolah menjadi objek OSMWay diiterasi satu persatu
2. **Menghitung titik pusat:** Titik pusat sebuah jalan yang didapat dari perhitungan menggunakan *Mercator Projection*
3. **Memproses tiap *Node ID*:** Seluruh *Node ID* yang dilewati oleh suatu jalan menjadikan titik patokan untuk pembuatan objek
4. ***Generate Object*:** Objek yang akan dibuat dari titik atau *node* ke titik yang lain dibuat berdasarkan informasi jumlah lajur yang ada di jalan tersebut, dan apakah jalan tersebut merupakan satu arah atau tidak.

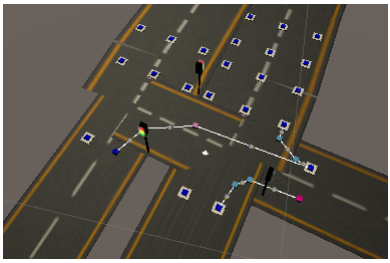
3.6.3 Penyempurnaan Data

Data yang kurang tepat dengan keadaan aslinya perlu dikoreksi dengan membandingkan kondisi pada jalan aslinya, beberapa kekurangan yang paling sering terjadi :

1. Jumlah Lane
2. Lampu Lalu Lintas
3. Satu arah
4. Informasi Persimpangan

Untuk informasi data nomor satu dan tiga, semuanya dapat dikoreksi dengan mengubah atau menambahkan value pada data jalan yang bersangkutan, untuk mengubah informasi jumlah lajur yang ada, cukup menambahkan *tag* lane dengan nilai jumlah lajur yang ada, sedangkan pada untuk data satu arah cukup menambahkan *tag* OneWay dan nilai *Yes/No*, namun secara default setiap jalan akan dianggap sebagai satu arah.

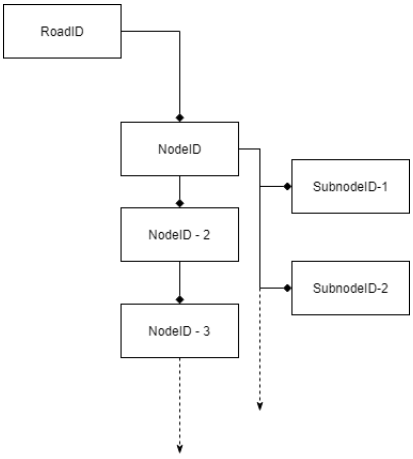
Sedangkan untuk nomor dua dan empat diperlukan perubahan manual dengan memanfaatkan fitur *editor* pada Unity



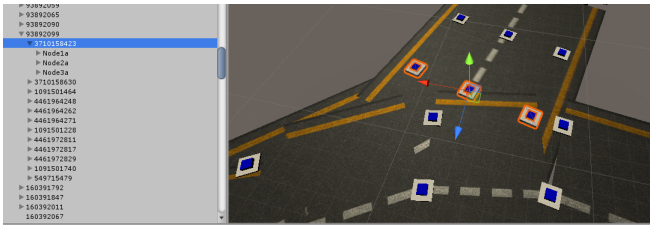
Gambar 3.17: *Pemodelan yang harus dilakukan manual*

Gambar 3.17 adalah bagian lampu lalu lintas, dan arah belok persimpangan yang harus diatur manual, selain karena data yang ada biasanya kurang lengkap, terkadang tiap persimpangan memiliki peraturan yang unik sehingga untuk saat ini *generator* jalan yang ada belum bisa melakukan otomasi pada bagian tersebut.

Jika tahap ini sudah selesai, hal terakhir yang dilakukan selanjutnya adalah menghubungkan antar node sesuai dengan data relasi dan data jalan yang ada.



Gambar 3.18: *Hierarchy object* hasil generator



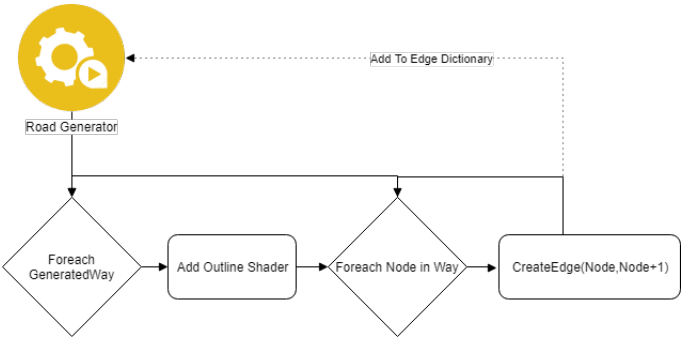
Gambar 3.19: *Hierarchy object* pada unity

Pada 3.18 adalah hasil *hierarchy* object yang telah selesai digenerate oleh *RoadMaker*, sedangkan gambar 3.19 menunjukan hasil objek yang dibuat pada *engine* Unity.

3.7 Implementasi Pathfinding

3.7.1 *Edge Generator*

Setelah sistem lalu lintas sudah selesai digenerate dan dikoreksi, maka langkah selanjutnya adalah menerapkan sistem pathfinding dengan memanfaatkan *node* yang sudah ada, setiap *OSMNode* akan dianggap sebuah node, namun *edge* yang menghubungkan antar *node* belum sepenuhnya terbentuk, *edge* yang menjadi penghubung antar *node* perlu di-*generate* terlebih dahulu. Untuk menghasikan *edge* dari data yang sudah ada maka tahapan yang dilakukan adalah sebagai berikut :



Gambar 3.20: *Flowchart Edge Generator*

Penjelasan *flow chart* diatas adalah :

1. Iterasi seluruh jalan yang ada
2. Menambahkan outline shader yang digunakan sebagai indikator kepadatan
3. Iterasi seluruh *node* yang dimiliki jalan yang sedang diproses
4. Membuat *edge* yang memiliki titik awal *node* ke i, dan *node* ke i+1.

