Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS
EMPRESARIALES

FLUJO VEHICULAR SOBRE SISTEMA DE TRÁFICO COMPUTARIZADO

PROYECTO DE TÍTULO PARA OPTAR AL GRADO DE INGENIERÍA CIVIL EN INFORMÁTICA

Por **Jorge Paz Pavez**
Bajo la guía de **Alejandra Segura Navarrete**

Concepción, Chile
Septiembre 2020

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Resumen

El tráfico vehicular forma parte de nuestras vidas desde hace tiempo, conforme pasa el tiempo se ha hecho relativamente más accesible el adquirir un vehículo, lo que se ha notado por las calles de Concepción y el resto de Chile. Lo que no vemos es precisamente cómo responde la infraestructura vial frente a este aumento en la población vehicular, que se traduce finalmente en flujos entrantes y salientes de tráfico en términos matemáticos. Hoy por hoy, los avances tecnológicos actuales permiten describir fenómenos reales como el tráfico, como miembros de la sociedad y al alcance de todas estas tecnologías surge la pregunta, ¿podemos utilizar las tecnologías para analizar los distintos flujos en una estructura vial?

El objetivo central de este estudio consiste en construir un modelo que represente el fenómeno del tráfico, y mediante éste calcular algunos índices que determinan el comportamiento del flujo vehicular en sí mismo.

Para poder llevar a cabo esto, se estudian las nociones de las dinámicas de tráfico, que permiten establecer y entender los índices que se calcularán en el proyecto. De igual forma, para obtener los datos se utiliza un banco de datos conocido como OSM que entrega los datos en un fichero XML, entonces se define cómo se procesan estos datos a modo de construir una zona. Luego, se establece cómo participa la inteligencia artificial sobre este entorno donde delimitamos el comportamiento particular de los vehículos y cómo procesan su toma de decisiones.

Por último estudiamos y determinamos los aspectos y elementos que condicionarán el flujo de tráfico, como los semáforos, las entradas y salidas del mapa.

Veremos finalmente cómo se desempeña una cierta estructura vial en términos de flujo vehicular y cómo los distintos flujos entrantes alteran la representación de un tráfico libre o congestionado.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Abstract

The vehicular traffic forms part of our lives since a while now, as time goes on it has been relatively easier to acquire a vehicle, which can be noticed over the streets of Concepción and the rest of Chile. What we don't see is precisely how does the road infraestructure answer to this increase on the vehicular population, which is finally translated on a incoming and outgoing flow in mathematical terms. As today the actual technological advances allow to describe real phenomena such as traffic, as members of society and within reach of all these technologies comes the question, can we utilize the technologies in order to analyze the different flows on a road structure?

The central objective of this study consist of building a model that represents the traffic phenomenon, and through it calculate some indexes that determine the behaviour of the traffic flow in itself. In order to do this, we study the dynamics of traffic that will allow us to stablish and understand the indexes that we will be calculating on the project.

Similarly, in order to obtain the data we utilize an open-source database known as OpenStreetMap which will give us the data as a XML file, then we define how are these data files processed to build a zone.Then, we establish how does the artificial intelligence participates over this environment where we delimit the particular behaviour of the vehicles and how do they make decisions.

By the end we study and determine the aspects and elements that will condition the traffic flow, such as traffic lights, entrances and exits of the map.

We will finally see how does a certain road structure perfom in terms of traffic flow and how does the different incoming flows alter the representation of a free flow or congested one.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Índice

MOTIVACIÓN 10
INTRODUCCIÓN 11

CAPÍTULO: 1 DEFINICIÓN DEL PROYECTO 12

1.1.- CONTEXTO DEL PROBLEMA 12
1.2.- OBJETIVOS 13

1.2.1.- Objetivo general 13
1.2.2.- Objetivos específicos 13

1.3.- SUPUESTOS, RESTRICCIONES Y LÍMITES 13
1.4.- AMBIENTE DE INGENIERÍA 14

1.4.1.- Hardware 14
1.4.2.- Software 14

1.5.- CONTRIBUCIÓN DEL PROYECTO 14

CAPÍTULO: 2 MARCO CONCEPTUAL- DINÁMICAS DEL FLUJO DE TRÁFICO 15

2.1.- INTRODUCCIÓN A LAS DINÁMICAS DEL FLUJO DE TRÁFICO VEHICULAR 15
2.2.- MODELO MICROSCÓPICO DEL FLUJO VEHICULAR 16
2.3.- MODELO MACROSCÓPICO DEL FLUJO VEHICULAR 17

2.3.1.- Flujo de tráfico 17
2.3.2.- Densidad vehicular 18
2.3.3.- Tiempo medio de viaje 18
2.3.4.- Motores Gráficos 18

2.3.4.1.- Blender 19
2.3.4.2.- Unreal Engine 19
2.3.4.3.- Unity3D 19

CAPÍTULO: 3 CONCEPTUALIZACIÓN Y CONSTRUCCIÓN DEL ENTORNO 21

3.1.- PROCESO DE CREACIÓN DEL MODELO 21
3.2.- CONCEPTUALIZACIÓN DE LA CIUDAD 22
3.3.- OPENSTREETMAP 22

3.3.1.- Formato de datos 23

3.4.- SERIALIZACIÓN DE DATOS OSM 25
3.5.- SERIALIZACIÓN XML EN C# 27

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

3.6.- DIAGRAMA DE CLASES 28
3.7.- CONSTRUCCIÓN DEL ENTORNO 31

3.7.1.- Proyección de Mercator 31
3.7.2.- Construcción procedural de vías 31
3.7.3.- Generación procedural de Meshes / Mallas 32
3.7.4.- Generación procedural de grafo que contiene estructura vial cargada 34

CAPÍTULO: 4 INTELIGENCIA ARTIFICIAL COMO AGENTES MICROSCÓPICOS DEL TRÁNSITO 36

4.1.- COMPORTAMIENTO DESEADO 36
4.2.- IMPLEMENTACIÓN INTELIGENCIA ARTIFICIAL 36

4.2.1.- Desplazamiento y detención 38
4.2.2.- Sistema de carriles 40
4.2.3.- Detección de colisiones 43
4.2.4.- Agentes y velocidades 45

CAPÍTULO: 5 SISTEMA DE TRÁFICO 47

5.1.- SEMÁFOROS 47
5.2.- ENTRADAS 50
5.3.- SALIDAS 52
5.4.- FLUJO 52
5.5.- DENSIDAD 53
5.6.- PRUEBAS 54

5.6.1.- Paicaví - O'higgins 55
5.6.2.- Paicaví – Carrera 60
5.6.3.- Conclusiones finales 64

REFERENCIAS 65
ANEXOS: GLOSARIO 66
ANEXOS: CLASES PRINCIPALES IMPLEMENTADAS 67

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Índice de Ilustraciones

Ilustración 1: Fenómeno de tráfico en una carretera....16
Ilustración 2: Distintos niveles de flujo estimados en investigación....17
Ilustración 3: Interfaz de Unity3D, en celeste la jerarquía en escenario. En verde el explorador del proyecto. La zona azul describe cada elemento o ítem que posee un objeto en escena, y el recuadro rojo permite pre-visualizar la escena....20
Ilustración 4: Secciones de estudio para el proyecto que comprende la Rotonda Paicaví – Carrera y la intersección de O'Higgins con Paicaví....22
Ilustración 5: Formato de datos OSM....24
Ilustración 6: Proceso de la serialización de un objeto....25
Ilustración 7: Funciones que serializan en objetos de clase distintos elementos del archivo OSM/XML....27
Ilustración 8: Sector de prueba Paicaví-Carrera. A la izquierda los datos representados por OSM y a la derecha los mismos datos luego de ser serializados visualizados en Unity....28
Ilustración 9: Diagrama de clases que componen la serialización XML a objetos de clase 28
Ilustración 10: Clase BaseOsm, con el único método de obtener valores de claves de una colección de atributos XML....29
Ilustración 11: Clase OsmBounds con sus distintas variables de acceso público....30
Ilustración 12: se agregan atributos a nodos que tengan la clave highway....30
Ilustración 13: Distintas mallas poligonales representando una misma figura, a la izquierda representada utilizando triángulos y en la segunda cuadrados....32
Ilustración 14: Texturas utilizadas para el proyecto....33
Ilustración 15: Ejemplo de generación procedural de mallas poligonales....33
Ilustración 16: Meshes de Paicaví – Carrera generadas....34
Ilustración 17: Primeros intentos de generar de manera procedural el grafo, en amarillo las vías que componen el set de datos....34
Ilustración 18: Función que determina si un punto ya existe en una posición....35
Ilustración 19: Resultado final de generación procedural de grafos a partir de un archivo osm....35
Ilustración 20: Algoritmo general que define la toma de decisiones de los agentes vehiculares....37
Ilustración 21: Agente vehicular....38
Ilustración 22: Sistema de detección de obstáculos....38
Ilustración 23: Perfil de rendimiento con alrededor de 100 vehículos....39
Ilustración 24: Función de traslación simple....40
Ilustración 25: Vehículo agente y su colisionador en verde....40
Ilustración 26: Perfil de rendimiento con enfoque simplista....40
Ilustración 27: Como podemos observar en la imagen, el bus mantiene una distancia (en rosado) con el centro....41
Ilustración 28: Función utilizada para calcular el punto más cercano de un punto a una recta AB....42
Ilustración 29: Pistas hasta 3 carriles....42

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Índice de Tablas

Tabla 1: Espectro de áreas que comprende el estudio de tráfico vehicular 15
Tabla 2: Atributos y sus claves utilizados para filtrar la información que se utiliza en el proyecto 25
Tabla 3: Modelos 3D y sus respectivas velocidades 45

Índice de Ecuaciones

Ilustración 30: El agente tiende a perder el rumbo cuando se desplaza a una velocidad muy alta, o existe una tasa de FPS baja 43
Ilustración 31: Demostración de vehículos detenidos a cierta distancia del siguiente vehículo o intersección 44
Ilustración 32: Proyección de rayo inmediata mediante la función Raycast() de la clase Physics 45
Ilustración 33: Semáforos como objeto 47
Ilustración 34: Ciclos de un semáforo, en la imagen se aprecian 2 semáforos 48
Ilustración 35: Script que se controla los ciclos de semáforos, recibe un parametro Cycle que define el temporizador del objeto 48
Ilustración 36: Gameobject que genera autos cada cierto tiempo 50
Ilustración 37: Corrutina que genera vehículos cada cierto tiempo. Es utilizada por cada carril que tenga una pista 51
Ilustración 38: Flujo de salida que destruye vehículos y estima tiempos de viaje 52
Ilustración 39: Registradores de flujo de tráfico: consisten en secciones transversales a la pista (en amarillo y rojo) 53
Ilustración 40: Mesh de una vía seleccionada 54
Ilustración 41: Distintas etapas con sus respectivos ciclos T para las pruebas 54
Ilustración 42: Grapher y registro de datos 55
Ilustración 43: Distintos niveles de tráfico observados en intersección 55
Ilustración 44: Perfil de rendimiento para intersección O'Higgins - Paicaví. En naranja uso de motor físico 56
Ilustración 45: Flujo del sistema en distintos escenarios para O'Higgins-Paicaví 57
Ilustración 46: Gráfico de densidad promedio por cuadro durante la prueba 58
Ilustración 47: Gráfico que detalla tiempos de viaje por vehículo en la prueba 59
Ilustración 48: Representación área total que comprende la zona 59
Ilustración 49: Perfiles de rendimiento para escenario Carrera - Paicaví frente a distintos ciclos T. En naranja uso del procesador de las físicas de los agentes. 60
Ilustración 50: Distintos niveles de tráfico observados para esta sección 61
Ilustración 51: Gráfica de flujo durante la prueba 61
Ilustración 52: Densidad durante la prueba para este escenario 62
Ilustración 53: Área total que ocupa esta sección 63
Ilustración 54: Gráfico del tiempo de viaje para cada vehículo en escenario de rotonda Paicaví 63

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Ecuación (1) Flujo de tráfico vehicular
Ecuación (2) Densidad Vehicular
Ecuación (3) Proyección de Mercator
Ecuación (4) Velocidad Representativa
Ecuación (5) Ciclo de semáforos
Ecuación (6) Ciclo de aparición vehicular

Motivación

Las calles urbanas de Concepción han sufrido muchos cambios en el último tiempo; hasta hace poco el control de circulación en las calles era bajo o nulo debido al contexto social de fines del año 2019, podíamos ver calles importantes funcionando sin semáforos, a lo que se le suman factores que ya incidían en el flujo vehicular como inhabilitación de calles por las reparaciones en progreso y un mal estado general de las mismas.

Hoy por hoy, con el nuevo contexto de crisis sanitaria, fue necesario aumentar el control de la circulación con los cordones sanitarios y cuarentenas, lo que aumentó en el tráfico y el tiempo de espera para cruzar de un lugar a otro.

Como vemos, el flujo vehicular varía mucho en el transcurso del tiempo, ya sea por las condiciones anteriores o bien por otros factores más complejos como la naturaleza humana. Y frente a las continuas cifras al alza del mercado automotor chileno y el consiguiente aumento de la plaza automotriz de la ciudad, nos vemos en la necesidad de afrontar nuevos desafíos de movilidad urbana de modo que en un futuro se puedan tomar decisiones que afecten de manera positiva a la infraestructura vial penquista.

Es por esto último que, como estudiante de ciencias de la computación y miembro de la comunidad penquista, sentí gran interés por explorar la contribución de las tecnologías en áreas de la ingeniería civil donde se han hecho estudios relativos al tráfico.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Introducción

Concepción es la segunda ciudad más grande de Chile, sin embargo, la ciudad no responde bien al alto flujo vehicular debido a su infraestructura vial. Conforme aumenta la plaza automotriz de la urbe, el problema se hace más insostenible y por ende la calidad de vida de la ciudad disminuye. Pero ¿se puede aprovechar de mejor manera la infraestructura actual? Bastaría con dirigir de manera dinámica el flujo de tránsito por la ciudad. Es esta interrogante combinada con la necesidad de aportar de alguna manera a la comunidad penquista, ofreciendo alternativas o quizás simplemente una experiencia que pueda contribuir a la problemática vehicular que se vive en la ciudad.

En el presente trabajo de tesis, se realizará un sistema de tráfico simple, mediante el cual podremos observar ciertos indicadores de tránsito. A continuación, se detalla el contenido de cada capítulo:

En el primer capítulo, definiremos el proyecto estableciendo objetivos, supuestos y limitaciones. Para el segundo capítulo abordaremos las dinámicas del flujo y cómo éstas se condicen para determinar indicadores como el flujo de tráfico y tiempos promedios de viaje en un segmento.

En el tercer capítulo, profundizaremos en el entorno en el que nuestro estudio y simulación se llevarán a cabo. ¿Cómo obtendremos los datos del mapa? ¿Cómo se construye el grafo que representará las calles de la zona a estudiar? Son algunas de las preguntas que se responden en dicho inciso.

Una vez que el entorno haya sido definido, podremos trabajar en la inteligencia artificial de la cual se compondrá la simulación. Veremos cómo los distintos vehículos toman decisiones en tiempo real en base al entorno mismo, y cómo manejaremos las físicas individuales o microscópicas vehiculares. Para estos tópicos tendremos el cuarto capítulo como referencia.

En el quinto y último capítulo, y en base a los asuntos anteriores, explicaremos el conjunto que compone el sistema de tráfico del presente proyecto. ¿Cómo interactúan los vehículos entre sí? ¿Cómo se desempeña la simulación? Además, seremos capaces de poder cuantificar indicadores de un sistema de tráfico, los cuales analizamos tras la recopilación de los mismos, y estableceremos conclusiones finales del estudio en sí.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Capítulo: 1 Definición del Proyecto

1.1.- Contexto del Problema

Para mantener un flujo de tráfico en movimiento en tiempos como los de ahora frente a un tráfico en ascenso y de recursos limitados, la ciencia se ve desafiada para encontrar soluciones innovadoras a estas problemáticas.

Contribuciones de distintos profesionales (físicos, ingenieros, matemáticos y psicólogos de comportamiento) han llevado a un mejor entendimiento del comportamiento de conductores y el flujo vehicular de tráfico.
El flujo vehicular es dinámico y por lo tanto, no existe un modelo matemático específico que sea cien por ciento fiel a la realidad, sino más bien aproximaciones de ésta que contribuyen a un mejor entendimiento de nuestra forma a la hora de movilizarnos. Es aquí donde entra el concepto de **modelado científico,** que tiene como fin construir un componente particular o característica del mundo real de modo que sea más fácil de entender, definir, cuantificar, visualizar o simular mediante la referencia a conocimientos de aceptación común. Como veremos más adelante, se requiere seleccionar e identificar aspectos relevantes de una situación para poder usar en distintos propósitos, tales como una simulación conceptual para entender y visualizar de mejor manera la materia en cuestión.

"… las ciencias no intentan explicar, difícilmente intentan interpretar, principalmente construyen modelos. Se entiende por modelo un constructo matemático el cual, con la adición de ciertas interpretaciones verbales, describe un fenómeno observado. La justificación de tal constructo matemático es solamente y precisamente que se espera funcione - esto es, describir de manera correcta un fenómeno desde una área razonablemente amplia" [1]

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

1.2.- Objetivos

1.2.1.- Objetivo general

Construir un modelo que describa el fenómeno de tráfico y sus distintas variables en un sistema computarizado que simula el flujo vehicular.

1.2.2.- Objetivos específicos

- Estudiar las dinámicas del flujo de tráfico y analizar sus distintas áreas de estudio para ser integradas en el estudio
- Identificar indicadores de estudio relativas al flujo vehicular que determinan el comportamiento de flujo vehicular
- Definir motor gráfico a utilizar en la construcción de simulación de tráfico
- Identificar fuente de datos cartográficos y estudiar su estructura
- Analizar proceso de serialización de datos para
- Especificar procedimientos de generación procedural
- Diseñar e implementar una inteligencia artificial
- Implementar elementos que componen un sistema de tráfico
- Analizar pruebas realizadas en el simulador

1.3.- Supuestos, Restricciones y Límites

Para la ejecución correcta del proyecto se supone lo siguiente:

- El flujo de tráfico es una representación del fenómeno real y bajo ninguna circunstancia una representación del flujo en un instante real

Como restricciones, se tienen:

- No existe un presupuesto para el proyecto, por lo que todo material en uso debe ser gratuito.
- Se debe cumplir con el plazo establecido por el límite de tiempo de la actividad de titulación de la carrera Ingeniería Civil en Informática sede Concepción.
- Se realizan modificaciones sobre el volumen de datos investigados, para obtener tiempos de respuesta acordes a la máquina sobre la cual se desarrolla.

Por último, como limitación tenemos que:

- Debido a la complejidad que supone el proyecto para una sola persona, se establece como entregable un sistema capaz de medir el flujo, densidad y tiempo de viaje promedios de una escena.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

- Se limita la simulación sólo a vehículos, pistas vehiculares y edificios, siendo estos últimos sólo como artefacto cosmético.

1.4.- Ambiente de Ingeniería

1.4.1.- Hardware

Para no tener problemas con el consumo de memoria, procesador y de gráficos a la hora de simular varios agentes, es necesario contar con un equipo potente en términos computacionales. Vale decir que a lo largo del proyecto, se realizan cambios sobre algunos aspectos para obtener comportamientos deseados acordes a la máquina.
Las especificaciones técnicas de la máquina son las que siguen:

- Procesador 3.7 GHz Intel i7-8700K
- 16 GB de memoria RAM
- Tarjeta gráfica 6GB NVIDIA GTX 1660
- SSD 250 GB de memoria

1.4.2.- Software

En el transcurso de la investigación se utilizaron diversas aplicaciones que contribuyeron de manera positiva al proyecto, ya sea desde la información que entregan como las herramientas que proveen.

- Unity3D – utilizado para describir la simulación de tráfico
- Microsoft Visual Studio Community Edition 2019 – usado para escribir los distintos scripts del proyecto.
- Microsoft Excel – utilizado para describir los resultados de prueba en gráficos.
- JOSM – aplicación que permite editar archivos .osm, se utilizó para comprobar archivos descargados desde OSM.
- GIMP – aplicación utilizada para diseñar las texturas de las vías.

1.5.- Contribución del proyecto

En estricto rigor, la investigación contribuye al estudio del flujo vial en determinadas intersecciones, no se pretende solucionar los problemas asociados a un alto tráfico pero sí proponer una representación vista desde el lado computacional. Como línea futura, se espera que se pueda implementar aspectos que se dejaron fuera de este proyecto para poder cumplir con la restricción del tiempo.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Capítulo: 2 Marco Conceptual- Dinámicas del flujo de Tráfico

2.1.- Introducción a las dinámicas del flujo de tráfico vehicular

El flujo de tráfico, se comporta de una **manera compleja y no-lineal**, dependiendo de una gran cantidad de vehículos yendo en distintas direcciones, pero mediante el mismo flujo unidireccional. Dada la naturaleza humana de las reacciones individuales de cada conductor, los vehículos no se rigen por un modelo estricto o alguna ley mecánica que establezca el flujo, pero muestran una formación agrupada que en conjunto son llamados clústeres y **una propagación parecida a las de una onda** en las direcciones en las que el flujo se propaga, debido a los semáforos y un fenómeno particular llamado ondas de pare y siga.
Las ciencias de tráfico vehicular, se componen de distintas áreas de estudio que se diferencian en su escala de tiempo en el que ocurren. Lo último debido a que el tráfico ocurre desde una escala sub microscópica que comprende aspectos tales como el control de los frenos y motor de cada vehículo, hasta un campo que estudia a largo plazo cambios demográficos y socioeconómicos. Observemos a continuación la delimitación de las dinámicas de flujo vehicular, descrita en la siguiente tabla:

Tabla 1: Espectro de áreas que comprende el estudio de tráfico vehicular. Elaboración propia en base a [2]

Escala de tiempo	Campo	Modelos	Aspecto del tráfico (Ejemplos)
<0.1 s	Dinámicas vehiculares	Sub microscópico	Control de motor, frenos y dirección
1s			Tiempo de reacción, brechas de tiempo
10s	Dinámicas de flujo vehicular	Modelos de seguimiento vehicular	Aceleración y deceleración
1 min		Modelos Macroscópicos	Período cíclico de los semáforos
10 min			Ondas Pare-Y-Siga
1h			Hora Peak
1 día		Asignación bajo demanda de tráfico	Patrón de demanda diaria
1 año	Planificación de transporte		Construcciones, cambios en infraestructuras
5 años		Estadísticas	Estructura socioeconómica

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Como se observa en la tabla 1, el espectro es muy amplio en términos de áreas de estudio que en conjunto comprenden las ciencias de tráfico vehicular.
Es necesario acotar estas áreas a conceptos que nos son más útiles a la hora de construir un sistema de tráfico simple. Comprenderemos entonces, aspectos sub microscópicos hasta los aspectos macroscópicos para la construcción y estudio del presente proyecto.

2.2.- Modelo Microscópico del flujo vehicular

En el tráfico, cada *átomo* individual de nuestra fórmula viene a ser los vehículos motorizados que circulan por los carriles. Limitamos nuestra atención sólo a estos en su conjunto y cómo interactúan con el entorno, por lo que sólo un aspecto inherente al tráfico - la velocidad - es de utilidad. Aspectos individuales tales como el pensamiento de los conductores a la hora de tomar una decisión, funcionamiento de motor y de frenos, y consumo de combustible quedan fuera de lugar.

Ilustración 1: Fenómeno de tráfico en una carretera

Veremos más tarde en el capítulo 3 cómo es implementada la velocidad para cada vehículo y como éstos vehículos toman decisiones, notaremos además que cada vehículo tiene como rol principal ser parte del flujo vehicular y registrar el tiempo que existió en la escena de la simulación.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

2.3.- Modelo Macroscópico del flujo vehicular

Este modelo es - como podríamos suponer- un modelo matemático de tráfico que formula la relación entre características del flujo vehicular, tales como la densidad, indicador de flujo, la velocidad media de un flujo de tráfico, entre otras.

2.3.1.- Flujo de tráfico

El flujo vehicular, o flujo de tráfico es el fenómeno causado por el flujo de vehículos en una pista, vía, calle, autopista y carriles en general. Como hemos mencionado anteriormente presenta similitudes con otros fenómenos como el flujo de partículas, dado a su composición atómica o individual.
Por lo general, estudios o modelos matemáticos del flujo matemático asumen una cola vertical, en la cual los vehículos a lo largo del carril no retroceden ni se *derraman* a otros carriles.
El flujo de tráfico vehicular de una pista se expresa entonces de la siguiente manera

$$(1)\ q = \frac{n * 3600}{t} \quad \left(\frac{vehículos}{hora}\right)$$

donde

q = *flujo vehicular (número de vehículos por hora)*
n = *número de vehículos pasando por una sección transversal del segmento en t segundos*
t = *intervalo de tiempo para los vehículos circulantes (s)*

Ilustración 2: Distintos niveles de flujo estimados en investigación

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Este índice nos es muy útil a la hora de medir el comportamiento de segmentos viales, sin embargo dado a factores tales como el número , largo y ancho de las pistas y ciclos de hora punta en distintas ciudades, se hace volátil medir cuán eficaz es una infraestructura vial.
Diremos que cuando q > 2.000 veh/hora indica una pista muy concurrida y en casos donde el número de pistas es menor a 3 probablemente tengamos formada una congestión. Del mismo modo, un q ~ 1.000 veh/hora indica por lo general un flujo estable como muestra la ilustración 2.

2.3.2.- Densidad vehicular

La densidad vehicular es el número de vehículos por kilómetro en una pista, se puede expresar como

$$(2)\ k = (n * 1000)/L$$

en donde

- $k = \text{vehículos por kilómetro}$
- $n = \text{número de vehículos presentes en un largo } L \text{ de la pista}$
- $L = \text{el largo de una calle en metros.}$

2.3.3.- Tiempo medio de viaje

El tiempo medio de viaje nos indica cuál es el tiempo promedio que toma a los agentes vehiculares transitar por un área o segmentos. Combinado con los ciclos semafóricos, es útil para entender la prioridad de algunas pistas frente a otras. Para este cálculo, se establece que cada vehículo agente de la simulación empezará un cronómetro al momento de aparecer en escena, y dejará de contar hasta llegar al momento de desaparecer de la misma, de modo que mediante el promedio de todos estos contadores estimemos el tiempo que toma pasar por una sección del mapa.

2.3.4.- Motores Gráficos

A fin de implementar un simulador que nos permita observar y cuantificar estas variables, requerimos de un motor gráfico que se encargue de los aspectos físicos de nuestro proyecto, como el movimiento de vehículos o detección de colisiones como también de aspectos visuales que son relacionados a la representación (renderizado como anglicismo) de modelos poligonales. En el mercado, existen muchos motores gráficos tanto para ambientes bidimensionales, tridimensionales o ambos en un mismo motor. Para este proyecto, se consideraron 3 aplicaciones de escritorio que son útiles a la hora de describir un modelo.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

2.3.4.1.- Blender

Blender es una suite de creación de multimedia 3D gratuita y de código abierto. Integra un amplio espectro de herramientas esenciales como lo son modelados 3D, renderizado, animación, texturización y provee de distintas simulaciones microscópicas como simulación del agua, de los tejidos, de gelatina, de lluvia entre otros fenómenos físicos de la vida real. Permite además modificar cualquier aspecto gracias a que cuenta con una interface de programación (API) en el lenguaje Python que se traduce en la capacidad de agregar scripts en este lenguaje.

2.3.4.2.- Unreal Engine

Unreal Engine es un motor gráfico perteneciente a la compañía de videojuegos Epic Games, provee de una suite completa para la creación de videojuegos, visualización arquitectónica y automovilística, creación de contenido multimedia, simulación entre otros. Respecto a la licencia, Unreal Engine provee dos licencias gratuitas, una para aplicaciones remuneradas donde se debe pagar un 5% de las ganancias a Epic Games, y la segunda es 100% gratuita para proyectos libres.
Respecto a la posibilidad de scripting, Unreal Engine provee de una interface de programación en C++ por lo que es posible realizar o modificar cualquier objeto en escena mediante este lenguaje.

2.3.4.3.- Unity3D

Unity3D es un motor físico, gráfico que es muy usado en la creación de videojuegos y cómo no, de simuladores, posee versiones comunitarias que permiten a cualquiera utilizarlo y de pago que además de agregar funcionalidades, permite distribuir los productos bajo una licencia comercial. Facilita de sobremanera la creación de ambientes y posibilita la creación de cualquier requerimiento a través de objetos que interactúan en una escena, y cada objeto puede contener distintos tipos de atributos, como un script, modelo 3D, imagen, sonidos, entre otros.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

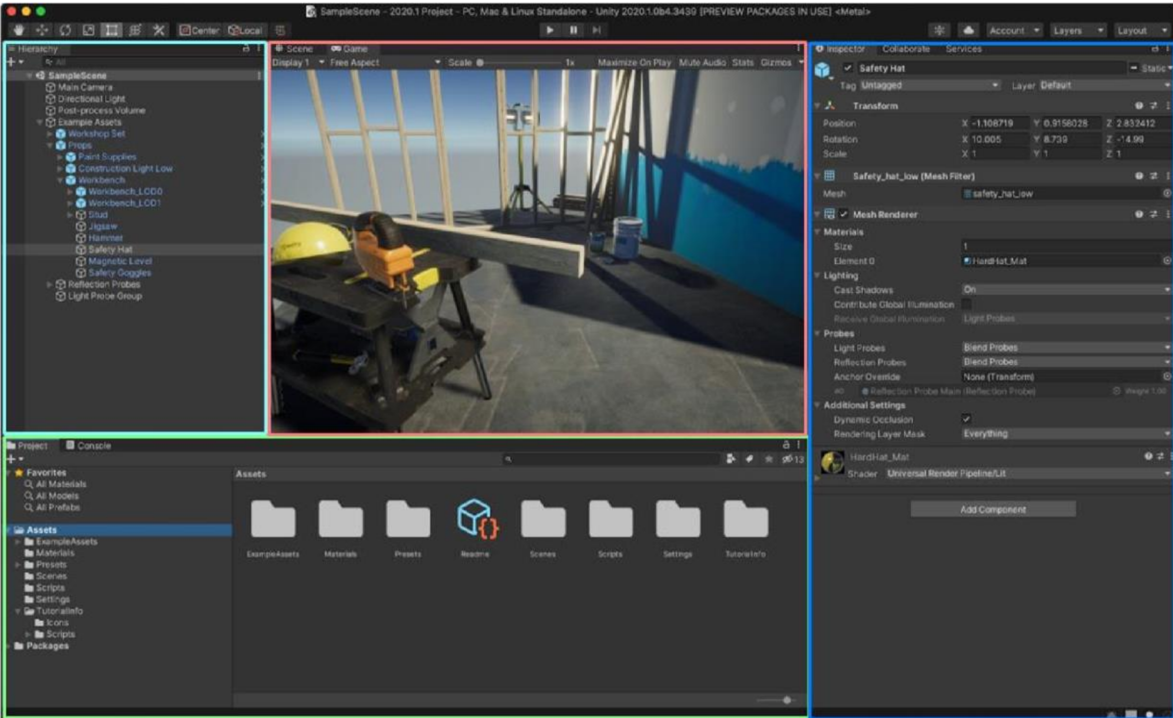

Ilustración 3: Interfaz de Unity3D, en celeste la jerarquía en escenario. En verde el explorador del proyecto. La zona azul describe cada elemento o ítem que posee un objeto en escena, y el recuadro rojo permite pre-visualizar la escena.

Unity3D, permite la creación de estos objetos (conocidos como *GameObjects*) y comportamientos mediante código, en este caso en el lenguaje de programación C#, aunque también es posible utilizar JavaScript para aplicaciones web.
Por esto, se decide usar Unity3D debido al conocimiento previo que se tiene de esta herramienta y del lenguaje C#.
En general los motores gráficos no deben suponer una diferencia de rendimiento o representación visual ya que estos sólo son un medio para modelar un sistema.
Vale la pena destacar que el entendimiento de álgebra lineal es muy importante para poder ser capaces de realizar distintos requerimientos, como veremos en el capítulo 3.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Capítulo: 3 Conceptualización y Construcción Del Entorno

Ahora que hemos podido interiorizar lo que el flujo de tráfico comprende en particular para este estudio, es necesario que podamos materializarlo y conforme a ello, asimilar los datos de la infraestructura vial que queramos estudiar.

> *"(...) Tenemos un sistema de transporte 2D y un ambiente de vida y trabajo 3D. Si vamos a un sistema de transporte 3D, seremos capaces de resolver el actual problema del tráfico vehicular. " Elon Musk* [3]

3.1.- Proceso de creación del modelo

Para materializar el sistema, se identificaron 3 elementos principales que constituyen el modelo, que son:

- Mapa o entorno
- Agentes Vehiculares (Inteligencia artificial)
- Reglas del sistema de tráfico

Como los vehículos son agentes que se desplazan por el mapa, se consideró que para iniciar el proyecto primero era necesario definir el entorno sobre el cual los agentes actúan. La lista de actividades para el primer tópico consiste de lo siguiente:

- Definir zona de estudio
- Definir fuente de datos
- Tratamiento/filtración de los datos
- Definir procedimientos que generan el entorno del proyecto (mallas poligonales y grafos)

Con respecto a la creación de los agentes vehiculares, se establecieron las siguientes actividades principales que darían forma a la inteligencia artificial.

- Establecer comportamiento esperado para el sistema
- Identificar pasos que ejecuta cada agente para decidir la siguiente acción
- Definir las acciones para los vehículos (avanzar y frenar)
- Definir procedimiento con el cual los vehículos hacen uso de la infraestructura vial generada.