Setelah selesai melakukan hal diatas jaringan jalan terlah terbentuk secara sempurna, dan siap digunakan.

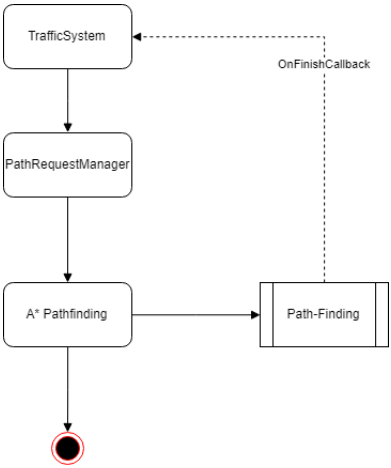
3.7.2 A* *Pathfinding*

Dengan sudah terbentuknya jaringan jalan secara keseluruhan, maka pencarian jalan terpendek dapat dilakukan, unuk melakukannya diperlukan sebuah algorima pencarian jalan atau yang biasa disebut *pathfinding*

Berikut merupakan pseudecode dari algoritma A* :

```
1:  function A*(start, goal)
2:  open_list = set containing start
3:  closed_list = empty set
4:  start.g = 0
5:  start.f = start.g + heuristic(start, goal)
6:  while open_list is not empty
7:  current = open_list element with lowest f cost
8:  if current = goal
9:  return construct_path(goal) // path found
10: remove current from open_list
11: add current to closed_list
12: for each neighbor in neighbors(current)
13: if neighbor not in closed_list
14: neighbor.f = neighbor.g + heuristic(neighbor, goal)
15: if neighbor is not in open_list
16: add neighbor to open_list
17: else
18: openneighbor = neighbor in open_list
19: if neighbor.g < openneighbor.g
20: openneighbor.g = neighbor.g
21: openneighbor.parent = neighbor.parent
22: return false // no path exists
23:
24: function neighbors(node)
25: neighbors = set of valid neighbors to node // check for obstacles here
26: for each neighbor in neighbors
27: if neighbor is diagonal
28: neighbor.g = node.g + diagonal_cost // eg. 1.414 (pythagoras)
29: else
30: neighbor.g = node.g + normal_cost // eg. 1
31: neighbor.parent = node
32: return neighbors
33:
34: function construct_path(node)
35: path = set containing node
```

```
36:  while node.parent exists
37:  node = node.parent
38:  add node to path
39:  return path
```



Gambar 3.21: *Flowchart Pathfinding Manager*

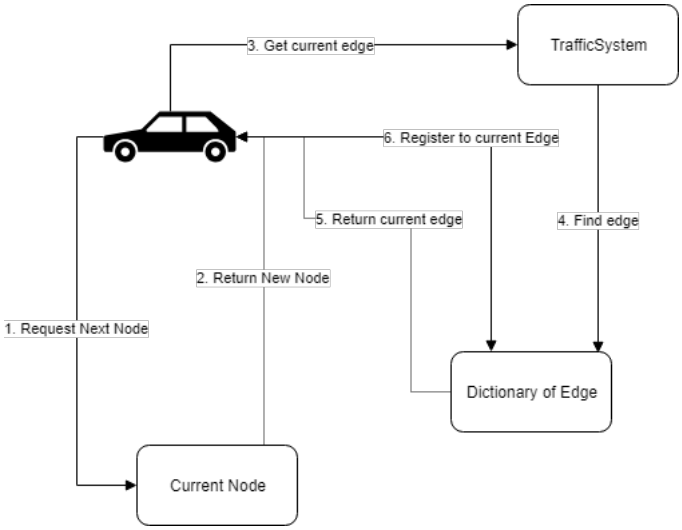
Seperti pada gambar 3.21 setiap permintaan pencarian jalan harus melewati *PathRequestManager* terlebih dahulu, setiap meminta jalan permintaan tersebut akan dimasukan ke sebuah *list queue* pada *PathRequestManager*, permintaan jalan akan dikerjakan secara runtut mulai dari yang paling awal meminta arah, dan hasilnya akan diterima oleh peminta arah melalui sebuah callback, input yang diperlukan oleh *PathRequestManager* adalah titik awal, tujuan, dan *callback* untuk menerima hasil pencarian jalan nantinya.

3.7.3 Perhitungan Delay

Gambar 3.22: Perhitungan Delay secara umum

Perhitungan delay sangat dipengaruhi dengan jumlah kendaraan yang ada serta kecepatannya, maka dari itu setiap kendaraan yang lewat perlu diregistrasi setiap masuk dan keluar sebuah edge

3.7.4 Pendata Kendaraan



Gambar 3.23: Mendaftarkan kendaraan pada *edge*

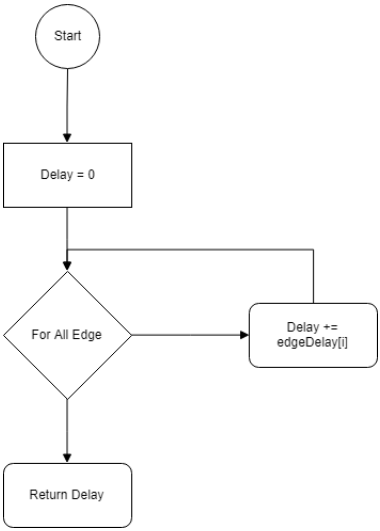
Kendaraan perlu diketahui sedang berada pada *edge* apa, sehingga kecepatan rata-rata dapat diperoleh, dan perhitungan delay dapat dihitung, pada 3.23 dijelaskan bagaimana tahapan suatu kendaraan dapat terdaftar pada edge tertentu.