En último lugar pero no menos importante se decidió abordar las reglas específicas del sistema que condicionan un correcto funcionamiento del tráfico y de los medidores de índices, para lo cual se realizaron las siguientes actividades:

- Establecer elementos condicionantes de un sistema de tráfico
- Definir cada uno de estos elementos para el modelo
- Definir cómo se registran los datos y cómo se procesan para cada índice de estudio

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

3.2.- Conceptualización de la ciudad

Debido que es necesario limitar la zona de estudio, se fija la comuna de Concepción como la zona de pruebas para nuestro proyecto. Como veremos más adelante en este mismo capítulo, las dimensiones de los lugares a observar variaron mucho, por lo que se estableció como zonas de prueba / estudio a calles o intersecciones que se consideran neurálgicas de la ciudad, las que son la rotonda Paicaví y la intersección entre O'Higgins y Paicaví como ilustra la siguiente imagen:

Ilustración 4: Secciones de estudio para el proyecto que comprende la Rotonda Paicaví – Carrera y la intersección de O'Higgins con Paicaví

3.3.- Openstreetmap

Openstreetmap, también conocido como OSM, es un proyecto colaborativo abierto en donde se crean mapas editables y de libre uso. Los mapas son creados mediante información geográfica capturada por dispositivos GPS móviles, orto fotografías y otras fuentes libres tales como la contribución de editores. Los datos proveídos por OSM, como los datos vectoriales, imágenes y la cartografía en general son distribuidos bajo la Licencia Abierta de Bases de Datos (ODbL por sus siglas en inglés).

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

3.3.1.- Formato de datos

OSM distribuye datos mediante archivos XML en el datum WGS84, que es un sistema geodésico (que divide tierras) y que contienen información útil como la longitud o latitud de un punto. Cada punto puede ser miembro de una relación superior de puntos, o nodos que en conjunto representan algo más.

Los elementos básicos de OSM son:

- Los Límites: Elemento que contiene área total que cubre el archivo en latitud y longitud.
- Los nodos: Puntos cartográficos que contienen al menos la latitud y longitud, en nuestro caso además de lo anterior indican si un punto es algún semáforo.
- Las vías: Son elementos que se componen de dos o más nodos que representan una línea o polígono en el mapa. Es importante destacar que dentro de una vía los nodos sólo están presentes en modo de referencia, de modo que para acceder a ellos debemos referenciar su id.
- Las relaciones: Son conjunto de nodos, vías u otras relaciones que tienen propiedades en común. (Por ejemplo, todas las vías, nodos y relaciones que pertenecen al área de la Universidad del Bío-Bío)
- Las etiquetas: Son asignadas a nodos, vías, o relaciones y se utilizan para describir atributos de un elemento mediante una clave y de un valor, por ejemplo gracias a las etiquetas podemos saber el número de pistas que tiene un carril.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">

 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900"
maxlon="12.2524800"/>

 <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO"
uid="46882" visible="true" version="1" changeset="676636"
timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter"
uid="36744" visible="true" version="1" changeset="323878"
timestamp="2008-05-03T13:39:23Z"/>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" uid="46882"
version="1" changeset="676636" />
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5"
changeset="4142606" timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Straße"/>
 </way>
 <relation id="56688" user="kmvar" uid="56190" visible="true" version="28"
changeset="6947637" timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
  ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Küstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  <tag k="operator" v="Regionalverkehr Küste"/>
  <tag k="ref" v="123"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
 </relation>
 ...
</osm>
```

Ilustración 5: Formato de datos OSM

En la ilustración 5 se muestra un archivo *genérico* proveído por OSM. Observemos que una etiqueta </osm> abarca toda la información, tiene como límites cartográficos las coordenadas descritas por la etiqueta <bounds/>. Notemos además que el archivo tiene las etiquetas <node/> primero, luego las vías <way/> con sus correspondientes <nd/> y por último las relaciones <relation/> con sus respectivos miembros <member/> y etiquetas <tag/>. Este orden siempre se respeta en la estructuración de los archivos OSM, pero no significa que siempre existan relaciones o vías en un archivo.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

3.4.- Serialización de datos OSM

Como vimos, los datos que provee OSM están en un fichero de estructura XML. Para poder trabajar con ellos, es necesario que estos datos sean serializados en una estructura de datos directa en C#. La serialización consiste en un proceso que convierte un objeto en una secuencia de bytes para ser almacenados o transmitidos a una memoria, una base de datos o a un archivo. El principal propósito de este proceso es guardar el estado de un objeto para poder volver a crearlo cuando sea necesario.

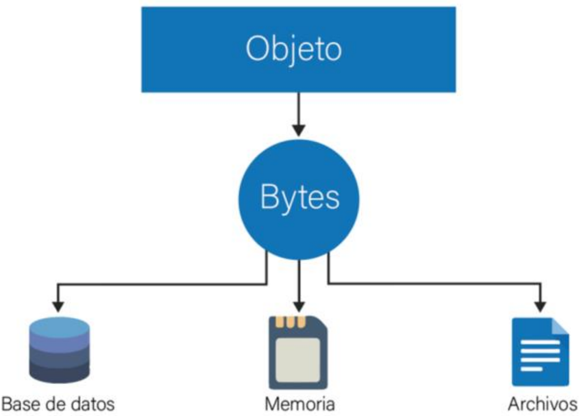

Ilustración 6: Proceso de la serialización de un objeto

En la sección 3.3.- vimos que cada atributo de un elemento OSM nos puede dar información útil a la hora de construir una infraestructura vial. A continuación listamos los atributos que utilizamos para serializar y filtrar la información

Tabla 2: Atributos y sus claves utilizados para filtrar la información que se utiliza en el proyecto

Nombre etiqueta	Clave	Valor	Descripción
bounds	minlat	{ límite mínimo latitud }	Auto explicativo
bounds	maxlat	{ límite máximo latitud }	Auto explicativo
bounds	minlon	{ límite mínimo longitud}	Auto explicativo
bounds	maxlon	{ límite máximo longitud }	Auto explicativo
node	id	{ id de un nodo }	Auto explicativo
node	lat	{ latitud de un nodo}	Auto explicativo
node	lon	{ longitud de un nodo }	Auto explicativo
node	highway	traffic_signals	Si está presente esta etiqueta, el nodo es marcado como un semáforo.
way -> tag	building:levels	{ Número de pisos de un edificio }	Cuando una way o vía sea un edificio, multiplicaremos este

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

			valor por una altura predefinida.
way -> tag	height	{ Altura de un edificio }	Si la etiqueta anterior no existe, usaremos este valor para definir altura de edificios
way -> tag	building	yes	Si se cumple, marcaremos la way como un edificio del mapa
way -> tag	lanes	{ Número de carriles }	Auto explicativo
way -> tag	name	{ Nombre de pista }	Auto explicativo
way -> tag	oneway	yes	Indica si la pista es de un sólo sentido, veremos más adelante que en general todas las pistas son de un sentido
way -> tag	highway	primary secondary tertiary residential living_street primary_link	Esta etiqueta puede tomar uno de los siguientes valores para ser considerada en el estudio, ya que calles de servicio, de tierra entre otras no están contempladas

De la tabla anterior se desprenden los distintos elementos que fueron utilizados en el proyecto; es decir mediante estos atributos y claves se filtraron los datos del archivo OSM, por lo que los atributos o elementos que no estén comprendidos en esta tabla quedan fuera del proyecto – y por ende – fuera de la memoria del sistema una vez que se termina la serialización.
Por ejemplo, se seleccionan los nodos que tengan la clave *highway* y que ésta clave tenga como valor *traffic_signals;* así se filtran los nodos que son semáforos en escena.
Por otro lado, veamos cómo una way puede representar distintos elementos de un mapa; para efectos de limitarnos a nuestro objetivo filtramos las *vías* que tienen por claves *highway, lanes, building, oneway,* entre otros. La clave *highway* en una etiqueta hija de *way* significa el tipo de calle de la vía; para este proyecto filtramos las que se consideran importantes dentro de una infraestructura vial como lo son las vías primarias, secundarias, terciarias y entre otras (descritas en tabla).

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

3.5.- Serialización XML en C#

La serialización XML serializa - valga la redundancia - las propiedades y los campos públicos de un objeto o los parámetros y valores devueltos de los métodos en una secuencia XML que en nuestro caso se condice con el esquema OSM. La serialización XML produce clases gracias a la librería de C# *System.Xml.Serialization* que contiene interfaces tanto para la serialización como deserialización de objetos.
Se puede aplicar filtros a un atributo o elemento de modo que podamos controlar la forma en que el serializador funciona en términos de filtración, para lo cual utilizaremos las claves del punto anterior.

```
1 referencia
void GetWays(XmlNodeList xmlNodeList)
{
    foreach (XmlNode node in xmlNodeList)
    {
        OsmWay way = new OsmWay(node);
        ways.Add(way);

    }
}

1 referencia
void GetNodes(XmlNodeList xmlNodeList)
{
    foreach (XmlNode n in xmlNodeList)
    {
        OsmNode node = new OsmNode(n);
        nodes[node.ID] = node;
    }
}

1 referencia
void SetBounds(XmlNode xmlNode)
{
    bounds = new OsmBounds(xmlNode);
}
```

Ilustración 7: Funciones que serializan en objetos de clase distintos elementos del archivo OSM/XML

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Observemos un set de datos antes y después de ser procesados:

Ilustración 8: Sector de prueba Paicaví-Carrera. A la izquierda los datos representados por OSM y a la derecha los mismos datos luego de ser serializados visualizados en Unity

Tal como se observa en la ilustración 8, los datos han sido *consumidos* correctamente en la memoria; en azul observamos edificios, estructuras o zonas como zonas residenciales y limitaciones en general, mientras que en rojo observamos las calles. Ahora en adelante podremos utilizar de manera sistemática estos objetos, los cuales se describen en el siguiente punto.

3.6.- Diagrama de clases

Para efectos de creación del mapa nos centraremos en las etiquetas node,way y bounds, mientras que la etiqueta tag es inherente a cada elemento.
El diagrama de clases es el que sigue:

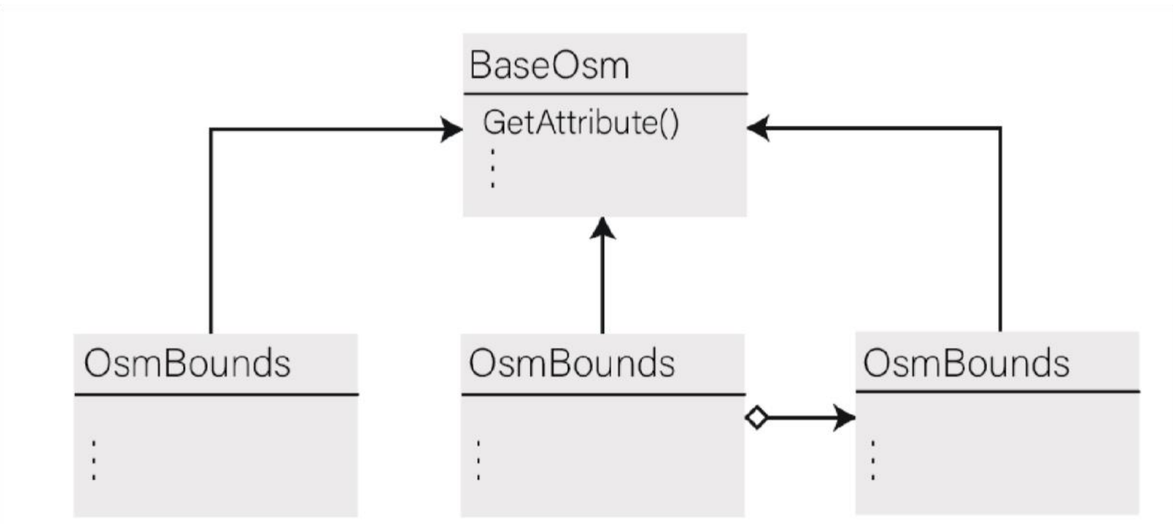

Ilustración 9: Diagrama de clases que componen la serialización XML a objetos de clase

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Revisaremos en detalle cada clase para entender su función.

- **BaseOsm:** Como indica su nombre, esta clase sirve de base para todas específicamente para leer los atributos individuales de cada elemento XML, por lo que cualquier clase que herede de BaseOsm podrá leer atributos convertidos a tipos requeridos.

```
class BaseOsm
{
    /// <summary>
    /// Get an attribute's value from the collection using the given 'attrName'.
    /// </summary>
    /// <typeparam name="T">Data type</typeparam>
    /// <param name="attrName">Name of the attribute</param>
    /// <param name="attributes">Node's attribute collection</param>
    /// <returns>The value of the attribute converted to the required type</returns>
    17 referencias
    protected T GetAttribute<T>(string attrName, XmlAttributeCollection attributes)
    {
        if (!(attributes[attrName] == null))
        {
            string strValue = attributes[attrName].Value;
            return (T)Convert.ChangeType(strValue, typeof(T));
        }
        else return default(T);
    }
}
```

Ilustración 10: Clase BaseOsm, con el único método de obtener valores de claves de una colección de atributos XML.

- OsmBounds: Esta clase es utilizada al inicio de la ejecución para dimensionar el área que comprende nuestro archivo OSM. Observemos cómo la clase hace uso directo de la clase base de OSM, para obtener las coordenadas que fijarán el límite de nuestro mapa. Notemos además que contiene cálculos para una proyección Mercator, la que abordaremos más adelante en este mismo capítulo.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
class OsmBounds : BaseOsm
{   /// <summary> Minimum latitude (y-axis)
    3 referencias
    public float MinLat { get; private set; }
    /// <summary> Maximum latitude (y-axis)
    3 referencias
    public float MaxLat { get; private set; }
    /// <summary> Minimum longitude (x-axis)
    3 referencias
    public float MinLon { get; private set; }
    /// <summary> Maximum longitude (x-axis)
    3 referencias
    public float MaxLon { get; private set; }
    /// <summary> Centre of the map in Unity units.
    5 referencias
    public Vector3 Centre { get; private set; }

    /// <summary> Constructor.
    1 referencia
    public OsmBounds(XmlNode node)
    {
        // Get the values from the node
        MinLat = GetAttribute<float>("minlat", node.Attributes);
        MaxLat = GetAttribute<float>("maxlat", node.Attributes);
        MinLon = GetAttribute<float>("minlon", node.Attributes);
        MaxLon = GetAttribute<float>("maxlon", node.Attributes);

        // Create the centre location
        float x = (float)((MercatorProjection.lonToX(MaxLon) +
                  MercatorProjection.lonToX(MinLon)) / 2);
        float y = (float)((MercatorProjection.latToY(MaxLat) +
                  MercatorProjection.latToY(MinLat)) / 2);
        Centre = new Vector3(x, 0, y);
    }
}
```

Ilustración 11: Clase OsmBounds con sus distintas variables de acceso público

- OsmNode: Esta clase se encarga de serializar los elementos *node* del archivo y filtrar en base a lo discutido en el punto 3.4., por ejemplo en la ilustración 12 se demuestra cómo se agregan los nodos que son semáforos mediante la variable *isTrafficSignals*

```
XmlNodeList tags = node.SelectNodes("tag");
foreach (XmlNode t in tags)
{
    string key = GetAttribute<string>("k", t.Attributes);
    if (key == "highway")
    {
        string value = GetAttribute<string>("v", t.Attributes);
        if (value.Equals("traffic_signals")) isTrafficSignals = true;
    }
}
```

Ilustración 12: se agregan atributos a nodos que tengan la clave highway

- OsmWay: Esta clase al igual que las anteriores, serializa los elementos way mediante la primera clase descrita en esta lista, y del mismo modo que OsmNode; filtra y serializa elementos útiles para el proyecto.

En general el proceso es iterar en base a cada subelemento que posea el elemento instanciado gracias a la serialización

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

3.7.- Construcción del entorno

Una vez que los datos fueron correctamente serializados en objetos, es posible empezar a utilizarlos para generar de manera procedural las calles y edificios, pero antes analicemos lo siguiente: mientras que Unity provee de un espacio amplio para la creación de escenas, recordemos que estamos trabajando con longitudes y latitudes, y al mismo tiempo, dada la forma esférica - específicamente elipsoidal - de la tierra, es imposible proyectar de manera que las distancias, superficies y formas no sufran alteraciones, por lo que es necesario omitir una u otra que al final dependerá del uso que tendrá el mapa en cuestión.

3.7.1.- Proyección de Mercator

Para este proyecto, trabajaremos con la proyección de Mercator a la hora de *convertir* las coordenadas cartográficas a puntos cartesianos. La proyección de Mercator fue creada por el flamenco Gerardus Mercator en 1569 para elaborar mapas de la superficie terrestre, y ha sido ampliamente utilizada incluso hasta la fecha, ya que permitía trazar rutas en las cartas náuticas como líneas rectas e ininterrumpidas.
Es una proyección cilíndrica y debido a esto sufre deformaciones a medida que se acerca a los polos. Si bien existen otras proyecciones, se justifica su elección ya que preserva la forma de edificios y calles debido a que es una proyección conforme.
Por último, es la proyección que utilizan los desarrolladores de Openstreetmap, lo que hace preferible su uso.

Su descripción matemática es la que sigue [4]:

$$x = \lambda - \lambda_0 \\ y = ln(tan\,\phi + sec\,\phi) \quad (3)$$

Donde ϕ es la latitud y λ la longitud (siendo λ_0 la longitud central del mapa).

3.7.2.- Construcción procedural de vías

La generación procedural es un método o forma de crear datos mediante algoritmos, es decir de manera sistemática, contraria a métodos anteriores para - por ejemplo - la creación de escenarios que convencionalmente se realizaban de forma manual.
En otras palabras, a un algoritmo se le entregan datos como parámetros que pueden ser modificados para conseguir distintos resultados.
En el presente inciso, abordaremos la construcción de las calles en escena de Unity para lo cual se pudieron identificar dos elementos principales que son necesarias para la producción, que son:

- Generación procedural de Meshes / Mallas
- Generación procedural de grafo que contiene estructura vial cargada

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Mientras que ambas consisten en una generación procedural, difieren en lo que realizan específicamente; la primera construye de manera visual las carreteras mientras que la segunda como se puede inferir por su nombre, crea una estructura de datos que contiene un grafo con los datos OSM procesados que utilizaremos más adelante para ajustar el comportamiento individual de cada agente.

3.7.3.- Generación procedural de Meshes / Mallas

Un *mesh* o malla poligonal, es una superficie generada por algoritmos dado un set de vértices (3 como mínimo) y que en conjunto pueden representar una figura tridimensional más compleja o más bien simple, dependiendo del caso. Por ejemplo, en la industria de los videojuegos se utilizan modelos de personajes y de atuendos por separado, a modo de aumentar la personalización en el juego y por otra parte existen figuras más simples como lo son cuadrados o figuras simples con texturas.
Según el número de vértices se pueden generar modelos más detallados; como vemos en la ilustración 13, con 4 vértices se puede obtener mayor detalle y suavidad en los objetos, sin embargo, dada la simplicidad que tiene construir una calle y edificios, usaremos una técnica que calcula los triángulos (tris) en base a 2 puntos en el mapa, vale decir que en este proyecto los edificios no interactúan de ninguna manera con el sistema de tráfico.

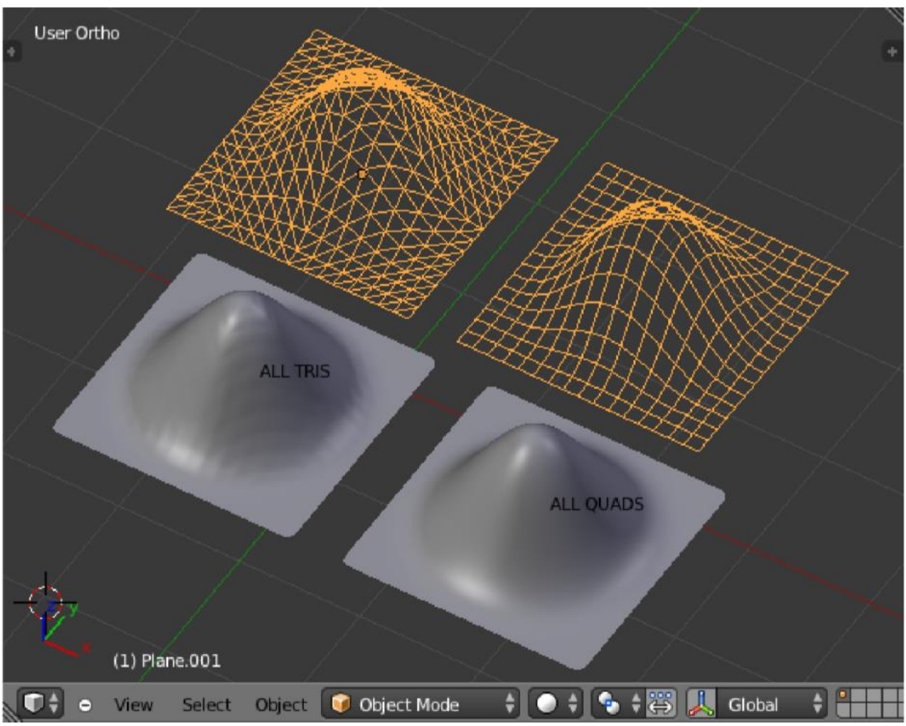

Ilustración 13: Distintas mallas poligonales representando una misma figura, a la izquierda representada utilizando triángulos y en la segunda cuadrados.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Otros conceptos a tener en cuenta para la generación procedural de meshes son los vectores normales a las superficies y las texturas que se deben aplicar a cada mesh.

- Vector normal: En la geometría, el vector normal es la dirección a la que una superficie es perpendicular; en resumen nos es útil para mostrar en la dirección correcta la textura
- Texturas : Para el modelado 3D, se utiliza una técnica llamada UV Mapping, que consiste básicamente en tener texturas en una imagen 2D y aplicar cada trozo de textura a una superficie 3D

Ilustración 14: Texturas utilizadas para el proyecto

Ahora que reunimos todos los conceptos previos, somos capaces de profundizar en el procedimiento mismo. Cuando los datos OSM fueron correctamente serializados, se procede a construir vía por vía, en donde se consulta por cada nodo perteneciente a la vía mediante LINQ, un componente de la plataforma .NET de Microsoft llamado LINQ (**L**anguage **In**tegrated **Q**uery) que agrega capacidades de consulta a datos (tipo SQL) de manera nativa a los lenguajes .NET (C#).
Luego por cada nodo se calcula un distanciamiento que viene dado por el número de pistas en el caso de las calles, y por la altura para el caso de los edificios, de modo que los modelos generados se condigan con los datos de OSM. Finalmente, se construyen los triángulos dado los vértices calculados mediante el distanciamiento anterior. La siguiente ilustración (15) explica el proceso:

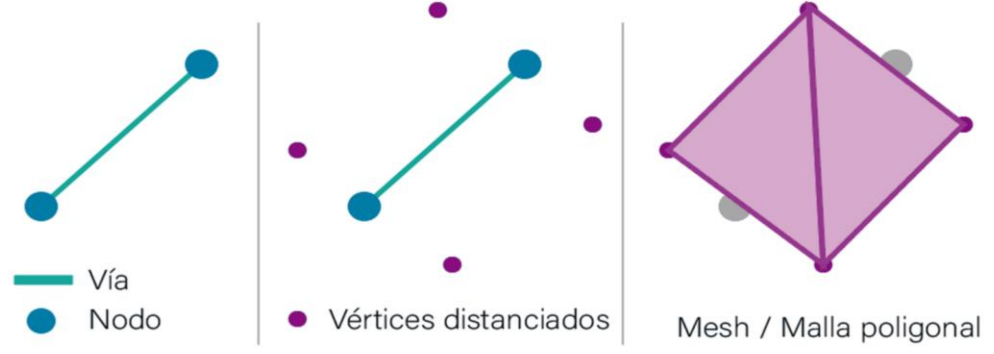

Ilustración 15: Ejemplo de generación procedural de mallas poligonales

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Con esto, ya podemos visualizar en escena las calles que serán representadas:

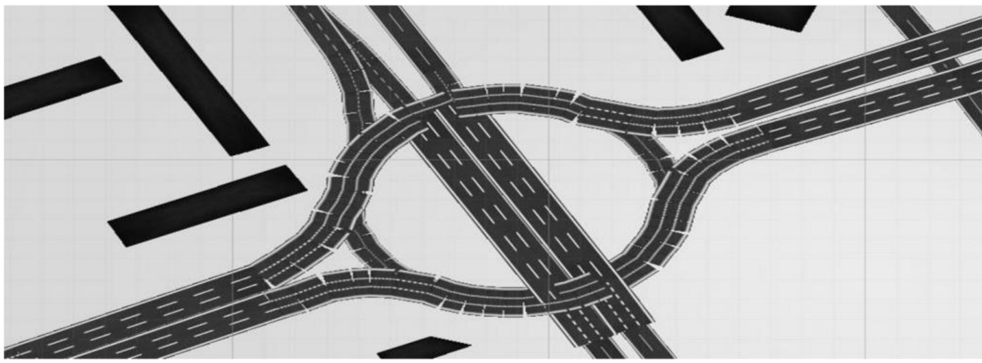

Ilustración 16: Meshes de Paicaví – Carrera generadas

3.7.4.- Generación procedural de grafo que contiene estructura vial cargada

Para el grafo, se considera una estructura de datos distinta a la que le cargamos los datos OSM. Para la realización de este punto, utilizamos una librería de código abierto para Unity3D llamada **2D/3D PathFinder Full** creada por Vijay Veluri. La librería provee de utilidades para la creación de grafos.
El procedimiento para la generación del grafo consiste en construir cada nodo por cada vía (parecido al procedimiento del punto anterior) pero en este caso, era necesario modificar el grafo cada vez que existiera una intersección, lo que al mismo tiempo significaba la no creación de un nodo, por lo que la lógica al abordar esta situación particular no era trivial; fue común llegar a resultados como los de la siguiente imagen (17), en donde el orden se alteraba debido a nodos superpuestos.

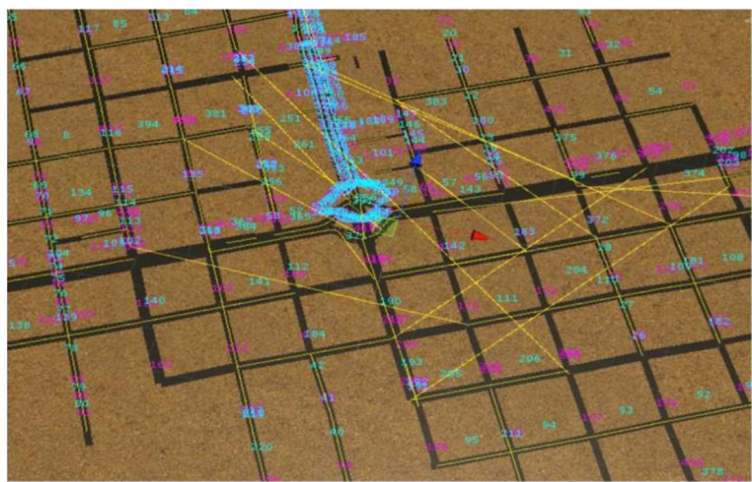

Ilustración 17: Primeros intentos de generar de manera procedural el grafo, en amarillo las vías que componen el set de datos.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Para solucionar lo anterior se creó una función (ilustración 18) que recibe el nodo como parámetro en forma de posición vectorial (conocidos como Vector3, por sus coordenadas x,y,z) y daba como resultado -1 si es que no existía un nodo previamente en el grafo en esa posición, y por otro lado arroja la ID del nodo si es que éste ya existía, de modo que se pudiera enlazar el camino con este nodo sobrepuesto.

```
// check if a point overlaps another
1 referencia
public int getID(Vector3 point)
{
    Node nearestNode = null;

    foreach (var node in script.graphData.nodes)
    {
        if (node.Position == point)
        {
            nearestNode = node;
            return node.autoGeneratedID;
        }
    }

    return -1;
}
```

Ilustración 18: Función que determina si un punto ya existe en una posición

Finalmente, se logró construir el grafo que contiene la estructura vial de los datos cargados, por lo que ahora podríamos implementar una inteligencia artificial que sea capaz de detectar un camino.

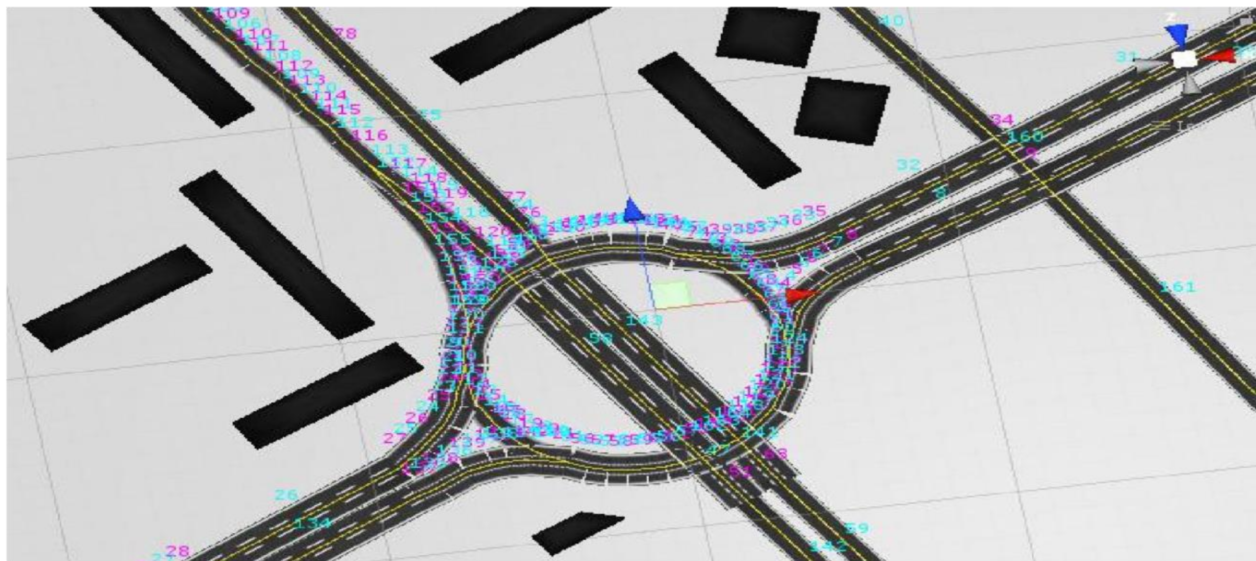

Ilustración 19: Resultado final de generación procedural de grafos a partir de un archivo osm.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Capítulo: 4 Inteligencia Artificial como Agentes Microscópicos del Tránsito

Los vehículos son parte intrínseca del flujo vehicular, se pueden describir como átomos de un sistema superior que recorren un camino siguiendo algunas reglas básicas de convivencia. Si bien se puede explicar el comportamiento individual de los vehículos, en la realidad es muy difícil predecir las decisiones que toman las personas a la hora de circular por un corredor vial. Esto último explica por qué pese a toda la tecnología que existe en términos de seguridad vehicular, la cantidad de accidentes de tráfico muestra una tendencia al alza desde el 2017 [5], ya que es el error humano el principal factor de los accidentes de tránsito.
Y es que la mente humana es tan compleja, que a día de hoy sigue siendo un misterio el poder describir de forma matemática el comportamiento humano, específicamente el comportamiento de las y los conductores vehiculares.
Dicho esto, podemos establecer que nuestro límite u objetivo para este aspecto es lograr un comportamiento lineal, en donde cada agente respete a los demás agentes vehiculares con reglas y lógicas que abordaremos en el presente capítulo.

4.1.- Comportamiento deseado

Para efectos de estudiar el flujo de tráfico, nos basta con que los vehículos se desplacen en línea recta siguiendo su camino y en ocasiones cambiar de dirección en las intersecciones por las que pasan. Además, deben respetar los semáforos y mantener una distancia prudente a los autos que le siguen, que se estableció como 1.5 unidades de distancia; en nuestro caso son metros.

4.2.- Implementación Inteligencia Artificial

Para completar el sistema de tráfico requerimos agentes que circulen por nuestra simulación; cada agente individual debe ser capaz de decidir su siguiente paso. De manera simplificada, el algoritmo por el cual se rigen los agentes vehiculares se describe en la ilustración 20:

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

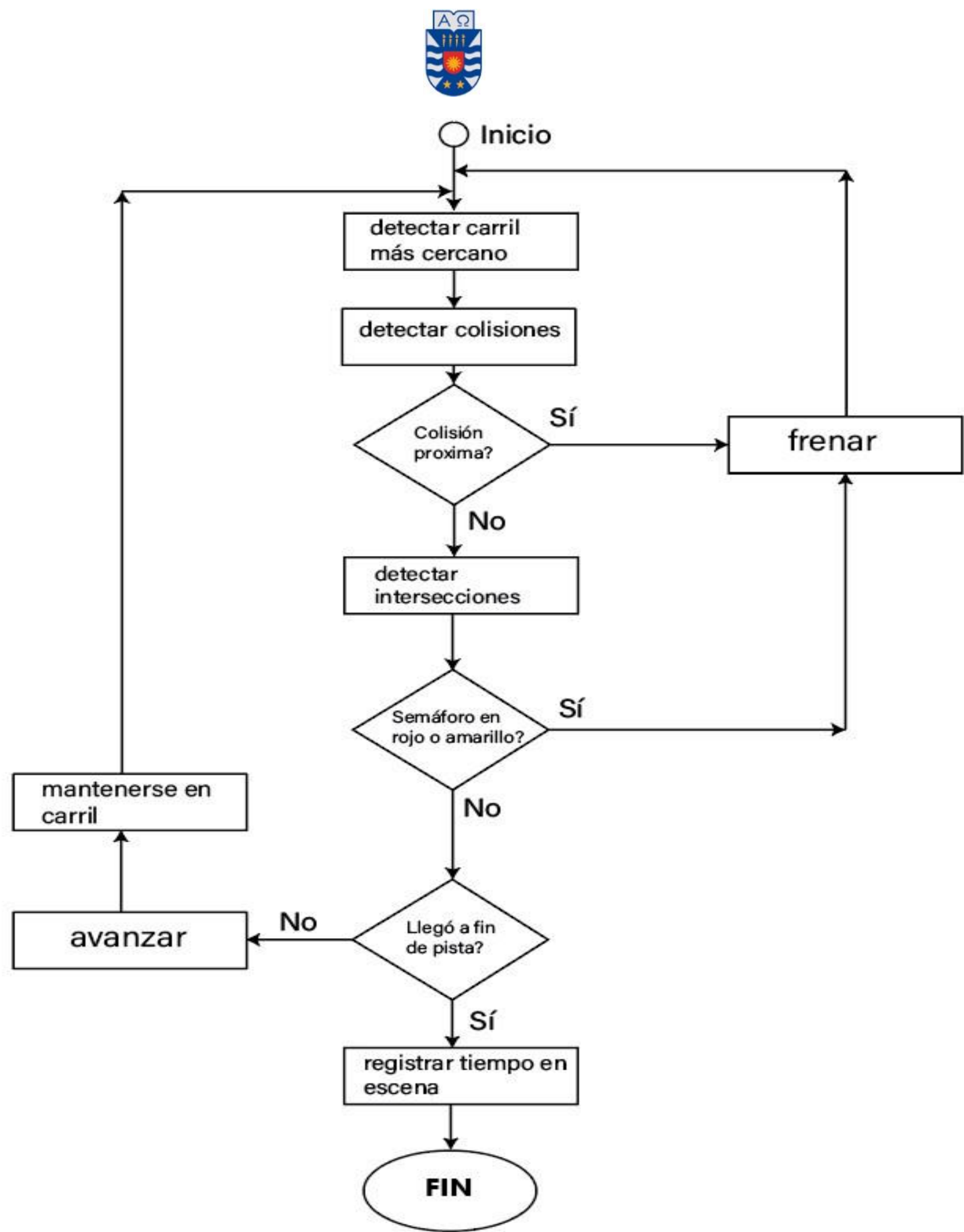

Ilustración 20: Algoritmo general que define la toma de decisiones de los agentes vehiculares

Para poder entender de mejor manera este diagrama, hay que tener en consideración que este procesamiento ocurre para cada vehículo por cada cuadro de ejecución, es decir ocurre cada vez que el motor gráfico y el procesador gráfico calculan un cuadro.
Como podemos observar en la ilustración 20, el agente tomará decisiones en base a cálculos y detecciones, por ejemplo para detectar el carril más cercano es en sí mismo un proceso de cómputo que posteriormente permite al agente mantenerse en el carril. En estricto rigor, el vehículo repetirá este algoritmo una vez por cuadro hasta que llega al final de pista.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

4.2.1.- Desplazamiento y detención

Los elementos básicos que conforman el concepto de transporte vehicular/motorizado son el desplazamiento y el freno, que pueden ser utilizados según estime conveniente el usuario o el agente como en nuestro caso.