1. Kendaraan meminta *node* selanjutnya untuk dilewati.
2. *Node* yang telah dilewati memberikan *node* selanjutnya yang terhubung.
3. Kendaraan akan mencari *edge* yang titik awalnya adalah *node* sebelumnya dan *node* yang baru kepada *trafficSystem*
4. *TrafficSystem* mencari *edge* berdasarkan data input yang diberikan pada sebuah *dictionary*
5. Jika ada maka *TrafficSystem* mengembalikan nilai berupa sebuah *edge* yang terhubung.
6. Kendaraan akan terdaftar sebagai kendaraan yang melintas

pada *edge* tersebut sehingga dapat memberikan informasi kecepatannya.

3.7.5 Menghitung *delay* pada sebuah jalan

Perhitungan *delay* dihitung oleh masing-masing jalan yang di dalamnya terdapat *edge-edge* yang menghubungkan antar *node*, *delay* yang akan ditampilkan merupakan hasil dari semua delay pada *edge* yang ada pada jalan itu.



Gambar 3.24: Flow Chart Perhitungan Delay Tiap Jalan

Sedangkan untuk rumus perhitungan *delay* tiap *edge* adalah sebagai berikut :

$$T = \frac{L}{V} \tag{3.1}$$

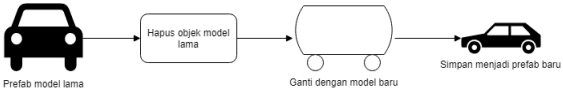
Dimana L merupakan panjang dari *edge* dan V adalah kecepatan kendaraan rata-rata yang ada pada *edge* tersebut, *delay* ini akan menjadi nilai *weight* pada jaringan jalan.

3.8 Penambahan Jenis Kendaraan

Jumlah kendaraan awal yang disediakan sangatlah sedikit, hanya menyediakan sebuah model kendaraan saja, maka dari itu kendaraan jenis lain ditambahkan agar visualisasi yang dihasilkan lebih maksimal

Gambar 3.25: Kendaraan baru

Kendaraan pada gambar 3.25 didapat dengan gratis dari Unity *Asset Store*, untuk menggunakan model kendaraan yang baru maka tahapannya adalah sebagai berikut :



Gambar 3.26: Memasang model baru

Tahapan memasang model kendaraan baru :

1. Pasangkan prefab kendaraan yang ada ke dalam editor
2. Hapus model kendaraan pada prefab
3. Pasangkan model baru, sesuaikan segala ukuran ban, *collider*, dan *collision detection* sesuai dengan kendaraan yang baru.
4. Simpan sebagai *prefab* baru

BAB 4
PENGUJIAN DAN ANALISA

Pada bab ini dilakukan pengujian fitur-fitur pada program simulasi lalu-lintas yang dikembangkan dengan graf berbobot. Pengujian dilakukan untuk mengetahui apakah dengan ditambahkannya fitur graf berbobot simulasi ini dapat memvisualisasikan kepadatan suatu lalu-lintas dengan baik.

4.1 Metode Pengujian

1. **Pengujian *User Interface***
   Untuk mengetahui fitur-fitur yang ada berjalan sebagai mana semestinya atau tidak, untuk menguji *user interface* digunakan beberapa skenario pengujian, skenario pengujian ditujukan untuk menilai fungsi-fungsi yang ada dapat diakses melalui *UI* pada simulasi ini.
2. **Pengujian Skenario**
   Untuk menguji graf berbobot waktu tempuh akan dibuat skenario tertentu untuk membuktikan jika kepadatan suatu lalu lintas dapat mempengaruhi jalan tercepat yang dihasilkan oleh *pathfinding*
3. **Pengujian Performa**
   Pengujian performa dilakukan apakah simulasi dapat menerima beban tanpa mengurangi performa, beberapa hal yang diuji adalah :

   (a) *Frame Per Second*
   Gambar yang dapat dihasilkan tiap detiknya mampu memberikan gambaran umum performa yang dihasilkan oleh simulasi.
   (b) *Resource Usage*
   Pada pengujian ini dapat diketahui bagian mana dari simulasi yang paling banyak menggunakan *resource*

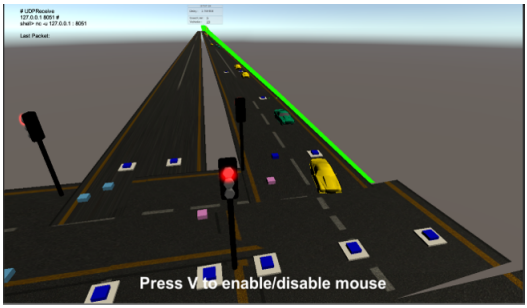
4.2 Pengujian *User Interface*

Pengujian *User Interface* digunakan untuk mengetahui apakah fitur-fitur yang tersedia dapat diakses dan berfungsi dengan baik, skenario pengujian dilakukan dengan mendesain skenario tertentu agar fitur-fitur yang ada dapat digunakan, kemudian akan dicek apakah sudah berjalan sesuai semestinya dan dapat dianggap bisa digunakan sesuai kebutuhan.