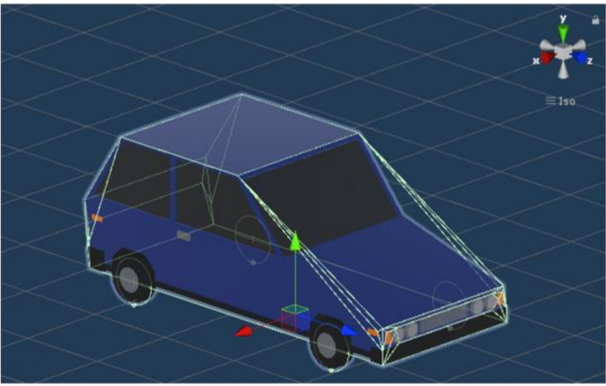

Ilustración 21: Agente vehicular

En primera instancia, se implementó un sistema completo de tracción vehicular a los agentes que viene pre-implementado en Unity3D llamado Wheel Colliders y que mediante algunos ajustes era muy fácil de implementar. El complemento funcionaba dividiendo el modelo vehicular en 5 partes: la carrocería, y las cuatro ruedas correspondientes, de modo que cada vez que se quisiera *acelerar*, se ajustaba el torque al máximo permitido, y de forma contraria - al frenar - se ajustaba en cero. Además, cada vez que se encontrara frente a un obstáculo, lo esquivaba mediante unos sensores que realiza proyectando rayos (o vectores) en distintas direcciones, de modo que - según el sensor activado - ajusta el giro evitando así los obstáculos, como se ilustra en la siguiente imagen.

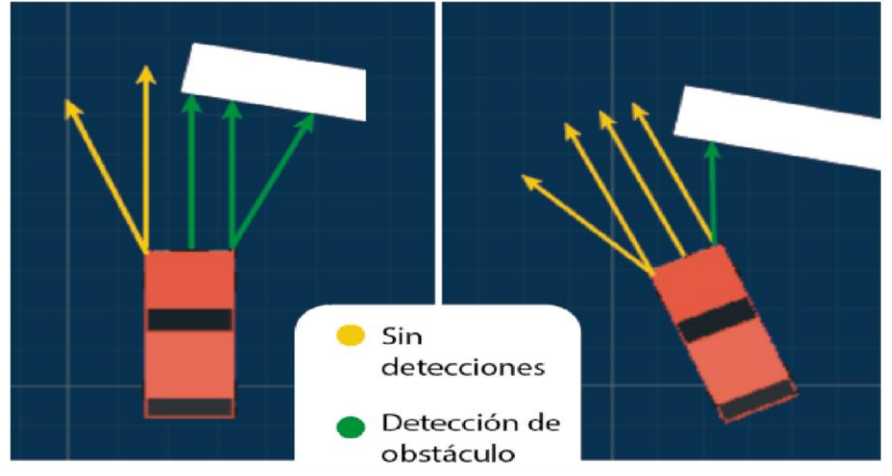

Ilustración 22: Sistema de detección de obstáculos

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Si bien este acercamiento resultó ser bastante prometedor al principio, se observó que los problemas comenzaron a surgir cuando el vehículo alcanzaba su máxima velocidad; el vehículo se deslizaba fácilmente a pesar de ajustar la fricción y cambiar las distintas variables que componían el *motor* del agente; lo que provocaba constantes colisiones y una fácil pérdida de la trayectoria. No obstante, el problema mayor estaba aún por venir, pues claro, cuando hablamos de sólo un agente en escena no debieran existir mayores problemas de rendimiento, pero si consideramos que en nuestro caso particular necesitamos simular la mayor cantidad posible de vehículos, este procesamiento pasa la cuenta.

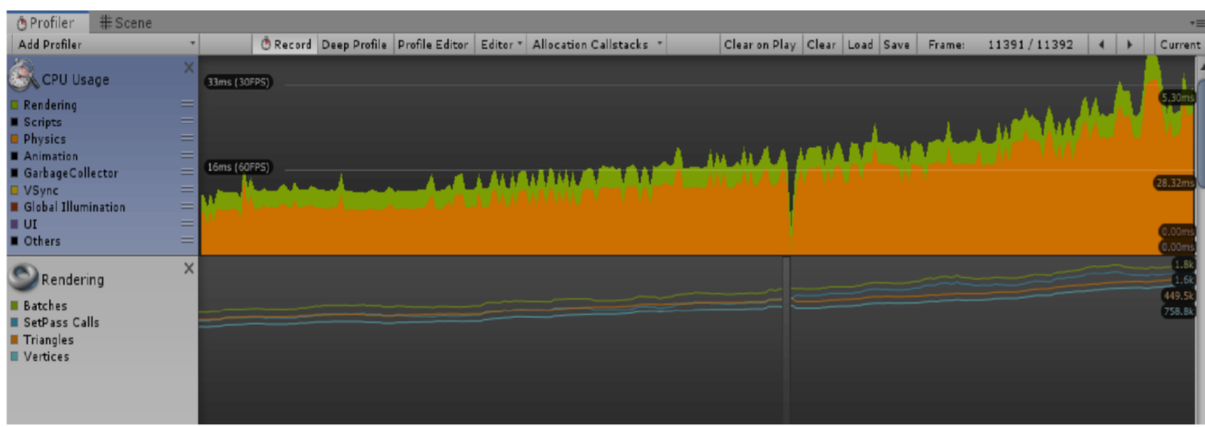

Ilustración 23: Perfil de rendimiento con alrededor de 100 vehículos

Los vehículos que contenían el script con el comportamiento descrito anteriormente en su conjunto rendían bien al comienzo pero conforme se instanciaban más vehículos, el perfil de rendimiento aumentaba en exceso llegando hasta un tiempo de cómputo de ~35ms como se observa en la ilustración 23, lo que a largo plazo no es conveniente si queremos lidiar eventualmente con muchos vehículos en escena. Como mencionamos, sólo el tiempo para el cálculo de físicas por cuadro ascendió hasta los 35ms, por lo que se decide tomar un enfoque más simplista que al mismo tiempo, significó trivializar la velocidad vehicular como se explica más adelante en este mismo capítulo.
Y tiene más sentido aún al considerar que nuestro alcance es el flujo de tráfico en sí y no el comportamiento específico vehicular, de todos modos se asume un apilamiento por carril donde no se espera que los vehículos cambien de carril si no es para cambiar de dirección.
Finalmente, se opta por asignar un colisionador prismático rectangular (ilustración 25) que contiene el modelo vehicular; por cada cuadro en el que el agente esté avanzando, se le aplica una traslación a una velocidad constante, multiplicada por la dirección vectorial a la que el vehículo está dirigiéndose, multiplicado por un delta T que es el tiempo que tomó calcular el último cuadro (ilustración 24), mientras que para detenerse, el vehículo simplemente deja de recibir esta traslación.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
transform.Translate(speed * Vector3.forward * Time.deltaTime);
```

Ilustración 24: Función de traslación simple

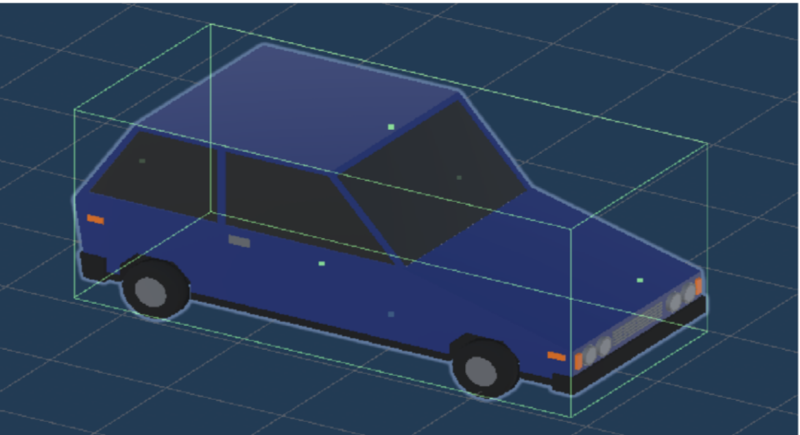

Ilustración 25: Vehículo agente y su colisionador en verde

Si comparamos el rendimiento del nuevo sistema con aproximadamente la misma cantidad de vehículos, podemos comprobar que es mucho más eficiente en términos de cálculo por cuadro que el método anterior (4ms vs 35ms), por lo que nos permite simular una mayor cantidad de vehículos que es precisamente nuestro propósito.
Notemos además que en general el perfilador se mantiene estable en la ilustración 26 y no demuestra un crecimiento importante en el consumo de procesador como sucedió en la ilustración 23.

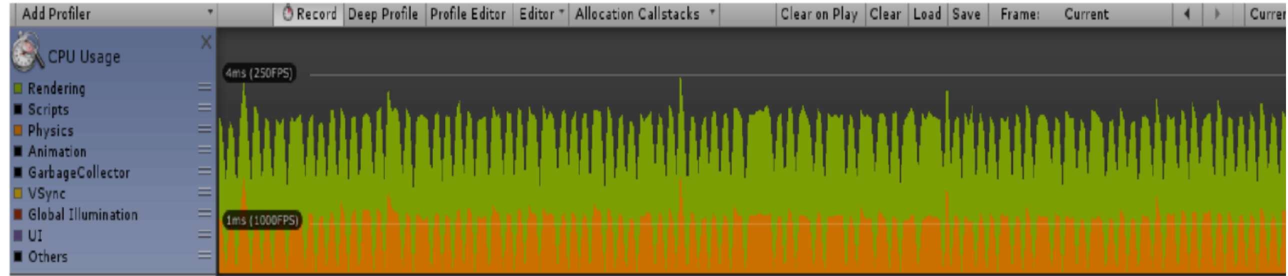

Ilustración 26: Perfil de rendimiento con enfoque simplista

4.2.2.- Sistema de carriles

Como vimos en el capítulo anterior realizamos la construcción procedural de un grafo en base a los datos de un mapa; grafo que contiene puntos vectoriales y forma trazados o caminos entre ellos, los cuales a su vez tienen una dirección vectorial hacia el siguiente punto y número de pistas.
Por esto, el mecanismo en que los agentes conductores se basan para ajustarse a un carril consiste en primero estimar en qué carril se encuentra, y segundo calcular la distancia

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

hacia el segmento más cercano, y en base a esta distancia determinar qué acción realizar; como por ejemplo virar más hacia la derecha para acercarse más a la distancia deseada que tiene el carril según el número de pistas.

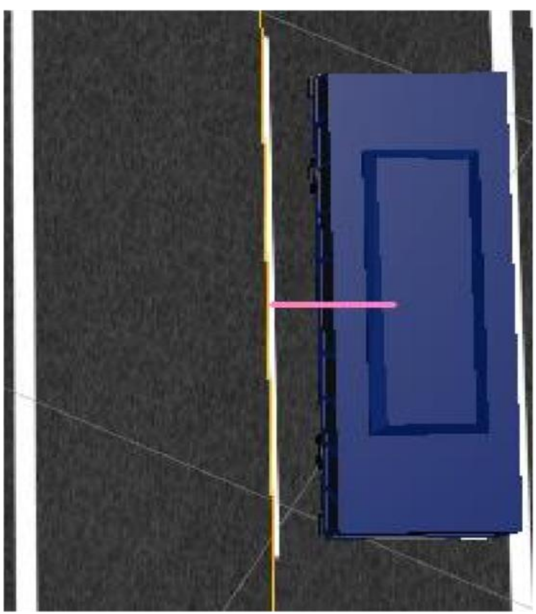

Ilustración 27: Como podemos observar en la imagen, el bus mantiene una distancia (en rosado) con el centro

En general, los pasos que siguen los agentes para este inciso son los siguientes:

- Detectar el segmento más cercano: Los vehículos realizan un cálculo en donde comparan la distancia del vehículo hacia el punto medio de cada camino en el grafo, en donde la distancia menor será entonces el camino más cercano sobre el cual se guía el vehículo
- Detectar punto más cercano del segmento: Como el cálculo anterior se realiza en base a un punto medio, se debe obtener el punto más cercano al vehículo de modo que se pueda obtener una distancia hacia el centro de la pista de manera correcta; para ello se utiliza la proyección del punto en que se encuentra el vehículo sobre el segmento mediante el producto punto, finalmente el resultado del producto punto multiplicado por la dirección del segmento se evalúa como el punto más cercano al vehículo, cabe destacar que es necesario acotar este resultado a los puntos que forman y pertenecen al segmento.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
// a - punto a del segmento
//b - punto b del segmento
//point - el punto del cual queremos obtener el punto más cercano al segmento
2 referencias
public Vector3 ClosestPointOnLine(Vector3 a, Vector3 b, Vector3 point)
{
    var vector1 = point - a;
    var vector2 = (b - a).normalized;

    var dist = Vector3.Distance(a, b);
    var t = Vector3.Dot(vector2, vector1);
    if (t <= 0)
        return a;
    if (t >= dist)
        return b;
    var vector3 = vector2 * t;
    Vector3 closestPoint = a + vector3;
    return closestPoint;


}
```

Ilustración 28: Función utilizada para calcular el punto más cercano de un punto a una recta AB

- Definir el carril sobre el cual está el vehículo: Una vez que se tiene el punto más cercano en el segmento, es posible calcular la distancia del vehículo hacia el centro y compararla con el punto más cercano, de modo que se pueda decir si el auto está a la izquierda o derecha del segmento. Para determinar en qué carril se encuentra el vehículo, se definen rangos de distancia que fijan las posibles opciones hasta pistas de 3 carriles como se ilustra en la siguiente imagen:

Ilustración 29: Pistas hasta 3 carriles

Luego según la distancia hacia el centro (que se evalúa en valores absolutos), el número de carriles de la pista y dependiendo de si el vehículo está a la derecha o a la izquierda giramos un grado por cuadro hasta que el agente llegue a la distancia que le corresponde por carril y se alinee con la dirección de la pista; de este modo se asegura que se mantendrá siempre en su carril y el único momento en que el agente puede cambiar de carril es cuando el número de carriles por pista disminuye o aumenta, o bien cuando gira en alguna intersección.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

La desventaja que posee este sistema, es que a velocidades muy altas los vehículos tienden a dar saltos entre las distancias hacia el centro calculadas, por lo que el comportamiento particular se vuelve errático, moviéndose de izquierda a derecha de manera sinusoidal, como muestra la imagen. Esto último se explica debido al desplazamiento natural que implica el movimiento a velocidades altas.
Por ello se decide finalmente utilizar valores representativos para la velocidad descritos en la tabla 3, de modo que se logre un comportamiento adecuado del flujo de tráfico simulado.

Ilustración 30: El agente tiende a perder el rumbo cuando se desplaza a una velocidad muy alta, o existe una tasa de FPS baja

4.2.3.- Detección de colisiones

Para que un vehículo responda correctamente a su entorno, debe recopilar información que le permita tomar decisiones en base a éstas. Como se mencionó anteriormente, esto se puede realizar mediante la proyección de rayos que "devuelven" al auto información que permite por ejemplo, determinar si el rayo se proyecta sobre otro vehículo o bien sobre un semáforo, y determinar para el último caso si está en rojo, amarillo o verde a fin de tomar una decisión sobre qué hacer en el siguiente paso.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Ilustración 31: Demostración de vehículos detenidos a cierta distancia del siguiente vehículo o intersección

Para esto, Unity provee de una API llamada Physics.Raycast [6] que es de utilidad a la hora de querer captar información del entorno y funciona como se ha descrito anteriormente: desde un punto, se proyecta un rayo - o línea vectorial - con cierta magnitud o largo de rayo, y si éste incide en algún objeto que **tenga colisionador**, se llaman a las funciones que gatillan estos eventos, que son:

- **OnCollisionEnter**: Se llama la primera vez que el rayo detecta una colisión
- **OnCollisionStay:** Se llama cada vez que el rayo permanece en una colisión
- **OnCollisionExit**: Se llama cuando en el próximo cuadro el rayo ya no incide en el colisionador

Se debe tener en cuenta que los colisionadores ocupan un espacio físico dentro de la escena, y en situaciones sólo queremos obtener detalles de algún objeto y no colisionar necesariamente, como por ejemplo en los semáforos que se detallan en el siguiente capítulo. Para ello, Unity agrega también la posibilidad de transformar un colisionador en un *trigger* (gatillo en español), que permite a los objetos ser atravesados sin provocar una colisión. De manera similar, sus eventos son:

- **OnTriggerEnter**: Se llama la primera vez que el rayo detecta un trigger.
- **OnTriggerStay**: Se llama cada vez que el rayo permanece en un trigger.

- **OnTriggerExit**: Se llama cuando en el próximo cuadro el rayo ya no incide en el trigger.

También se pueden proyectar rayos directamente y no depender de funciones, como en el siguiente ejemplo:

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
if (Physics.Raycast(sensorStartPos, transform.forward, out hit, sensorLenght * 1.5f))
{
    if (hit.collider.CompareTag("Intersection"))[...]
    else if (hit.collider.CompareTag("Vehicle"))
    {
        isBraking = true;
        return;
    }

}
```

Ilustración 32: Proyección de rayo inmediata mediante la función Raycast() de la clase Physics

4.2.4.- Agentes y velocidades

Los modelos vehiculares que se utilizan en este proyecto fueron descargados gratuitamente desde la tienda de Unity, el paquete se llama **Cartoon Vehicles Lowpoly** y su autor es el usuario **Lowpoly_Master.** Como se adelantaba al final del punto 3.2.2, la velocidad de los vehículos es más bien representativa, no obstante existen vehículos que se mueven lento en comparación a otros y causan enlentecimiento en el flujo tal como sucede en la realidad.
Podemos describir la velocidad representativa de la siguiente forma:

$$(4)\quad Velocidad\ Representativa = Velocidad\ Real * 3$$

Tabla 3: Modelos 3D y sus respectivas velocidades

Vehículo	Velocidad Representativa	Velocidad Real
	60 km/h	20
	60 km/h	20

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

	39 km/h	13
	48 km/h	16
	30 km/h	10

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Capítulo: 5 Sistema de tráfico

Una vez que se tienen el entorno y los agentes vehiculares se puede combinar éstos en un sistema de tráfico que permita cuantificar y asimismo analizar los tiempos de viaje promedio, flujos de tráfico y densidad de tráfico vehicular. En este capítulo abordaremos la implementación de los elementos que condicionan un sistema de tráfico, como lo son: semáforos, entradas, salidas y registradores de datos.
Hasta ahora se cuenta con una estructura vial física (en mesh y grafo) y agentes vehiculares listos para entrar a la escena, por lo que sólo nos queda agregar los elementos mencionados anteriormente; para ello se filtran los nodos y pistas necesarias según se requiera; por ejemplo los semáforos sólo se colocan en una intersección si es que en el mapa OSM existe, esto para asignar los comportamientos que siguen a continuación.

5.1.- Semáforos

Para simular correctamente los semáforos, es necesario llevar cuenta del estado del ciclo (verde, amarillo, rojo) y alternar ambos lados de manera sincronizada. Para esto, creamos un Gameobject con paredes (ilustración 33) que determinarán si el vehículo puede atravesar el colisionador de la pared o no. Se agregan también modelos físicos de los semáforos. Para el ciclo, utilizamos un método de Unity llamado *Coroutines* - corrutinas en español -, que permiten la ejecución paralela de funciones (no confundir con *multithreading* que consiste en crear hilos individuales para cada proceso directamente en el procesador) por lo que se ejecutan dos corrutinas que llevan cuenta de dos ciclos (ilustración 34); es decir actualmente no existen semáforos de viraje en la simulación.

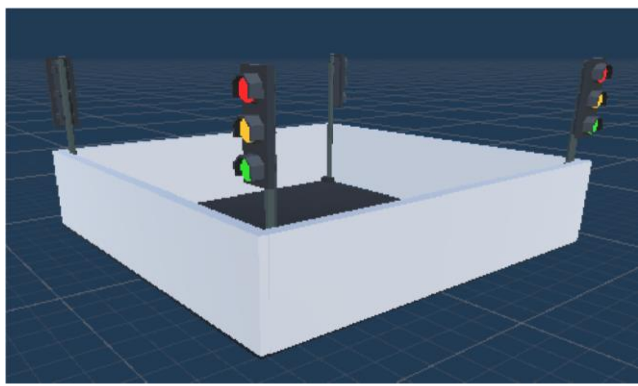

Ilustración 33: Semáforos como objeto

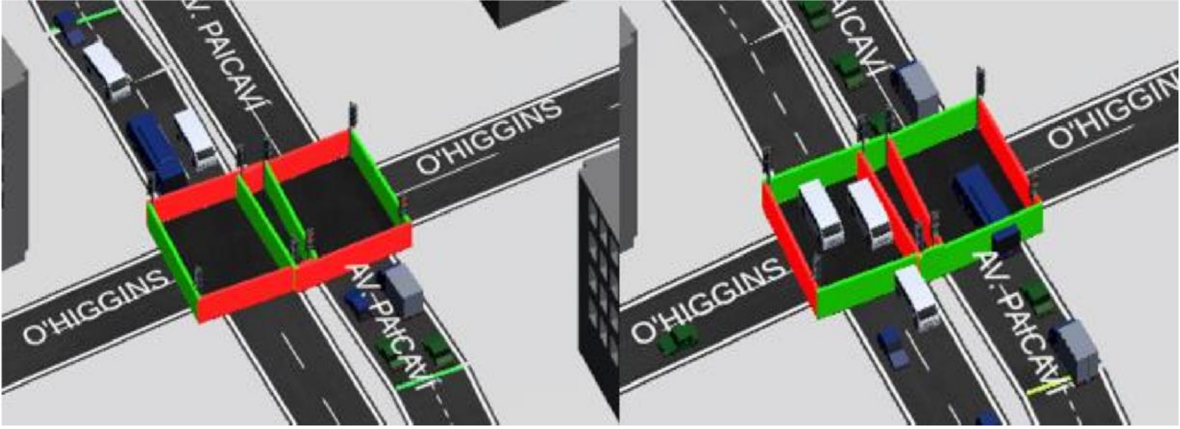

Ilustración 34: Ciclos de un semáforo, en la imagen se aprecian 2 semáforos

Para agregar la funcionalidad anterior y la que cambiará los colores y estados de cada semáforo agregamos un script (ilustración 35) que recibe como parámetro un ciclo en segundos, y en base a este se asignan los tiempos de los semáforos de la siguiente manera:

(5) $Tiempo\ en\ Rojo:\ c$

- $Tiempo\ en\ Amarillo: c * 0.3$
- $Tiempo\ en\ Verde: c * 0.7$

donde c es el ciclo en segundos para cada semáforo.

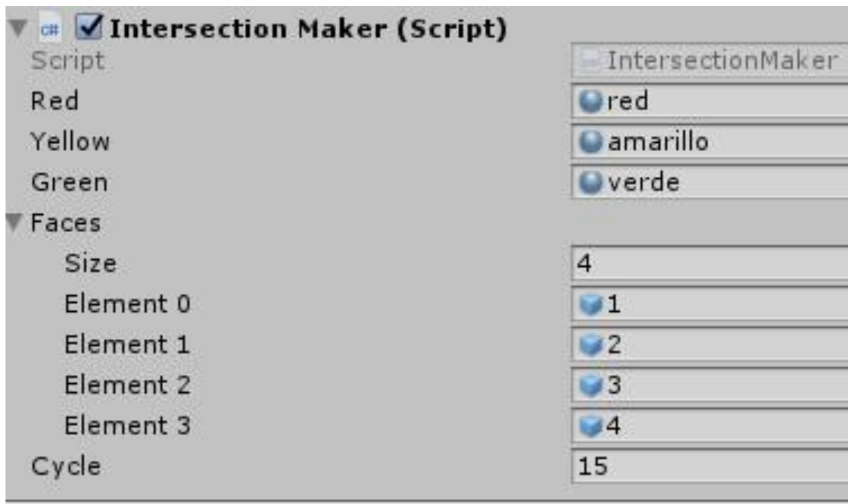

Ilustración 35: Script que se controla los ciclos de semáforos, recibe un parametro Cycle que define el temporizador del objeto

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Por otra parte, los agentes detectan estas paredes mediante Raycast y según sea el estado/color de la muralla deciden la próxima acción a tomar que puede ser continuar con el camino - si es verde, o detenerse en el caso de amarillo y rojo. Los vehículos que ya entraron a la intersección, tienen la ventaja de continuar su camino a toda costa, ya que se espera que éstos no interrumpan el flujo de tráfico. Por esto último es necesario además controlar y detectar vehículos que pueden estar causando un bloqueo durante largos períodos, por aquello a los vehículos se les agrega un cronómetro que cuenta el tiempo que han estado **detenidos y sobre la intersección;** en términos prácticos esto es que si un vehículo permanece sobre una intersección por más de 3 segundos (arbitrarios), se quitará el auto de escena sin registrar su tiempo de viaje.

Finalmente cabe destacar que si bien no existen los semáforos con viraje, los vehículos están *implícitamente permitidos* a virar cuando estén en una intersección, dependiendo de si están en el primer carril de derecha a izquierda de la pista podrán doblar cuando exista una pista que siga por la derecha y viceversa para doblar a la izquierda, con la diferencia que esta vez se evaluará si el vehículo está en el último carril de derecha a izquierda.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

5.2.- Entradas

Se necesita contar con puntos que generan los vehículos y que en conjunto, se forme el fenómeno del tráfico. Es necesario tener en consideración que la aparición de vehículos sea lo suficientemente aleatoria como para hacer el sistema más caótico / dinámico como en la realidad.

Para lo anterior, se crea un GameObject que se ubica donde comienzan segmentos de pista, y por cada carril que tenga la pista comenzar a instanciar vehículos según un ciclo ajustado por el usuario y un porcentaje de probabilidad; estos son individuales para cada carril generador, por lo que no se puede predecir exactamente dónde y cuándo aparecerá un vehículo.

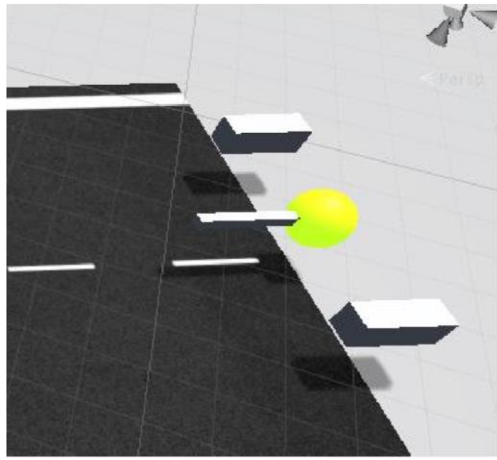

Ilustración 36: Gameobject que genera autos cada cierto tiempo

Los ciclos por carril son instanciados mediante una corrutina (ilustración 37) que recibe como parámetro un arreglo de objetos, donde [0: posición del spawn, 1: carril de spawn], por lo que se puede ejecutar para pistas con cualquier número de carriles.

Como en general los ciclos serán instanciados relativamente al mismo tiempo, se podría esperar algún tipo de sincronización cuando se comienzan a instanciar vehículos, por lo que la primera vez que se ejecuta la corrutina se genera un número aleatorio de 0 - 6 segundos (arbitrario) y se espera éste tiempo para ejecutar la tarea, esto para efectos de aumentar el azar que tienen los sistemas de tráfico.

Posterior a eso, se detecta si es que existe alguna colisión en los puntos generadores que pueda interferir en la creación de un nuevo agente, en caso de que sí exista una colisión con los *generadores,* estos se bloquean y no generan más vehículos hasta que la colisión se desactive. Luego, el ciclo aleatorio de aparición vehicular se describe de la siguiente forma:

Dado un ciclo T y una función R(x,y) que elige un numero al azar entre dos números tenemos:

$$(6)\quad Ciclo\ de\ aparición\ = R(T\ -\ 1, T\ +\ 2)\ :\ Donde\ T\ > 0\ y\ T\ < 100$$

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Si el ciclo de aparición es -1, los vehículos son instanciados siempre que tengan la posibilidad. Además, el ciclo debe contar con un tiempo de enfriamiento mínimo antes de instanciar un nuevo vehículo, que es de 0.5 segundos arbitrarios.
Ajustando estos ciclos por cada entrada del mapa, podemos esperar distintos resultados, como mayor congestión cuando el ciclo sea más cercano a 0 y por otro lado un flujo libre cuando T sea más cercano a 100.

```
6 referencias
IEnumerator SpawnV2(object []param) //[position to spawn, lane to spawn]
{
    yield return new WaitUntil(() => go.roadsReady);
    var init_wait = Random.Range(0f, 6f);

    yield return new WaitForSeconds(init_wait);
    while (true)
    {

        bool shouldSpawn = false;
        switch (param[1])
        {
            case 1: //left
                if(l.GetComponent<NotifyCollision>()!= null)
                    shouldSpawn = !l.GetComponent<NotifyCollision>().getIsBlocked();
                break;
            case 2: //center
                    shouldSpawn = !isBlocked;
                break;
            case 3: //right
                if (r.GetComponent<NotifyCollision>() != null)
                    shouldSpawn = !r.GetComponent<NotifyCollision>().getIsBlocked();
                break;
            default:
                shouldSpawn = false;
                break;
        }
        var spawnTime = Random.Range(spawnCycle - 1f, spawnCycle + 2f);
        var cooldown = 0.5f;