1. **Visualisasi Kepadatan**
   Setiap jalan yang sudah terhubung kedalam suatu *graf* maka jalan tersebut memiliki nilai bobot, pengujian ini dilakukan untuk mengetahui apakah kepadatan sebuah jalan sudah dapat divisualisasikan dengan baik.
2. **Pengujian Pengaturan Lalu Lintas**
   Pada bagian ini fitur mengubah aturan lalu lintas yang ada akan diuji apakah kendaraan yang disimulasikan dapat mengikuti aturan yang baru diubah.

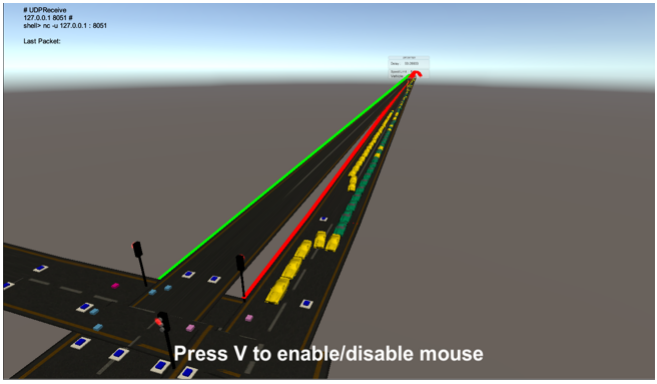
4.2.1 Visualisasi Kepadatan

Mirip seperti aplikasi peta milik google, jalan akan berwarna hijau jika sedang tidak ada kepadatan, dan jika kepadatan semakin meningkat maka warna akan menjadi lebih kuning hingga akhirnya menjadi warna merah. Skenario yang diberikan, simulasi akan berjalan selama 30 menit, setelah 30 menit berjalan lalu lintas akan terisi dan dapat terlihat perbedaan outline dari jalan tersebut, untuk mempercepat proses pengujian, pada setiap *node* yang ditentukan akan muncul kendaraan tiap 2 detik, berikut hasil awal saat program mulai berjalan :



Gambar 4.1: Saat awal program berjalan

Pada 4.1 mobil yang berada pada jalan tersebut masih terlihat sedikit, sehingga warna jalan pun masih berwarna hijau, beberapa menit kemudian kondisi tersebut akan berubah



Gambar 4.2: Kendaraan telah memadati jalan

Setelah beberapa saat dapat dilihat pada 4.2 jika kepadatan semakin meningkat, dan warna outline jalan telah berwarna merah, yang berarti jalan tersebut sangat padat.

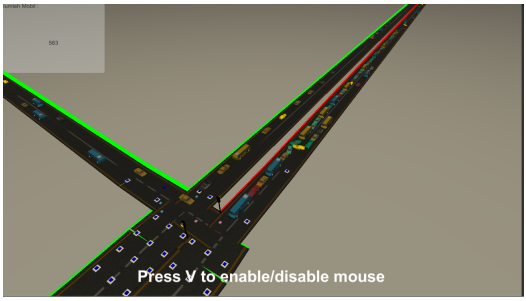
Dari dua gambar diatas bisa disimpulkan jika perhitungan *delay* dapat dipakai sebagai parameter untuk menunjukan suatu kepadatan, namun cara ini memiliki kelemahan, yaitu jika rata-rata kecepatan kendaraan jauh dari standardnya walaupun masih ada jalur

yang kosong akan tetap berwarna merah, karena dianggap memiliki delay yang lama.

4.2.2 Pengujian Pengaturan Lalu Lintas

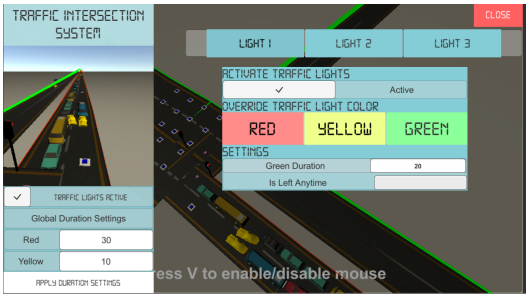
Suatu kepadatan lalu lintas biasanya berujung pada pengaturan durasi lampu lalu lintas yang kurang tepat, pada pengujian skenario ini ada beberapa hal yang di uji, setiap fitur pengaturan aturan lalu lintas akan di uji untuk mengetahui apakah perubahan aturan tersebut memberikan dampak pada kondisi simulasi lalu lintas.

1. **Pengubahan warna lampu**
   Pada skenario ini akan dilakukan pengujian terhadap efek dari perubahan warna lampu lalu lintas pada sebuah kemacetan.



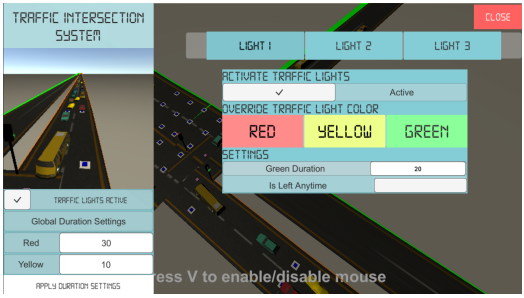
Gambar 4.3: Terjadi sebuah kepadatan

Awal kondisi lalu lintas dapat dilihat pada gambar 4.3, bisa terlihat jalan sangat padat, dan *outline* berwarna merah, pada kasus ini bisa diakibatkan lampu merah yang terlalu lama, salah satu solusi yang ada yaitu mengubah lampu lalu lintas menjadi warna hijau secara manual.