        if (shouldSpawn)
        {

           Instantiate(generateVehicle(),(Vector3)param[0],transform.rotation,transform);
            FindObjectOfType<DataManager>().vehiclesInScene++;
            FindObjectOfType<DataManager>().TotalVehiclesInstantiated++;
        }
        yield return new WaitForSeconds(spawnTime + cooldown);
    }
}
```

Ilustración 37: Corrutina que genera vehículos cada cierto tiempo. Es utilizada por cada carril que tenga una pista.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

5.3.- Salidas

Las salidas se ubican en todo nodo final de un segmento (ver ilustración 38), donde no existe ninguna otra pista que seguir. En estas locaciones, una instancia de Gameobject es creada, la que tendrá como objetivo destruir - y con ello limpiar - vehículos de la escena. Cada vehículo que colisione con este objeto registrará su tiempo total en escena dentro una lista que contiene todos los registros de tiempos de viaje, y con cada nuevo registro se calcula el nuevo promedio.

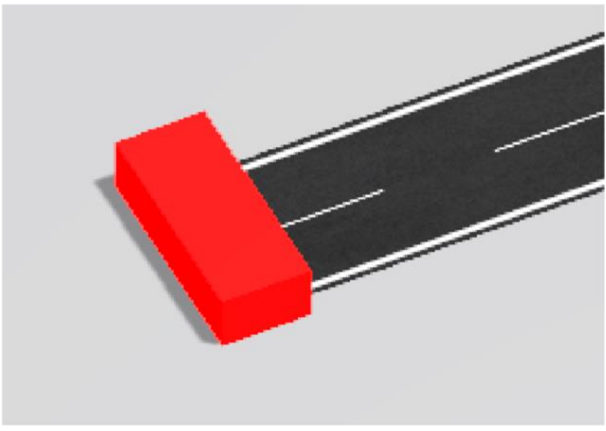

Ilustración 38: Flujo de salida que destruye vehículos y estima tiempos de viaje.

5.4.- Flujo

Para incorporar el concepto de flujo analizado en el primer capítulo a la simulación, se utilizan GameObjects en cada nodo que no sea uno de los anteriores como registrador de datos. En estricto rigor, se trata de una forma geométrica alargada que se sitúa de manera perpendicular al segmento. Recordemos la función de flujo:

$$(1) \quad q = (n * 3600)/t$$

donde

$$q = flujo\ vehicular\ (\,número\ de\ vehículos\ por\ hora)$$
$$n = número\ de\ vehículos\ pasando\ por\ una\ sección\ transversal\ del\ segmento\ en\ t\ segundos$$
$$t = intervalo\ de\ tiempo\ para\ los\ vehículos\ circulantes\ (s)$$

Como ya se tiene nuestra sección transversal, se debe poder contabilizar la cantidad de vehículos que pasan por la sección antes de que se complete un intervalo t. Para esto se recurre nuevamente a los colliders; se agrega un OnTriggerEnter que suma 1 al contador de vehículos. Cabe destacar que el flujo se actualiza conforme el tiempo t varía, es decir si por ejemplo se tiene t = 60, el flujo se actualizará cada 1 minuto lo que se refleja en la ilustración 39, donde a pesar de existir una congestión, los medidores en amarillo aún no se actualizan con el nuevo flujo.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Por último, cada vez que se calcula el flujo, el contador de vehículos se reinicia a 0 y suma su resultado a una lista que contiene todos los registros de flujos de la escena.

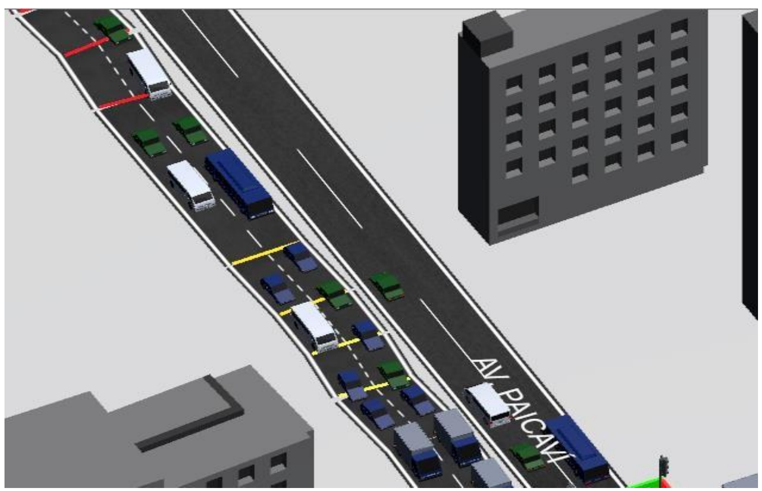

Ilustración 39: Registradores de flujo de tráfico: consisten en secciones transversales a la pista (en amarillo y rojo)

5.5.- Densidad

Por último, para estimar la densidad vehicular de un segmento recordemos la fórmula que se estableció en el primer capítulo:

$$(2) \quad Densidad\ k = (n * 1000)/L$$

en donde

- $k = vehículos\ por\ kilómetro$
- $n = número\ de\ vehículos\ presentes\ en\ un\ largo\ L\ de\ la\ pista$
- $L = el\ largo\ de\ una\ calle\ en\ metros.$

Para implementar este cálculo, es necesario que cada mesh de vías generadas tenga un controlador o script que permita calcular L y n. En el capítulo 2 se generó de manera procedural los triángulos que conforman cada pista; y tienen la misma dirección que la pista sobre el eje Z, por lo tanto necesitamos el largo de la totalidad de cada mesh en el eje Z. Para ello, Unity provee de accesibilidad a datos de mesh como sus dimensiones. Es importante destacar que este valor está en metros tal y como es requerida en la función anterior.
Por otro lado, para calcular el número de vehículos presentes se utilizan - nuevamente - colisionadores que se activan cuando un vehículo entra o sale de la pista, y luego suma un vehículo a n o lo resta, respectivamente.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Ilustración 40: Mesh de una vía seleccionada

5.6.- Pruebas

En esta sección, se analizan dos archivos OSM frente a 3 escenarios distintos: tráfico expedito, tráfico regular y tráfico congestionado. Para lograr estos distintos comportamientos nos basta con modificar el número de autos que ingresan a escena, esto es modificar el ciclo T del punto 4.2; con un T más cercano a 0 ingresarán más autos a la escena, mientras que con T más cercano a 100 se presume que se despejan las vías. Por lo que los tiempos de prueba se detalla en la siguiente ilustración:

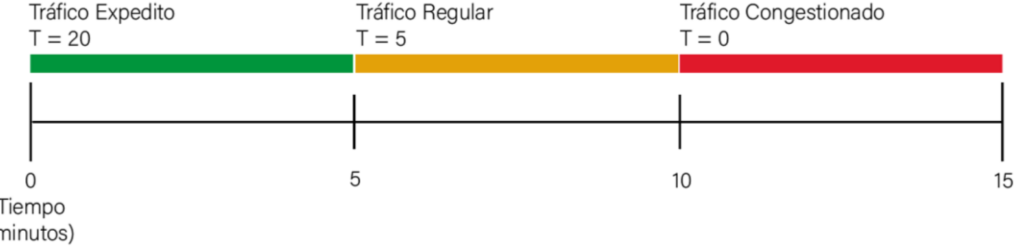

Ilustración 41: Distintas etapas con sus respectivos ciclos T para las pruebas

El primer archivo OSM contendrá la intersección entre O'higgins-Paicaví, el segundo archivo contendrá - en cambio - la rotonda Paicaví-Carrera. Estas áreas fueron seleccionadas arbitrariamente debido a que el sistema se creó utilizando estas zonas como referencia.
Para registrar los datos, se utilizó una librería gratuita descargada desde la tienda de Unity llamada **Grapher**. Grapher dispone de funciones que reciben valores para ser añadidos a una variable en la gráfica (en nuestro caso, flujo, densidad y tiempo de viaje). Una vez que se detiene el editor de Unity, la librería guarda en archivos *csv* separados los valores de cada variable registrados y permite - además - observar en tiempo real el comportamiento de cada variable, como se aprecia en la siguiente imagen:

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

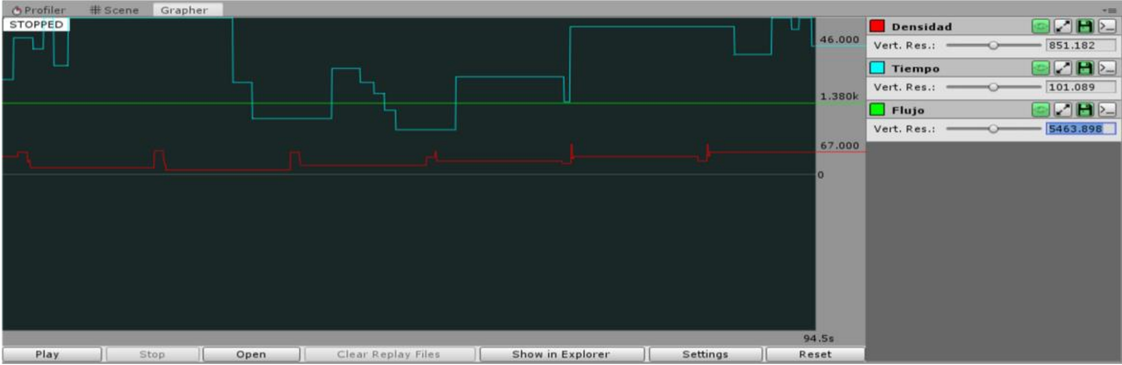

Ilustración 42: Grapher y registro de datos

5.6.1.- Paicaví - O'higgins

Esta intersección resulta ser ideal para poner a prueba el sistema de tráfico ya que todos los cálculos corresponden a un solo cruce, por lo tanto los resultados serán más específicos. En la siguiente imagen, se aprecian los distintos niveles de tráfico en la simulación para los distintos ciclos T.

Ilustración 43: Distintos niveles de tráfico observados en intersección

En general, el sistema se desempeña bien en términos de rendimiento aunque en la etapa de tráfico congestionado la simulación comenzó a tener un aumento en los tiempos de procesamiento y por ende la fluidez de cuadros por segundo se aprecia mermada (ver ilustración 44).