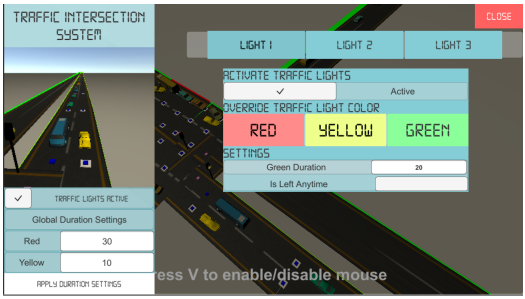
Gambar 4.4: Tampilan *UI* pengaturan

Untuk mengakses pengaturan yang ada, pilih salah satu dari tombol *light* yang terlihat pada gambar 4.4, jika salah satu tombol dipilih maka tampilan gambar pada sebelah kiri akan menampilkan kondisi lalu lintas pada lampu lalu lintas yang dipilih.



Gambar 4.5: Override warna lampu menjadi hijau

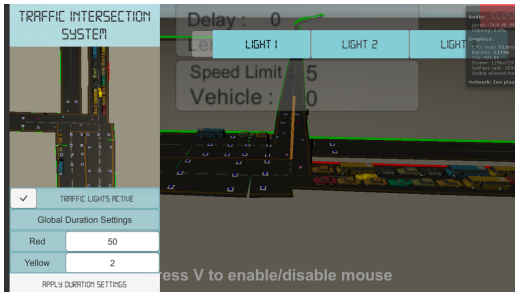
Ketika tombol dengan tulisan *green* ditekan, seketika lampu akan berwarna hijau sesuai dengan durasi warna hijau pada lampu lalu lintas tersebut, dan bisa dilihat pada 4.5 kendaraan akan langsung berjalan.



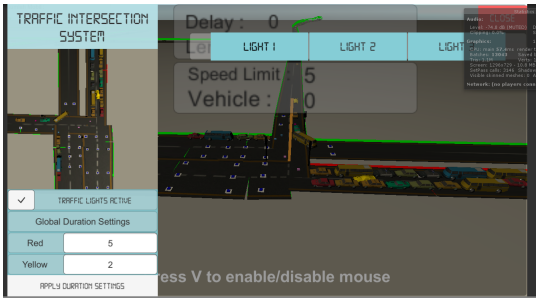
Gambar 4.6: Kendaraan telah memadati jalan

Setelah beberapa saat kendaraan yang tadinya berhenti sudah bisa berjalan dan kondisi jalan yang sebelumnya berwarna merah sudah berubah menjadi hijau

2. **Perubahan durasi lampu** Awal kejadian pada skenario ini sama seperti sebelumnya, yaitu adanya kepadatan pada sebuah persimpangan dengan lampu lalu lintas, namun kali ini yang dilakukan adalah mengubah durasi warna lampu pada persimpangan tersebut.



Gambar 4.7: Durasi awal lampu merah



Gambar 4.8: Durasi lampu merah telah diubah

Dengan kondisi kemacetan seperti gambar 4.3 namun dengan pengaturan durasi lampu merah seperti pada gambar 4.7, bisa dilihat durasi lampu merah yang mencapai 50 detik, dengan pengaturan seperti itu menghasilkan kepadatan pada salah satu bagian perempatan, maka dari itu durasinya dikurangi menjadi seperti pada gambar 4.8

Gambar 4.9: Durasi lampu hijau

Durasi lampu warna hijau pada jalur yang terkena kemacetan sudah di set cukup yaitu 20 detik, seperti pada 4.9 sehingga tidak perlu perubahan nilai durasi lampu hijau.

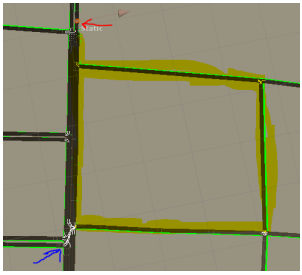
Berdasarkan pengujian yang sudah dilakukan diatas, perubahan aturan lalu lintas sudah dapat mempengaruhi kondisi kepadatan yang ada, selain itu kondisi kepadatan lalu lintas sudah dapat divisualisasikan dengan warna hijau untuk lancar hingga merah untuk padat.

4.3 Pengujian Skenario

Jaringan jalan yang sudah berbentuk *graf* dapat digunakan untuk mencari jalan terdekat antar *node* nya, sehingga *pathfinding* dapat mencari jalan tercepat sesuai dengan keadaan lalu lintas yang ada, misalnya jika suatu jalan terlalu padat, maka hasil dari *pathfinding* akan memberikan jalur tercepat alternatif.

4.3.1 Skenario 1

Pada skenario pertama ini akan dilakukan pengujian pada area alun-alun sidoarjo, pada skenario ini lalu lintas akan direkayasa sehingga akan terjadi kepadatan pada jalan di depan alun-alun sidoarjo, dan pencarian arah dilakukan dengan start dari arah surabaya dengan tujuan melewati alun-alun sidoarjo



Gambar 4.10: Target dan Tujuan

Pada gambar 4.10 menunjukan titik awal dan titik tujuan untuk dicari jalan terpendeknya, pencarian dilakukan dua kali, yaitu ketika jalan masih sepi, dan ketika jalan sudah padat,

Gambar 4.11: Garis kuning menunjukan hasil pencarian jalan

Setelah dilakukan pencarian jalan pada kondisi lancar, maka hasilnya akan menjadi seperti pada gambar 4.11, hasil pencarian jalan terpendek ditandai dengan garis berwarna kuning.

Gambar 4.12: Garis kuning menunjukan hasil pencarian jalan

Beberapa saat kemudian, setelah kondisi jalan telah berubah, maka hasil pencarian jalan terpendek pun ikut berubah, karena *pathfinding* yang diterapkan menghitung jalur tercepat bukan jalur terpendek, sedangkan pada kondisi di jalan raya terkadang jalur terpendek bukan lah jalur yang tercepat, terutama jika terjadi kemacetan yang cukup panjang.

Tabel 4.1: Waktu tempuh skenario 1

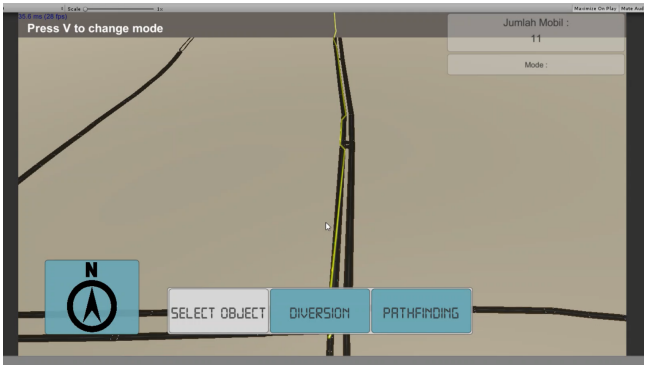
Jalur Pertama	
Kendaraan	**Waktu Tempuh**
0	91 Detik
64	188 Detik

Jalur Kedua	
Kendaraan	**Waktu Tempuh**
0	173.5 Detik

Pada tabel 4.1 menunjukan pada jalur pertama memiliki waktu tempuh selama 91 detik pada keadaan sepi, dan ketika waktu tempuh bertambah hingga 188 detik karena lalu lintas yang ada *pathfinding* akan memperoleh hasil yang berbeda, pada jalur kedua waktu tempuh yang diperlukan selama 173.5 detik lebih cepat dari jalur pertama.

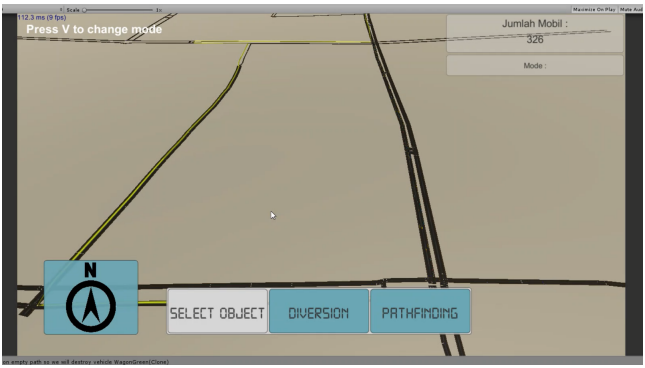
4.3.2 Skenario 2

Pada skenario kedua pengujian dilakukan dengan memsimulasikan sistem lalu lintas yang ada pada daerah merr hingga galaxy mall, metode yang digunakan sama seperti skenario pertama yang berbeda hanyalah tempatnya saja.



Gambar 4.13: Garis kuning menunjukan hasil pencarian jalan

Pada gambar 4.13 diperlihatkan jalur pertama ketika keadaan sepi, jalan yang dipilih langsung lurus ke arah tujuan dimana ini memang jalan tercepat dan terpendek yang bisa dilewati.



Gambar 4.14: Jalur tercepat yang ditampilkan berbeda dari gambar 4.13

Ketika jalur pada 4.13 sudah terlalu padat maka jalur pada gambar 4.14 yang ditampilkan, jalur ini bukan yang terpendek namun memiliki waktu yang tercepat untuk dilewati.

Tabel 4.2: Waktu tempuh skenario 2

Jalur Pertama	
Kendaraan	**Waktu Tempuh**
0	211 Detik
204	719 Detik

Jalur Kedua	
Kendaraan	**Waktu Tempuh**
0	501 Detik

Hasil pengujian dari skenario kedua dapat dilihat pada tabel 4.2, sama seperti skenario pertama, jalur kedua menjadi lebih cepat ketika jalur pertama dilewati oleh kendaraan yang terlalu banyak yang mengakibatkan kepadatan.

4.4 Pengujian performa

Pada pengujian ini, akan dihitung seberapa baik simulasi ini berjalan, sebuah program simulasi lalu lintas tentunya harus dapat memvisualisasikan banyak kendaraan sekaligus, beserta logikanya, selain kendaraan, logika lampu lalu lintas juga harus dijalankan, selain itu area yang luas juga dapat memberikan beban kepada sistem, pada pengujian kali ini bertujuan untuk menentukan apakah simu-

lasi sudah dapat berjalan dengan baik, dan jika tidak dapat dicari pada bagian mana simulasi terlalu banyak menggunakan resource.

4.4.1 Pengujian tahap pertama

Pada tahap ini pengujian dilakukan untuk mengetahui seberapa besar resource GPU yang diperlukan untuk melakukan *rendering* seluruh objek yang ada, jika pada simulasi terdapat 500 mobil, maka GPU harus memproses model semua 500 mobil tersebut, selain mobil GPU juga harus menggambar objek lainnya seperti jalan, dan lampu lalu lintas, optimisasi seperti *occusion occuling* sudah diterapkan saat pengambilan data, sehingga dapat menghasilkan performa yang optimal.

Tabel 4.3: Spesifikasi

Perangkat	**Spesifikasi**	
	A	B
OS	Windows 10 64 bit	
CPU	Intel Core i7 3630QM 2.4GHz - 3.2Ghz (Boost)	
GPU	Nvidia Geforce GT650m	Intel HD 4000
RAM	8GB	

Untuk menghindari pengaruh perbedaan spesifikasi yang lain, maka spesifikasi yang tidak termasuk dalam pengujian tidak berbeda, sehingga performa tidak terpengaruh oleh komponen yang lain selain yang diujikan, pada tabel 4.3 bisa dilihat jika satu-satunya yang berbeda adalah spesifikasi GPU, yang lainnya tetap sama.

Metode pengujian yang dilakukan program akan dibiarkan berjalan hingga simulasi menjalankan lebih dari 500 kendaraan, setiap 50 kendaraan yang sudah disimulasikan data *frame per seconds* akan disimpan, dan akan dibandingkan hasilnya antara 2 spesifikasi yang diujikan.

Tabel 4.4: Hasil tes tahap 1

Jumlah Kendaraan	**FPS**	
	Geforce GT 650m	**Intel HD 4000**
0	40	12
50	30	11
100	26	11
150	23	11
200	22	11
250	20	11
300	20	10
350	17	10
400	16	10
450	15	10
500	14	10

Berdasarkan data pada tabel 4.4, penggunaan GPU yang lebih cepat dapat mempercepat proses rendering dengan peningkatan *frame per second* yang cukup signifikan, namun seeringnya dengan bertambah jumlah kendaraan fps yang dihasilkan menurun sangat drastis, sedangkan pada GPU yang lebih lemah penurunan fps saat bertambahnya jumlah kendaraan tidak terlalu berpengaruh.

4.4.2 Pengujian tahap kedua

Pada tahap ini pengujian dilakukan untuk mengetahui seberapa besar resource CPU yang diperlukan untuk memproses seluruh script yang ada, seperti *AI* milik kendaraan, *AI* Lampu lalu lintas, dan sebagainya. Dalam simulasi dapat banyak kendaraan yang melakukan kalkulasi secara bersamaan, maka dari itu pengujian kali ini dilakukan untuk mengetes seberapa besar resource CPU yang diperlukan.

Tabel 4.5: Spesifikasi

Perangkat	**Spesifikasi**	
	A	B
OS	Windows 10 64 bit	
CPU	Intel Core i7 3630QM	
	2.4GHz - 3.2Ghz (Boost)	1.19Ghz
GPU	Nvidia Geforce GT650m	
RAM	8GB	

Untuk mendapatkan kecepatan processor seperti spesifikasi B pada tabel 4.5 dapat menggunakan fitur *power saver* sehingga kecepatan *clock* akan menjadi lebih rendah untuk mengehemat daya.

Tabel 4.6: Hasil tes tahap 2

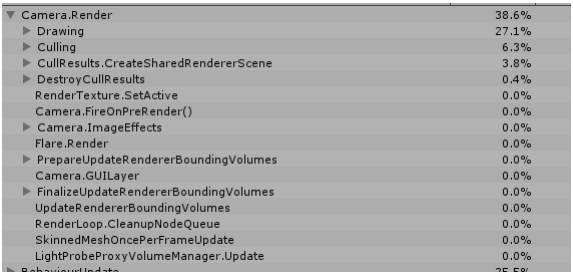
Jumlah Kendaraan	**FPS**	
	2.4 - 3.2 Ghz	1.19 Ghz
0	40	22
50	30	18
100	26	17
150	23	9
200	22	7
250	20	6
300	20	5
350	17	5
400	16	4
450	15	3
500	14	3

Pada hasil pengujian tahap 2, dari tabel 4.6 hasil *frame per second* yang dihasilkan lebih baik pada processor dengan *clock speed* yang lebih tinggi, keduanya sama-sama menggunakan gpu yang lebih kuat sehingga pada saat awal simulasi berjalan ketika kendaraan yang ada belum terlalu banyak fps yang dihasilkan tergolong cukup baik, bahkan jika dibandingkan dengan hasil tes pertama pada 4.4, processor dengan kecepatan yang lebih lambat jika dipasangkan dengan GPU yang cukup baik fps yang dihasilkan lebih

baik dibandingkan dengan processor dengan kecepatan tinggi dipasangkan dengan GPU yang lebih lemah.

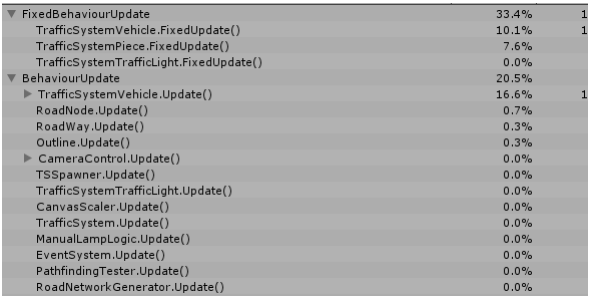
4.4.3 Pengujian dengan profiler

Profiler merupakan salah satu fitur milik Unity *Engine* yang berguna untuk melakukan monitoring terhadap proses yang berjalan, setiap proses yang berjalan dapat diketahui seberapa banyak proses tersebut menggunakan resource dari komputer yang digunakan.



Gambar 4.15: Tampilan *Profiler*

Pada awal program berjalan, berdasarkan pada gambar 4.15 proses dengan penggunaan *resource* terbanyak adalah proses *draw* atau *render*, dimana hal ini sangat sesuai dengan pengujian tahap pertama dan kedua, dimana penggunaan GPU dapat meningkatkan FPS secara sigifikan.



Gambar 4.16: Tampilan *Profiler* 30 Menit Kemudian

Gambar 4.16 menunjukan proses terberat yang terbaca oleh

profiler, dapat dilihat pada gambar tersebut proses yang mengatur *behavior* menggunakan *resource* yang cukup besar, yang menunjukan jika pada saat itu terjadi *resource* GPU yang kuat menjadi tidak terlalu penting, karena terdapat proses yang lebih berat, hal ini sesuai dengan hasil 4.6 dimana pada saat bertambahnya jumlah kendaraan yang ada, spesifikasi dengan GPU kecepatan rendah tidak berpengaruh banyak, hal ini menunjukan jika kecepatan CPU sangat berpengaruh pada proses ini.

BAB 5
PENUTUP

5.1 Kesimpulan

Dalam pengembangan simulasi ini telah diterapkan banyak sekali hal, mulai dari penggunaan *OpenStreetMap* sebagai data untuk membangun jalan, penggunaan OSMFilter untuk menyaring data yang didapat, penambahan fitur graf pada jaringan jalan yang telah di*generate* secara otomatis, indikator yang menunjukan jika suatu jalan telah terlalu padat atau sedang lengang,

Pada simulasi ini, pengaturan lampu lalu lintas, dan aturan lainnya berhasil meununjukan hubungan antara satu dengan yang lain. Simulasi ini berhasil menunjukan jika pengaturan lampu lalu lintas terlalu lama, atau terlalu cepat dapat menyebabkan kemacetan atau bahkan terjadi kondisi lalu lintas yang mengunci, selain itu penggunaan data dari *OpenStreetMap* semakin mudah untuk memvisualisasikan kondisi suatu jalan yang ada pada dunia nyata, diterapkannya fitur *pathfinding* juga dapat membantu pengguna untuk mencari atau menyediakan jalur alternatif jika jalan utama sudah terlalu ramai atau macet.

Namun *AI* yang disediakan oleh *framework Traffic Simulation* yang di kembangkan oleh *WiredDevelopment* sangat memakan resource terutama kinerja CPU, sehingga akan memberikan dampak pada simulasi skala yang lebih besar, bahkan optimisasi seperti *Occusion Occuling* tidak memberikan dampak yang begitu besar pada performa.

Selain itu data yang ada pada *OpenStreetMap* kurang begitu lengkap, sehingga masih perlu penyesuaian manual yang harus dikerjakan, terutama pada lokasi persimpangan, namun berkat yang sifatnya *Open Data* penggunaan data *OpenStreetMap* dapat dioptimisasi karena data yang di*export* dapat diubah-ubah sesuai dengan keinginan kita.

5.2 Penelitian Lanjutan

Penelitian selanjutnya diharapkan dapat meningkatkan performa simulasi yang dijalankan, terutama performa *AI* kendaraan yang perlu di rombak ulang, sehingga tidak terlalu memakan banyak *re-*

source, selain itu simulasi ini dibuat agar bisa diterapkan bersamaan dengan penelitian yang lain, misalnya penelitian tentang pengaturan otomatis lampu lalu lintas, dan penelitain lain yang berhubungan dengan sistem lalu lintas lainnya, dengan adanya simulasi ini penelitian tidak harus dilakukan di dunia nyata yang bisa menganggu lalu lintas.

DAFTAR PUSTAKA

[1] "domestik auto market and production (2017)." https://www.gaikindo.or.id/domestic-auto-market-production-2017/. Accessed: 2017-12-30. (Dikutip pada halaman 1).

[2] "penyebab kemacetan di jalan raya." http://nationalgeographic.co.id/berita/2017/11/ternyata-ini-penyebab-kemacetan-tanpa-alasan-yang-serin Accessed: 2017-11-15. (Dikutip pada halaman 1).

[3] M. said El Hmam, "Macro-micro simulation of traffic flow," 2006. (Dikutip pada halaman 8).

[4] H. F. Halauoi, "Smart traffic online system (stos): Presenting road networks with time-weighted graphs," 2010. (Dikutip pada halaman 10).

Halaman ini sengaja dikosongkan

BIOGRAFI PENULIS

Bagus Himawan, lahir pada 05 Desember 1995 di Banjarmasin, Kalimantan Selatan. Penulis menempuh pendidikan S1 Departemen Teknik Komputer bidang studi *Game and Mobile Apps* Fakultas Teknologi Elektro ITS.Penulis sudah mulai mengenal dunia programming sejak kelas 3 SMP, dan waktu SMA berhasil menjuarai beberapa lomba *competitive programming*. Setelah lulus SMA penulis melanjutkan studi di Teknik Multimedia dan Jaringan yang sekarang sudah berubah nama menjadi Teknik Komputer, pada tahun 2014 penulis aktif dalam UKM perfilman CLICK ITS, Sejak tahun 2016 penulis merupakan anggota aktif Lab-B201Crew dan menjabat sebagai Admin unyum Laboratiorium pada tahun 2017-2018.Pada tahun 2016 - 2017 penulis menjadi panitia acara *Global Game Jam 2017* di Surabaya. Penulis juga mendirikan sebuah *startup game studio* bernama Calcatz pada tahun 2015. Penulis menghabiskan banyak waktu untuk mendalami pengambangan *game*, *game design*, pengembanan aplikasi android, dan *physics* untuk kendaraan pada sebuah game.

Halaman ini sengaja dikosongkan