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

A continuación se presenta el rendimiento general para esta prueba específica, notemos cómo aumenta el uso del motor de físicas conforme aumenta el número de vehículos en escena. En general se mantiene alrededor de 1-5ms en las primeras dos etapas, sin embargo en la última el tiempo de procesamiento por cuadro alcanza hasta los ~16ms.

Tráfico Expedito

1ms (1000FPS)

Tráfico Regular

4ms (250FPS)

1ms (1000FPS)

Tráfico Congestionado

16ms (60FPS)

Ilustración 44: Perfil de rendimiento para intersección O'Higgins - Paicaví. En naranja uso de motor físico

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

A continuación los gráficos obtenidos a partir de la simulación.

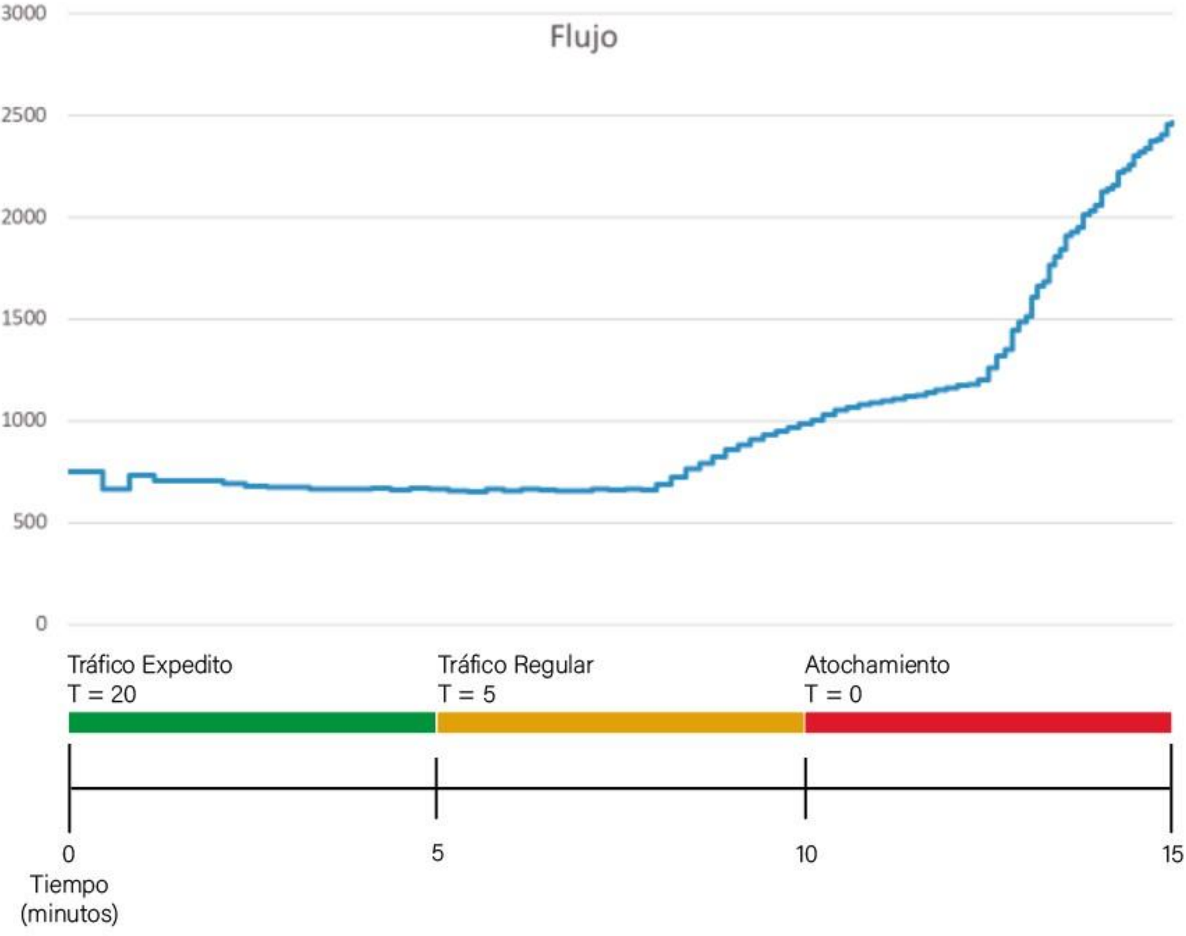

Ilustración 45: Flujo del sistema en distintos escenarios para O'Higgins-Paicaví

En la ilustración anterior se aprecia que el flujo llega a un equilibrio después de haber sido recién inicializado. Como era de esperar el flujo aumenta cada vez que más vehículos entran a escena, sin embargo cada vez que se modifica el valor de T los cambios sobre el flujo se ven afectados al tiempo después y esto es debido a que el flujo es el promedio de todos los detectores de flujo en un instante x y no todos los detectores reciben el mismo flujo.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

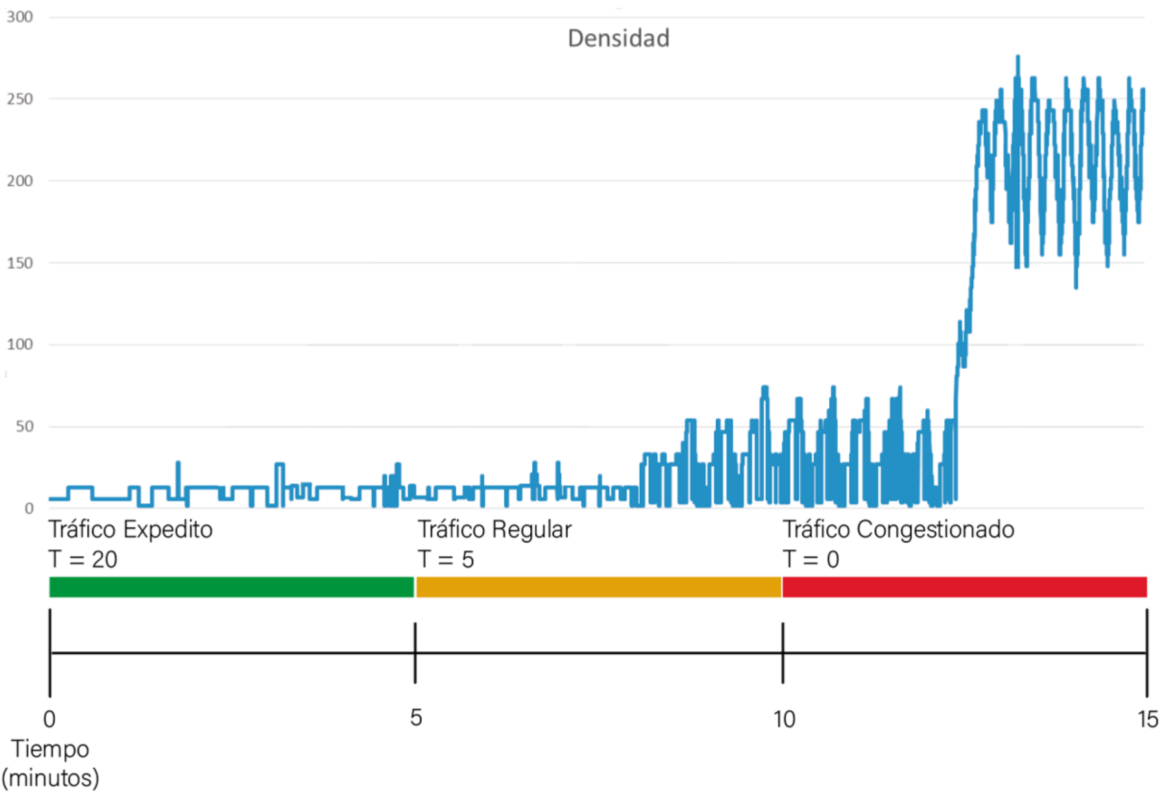

Ilustración 46: Gráfico de densidad promedio por cuadro durante la prueba

Para el caso de la densidad observamos que mantiene el mismo comportamiento que el flujo, donde el cambio sobre T se comienza a observar luego de un periodo sobre el índice de densidad. Se puede decir que la densidad también es inversamente proporcional al flujo entrante de vehículos; veamos cómo al final de la prueba la densidad llegó hasta los ~270 vehículos por kilómetro diferenciándose con creces del tráfico expedito y del tráfico regular.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

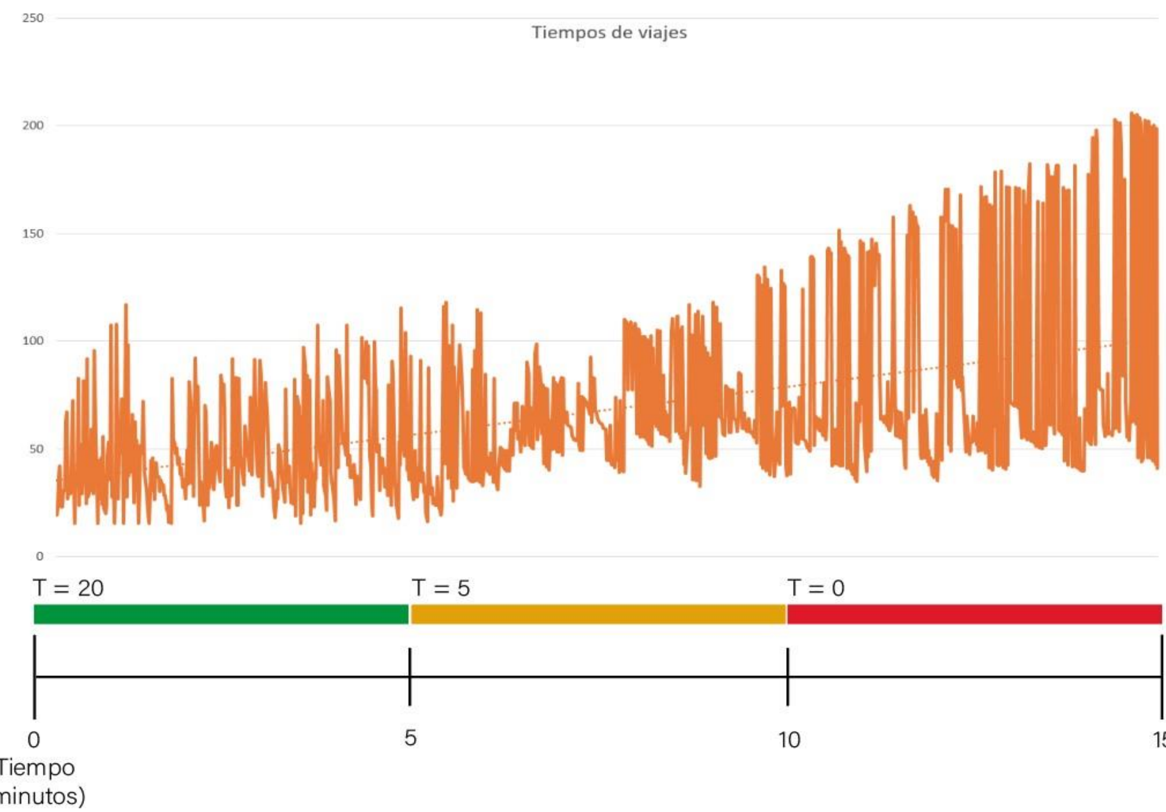

Ilustración 47: Gráfico que detalla tiempos de viaje por vehículo en la prueba

Por último, en la imagen anterior se describen los tiempos de viaje individuales por agente. Observemos que la tendencia fue a ir en aumento conforme cambiaban las etapas, sin embargo se aprecia una variación amplia lo que se explica con que los vehículos podían aparecer en tramos más largos del escenario (específicamente por O'Higgins) y por ende registrar más tiempo; también hay que considerar que los semáforos pueden contribuir al tiempo de espera de un agente particular. Si bien la tónica de los tiempos de viaje se mantiene irregular, sobre las últimas 2 etapas se verifica que – frente a estos escenarios – el tiempo de viaje promedio en esta zona tiende ir al alza.

Ilustración 48: Representación área total que comprende la zona

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

5.6.2.- Paicaví – Carrera

Si bien esta intersección resulta ser algo más complicada para recorrer desde el punto de vista de los agentes (debido a la rotonda), el sistema se desempeña bien en términos de comportamiento vehicular a la hora de simular un tráfico que recorriera esta zona. No obstante, al tener un flujo más alto el rendimiento se vio seriamente mermado debido al *tamaño* del mapa y la cantidad de vehículos que soportó en la prueba; tal y como se aprecia en la siguiente imagen los cuadros por segundo aumentaron conforme el flujo vehicular aumenta. En la primera etapa el tiempo de procesamiento por cuadro es ~1ms, en la segunda fluctúa entre 3~10 ms y en la última etapa el procesamiento llega a tomar hasta ~87ms.

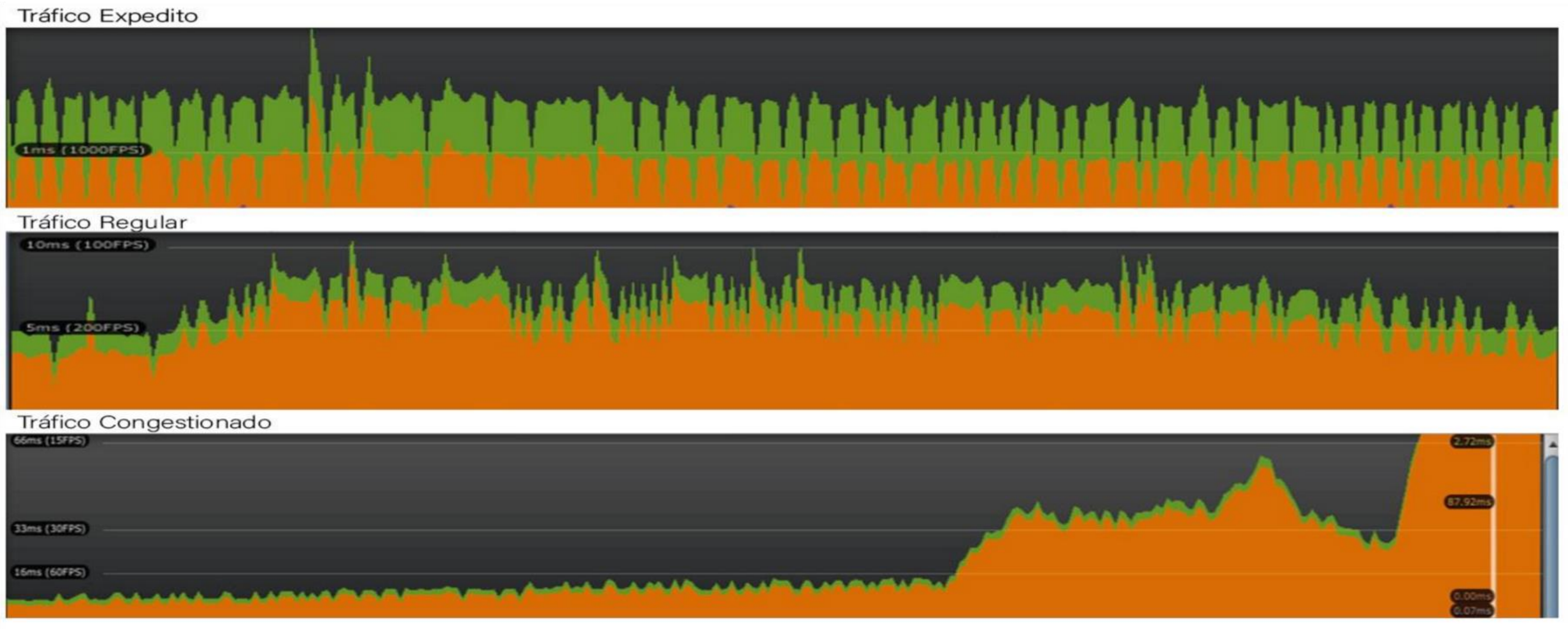

Ilustración 49: Perfiles de rendimiento para escenario Carrera - Paicaví frente a distintos ciclos T. En naranja uso del procesador de las físicas de los agentes.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Debido a lo anterior es que los agentes tienen un movimiento más restringido porque como captan la distancia por cuadro para tomar una decisión, la precisión en que realizan estas acciones disminuye y por ende causan más "accidentes" o interrupciones del flujo, lo que se puede observar en el flujo de tráfico cuando está congestionado; se aprecian vehículos fuera de lugar y en general una pista mucho más caótica.

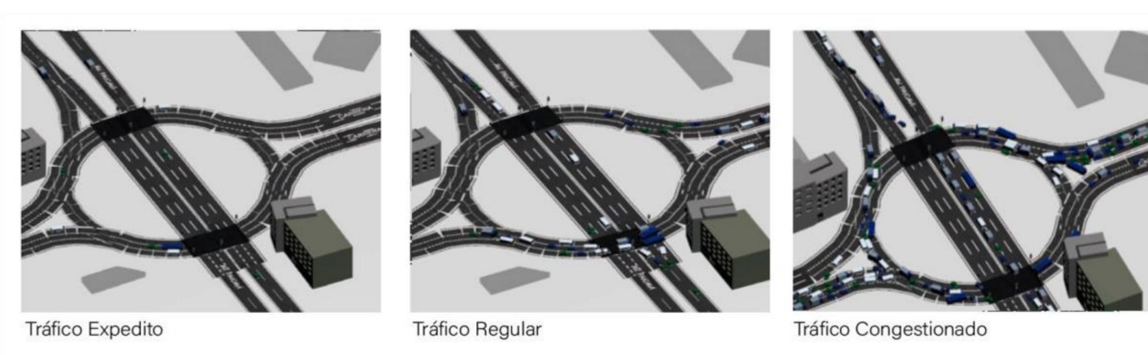

Ilustración 50: Distintos niveles de tráfico observados para esta sección

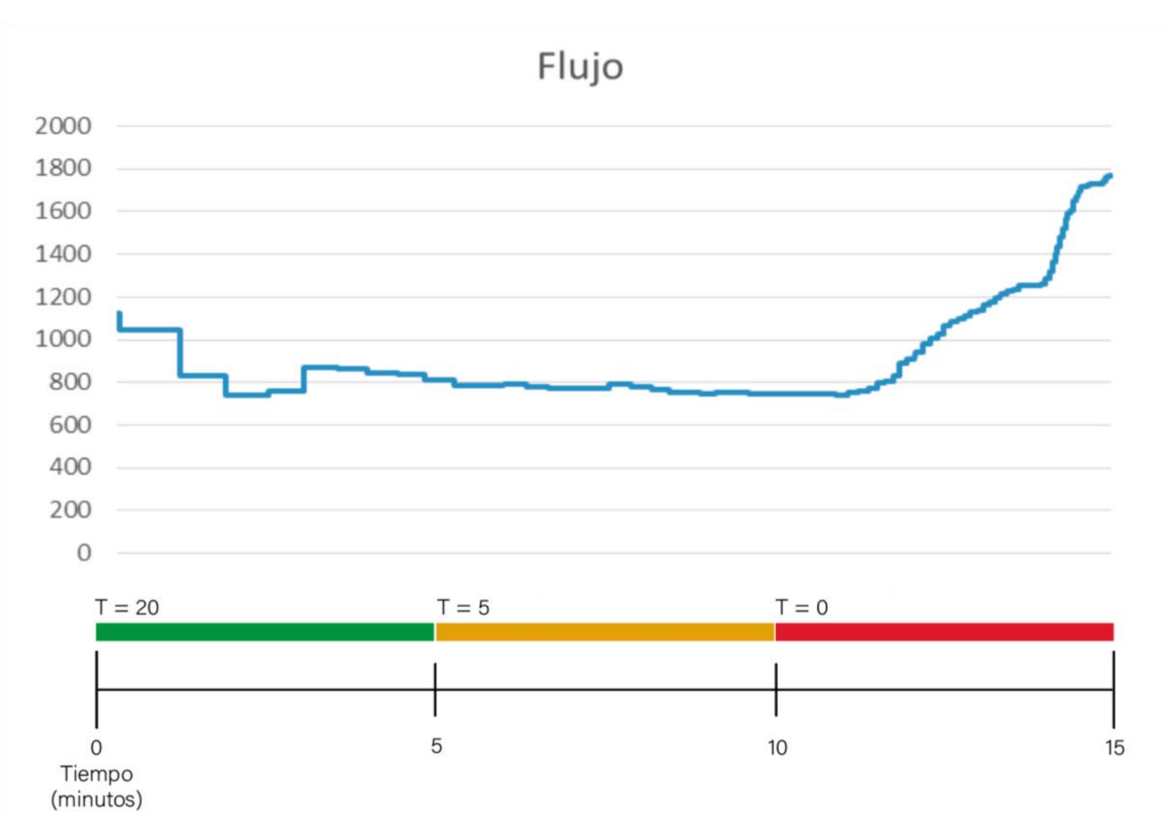

Ilustración 51: Gráfica de flujo durante la prueba

Al observar el flujo, se aprecia que en primera instancia, éste fluctúa hasta equilibrarse en las primeras dos etapas, mientras que en la tercera etapa de tráfico congestionado, vemos que aumenta dramáticamente el flujo vehicular. Esto se debe por el tamaño que

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

tiene la sección que se está analizando, lo que explica además por qué el flujo en esta área es menor que la zona anterior; como hay más detectores de tráfico, el índice promedio por cuadro tiende a equilibrarse cuando hay registradores con mucho flujo y otros con poco flujo de tráfico. No obstante, el alza del final del gráfico da cuenta que a pesar de lo anterior, el comportamiento del flujo siguió siendo inversamente proporcional al ciclo de ingreso vehicular.

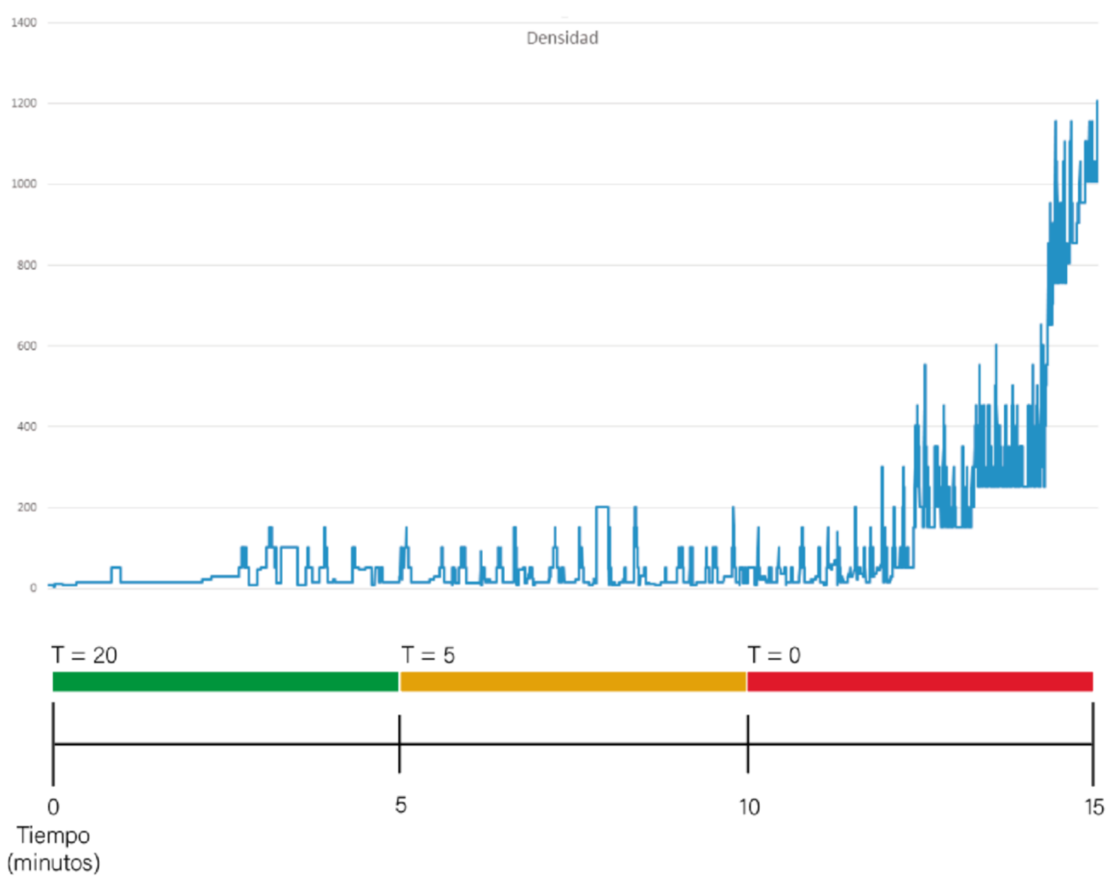

Ilustración 52: Densidad durante la prueba para este escenario

Para el caso de la densidad, a diferencia del primer mapa se llegaron a números mucho más altos de densidad debido al tamaño del mapa y que incluían más vías.
No obstante al igual que en el flujo, el mayor ascenso en el índice ocurrió en la última etapa mientras que en las primeras dos se mantuvo relativamente equilibrado, lo que – de nuevo – se explica porque la densidad por cuadro es el promedio de todas las densidades de tráfico por vía. De todas maneras, la densidad mantiene la relación inversamente proporcional con el tiempo T que define el flujo de tráfico.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

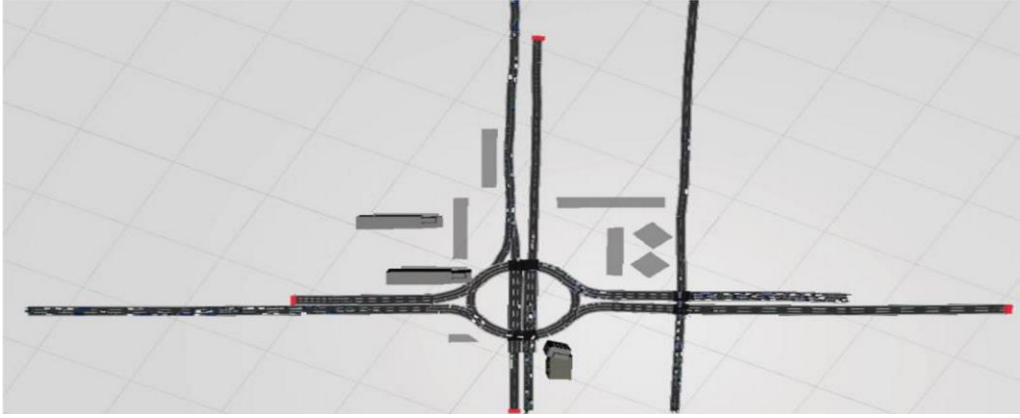

Ilustración 53: Área total que ocupa esta sección

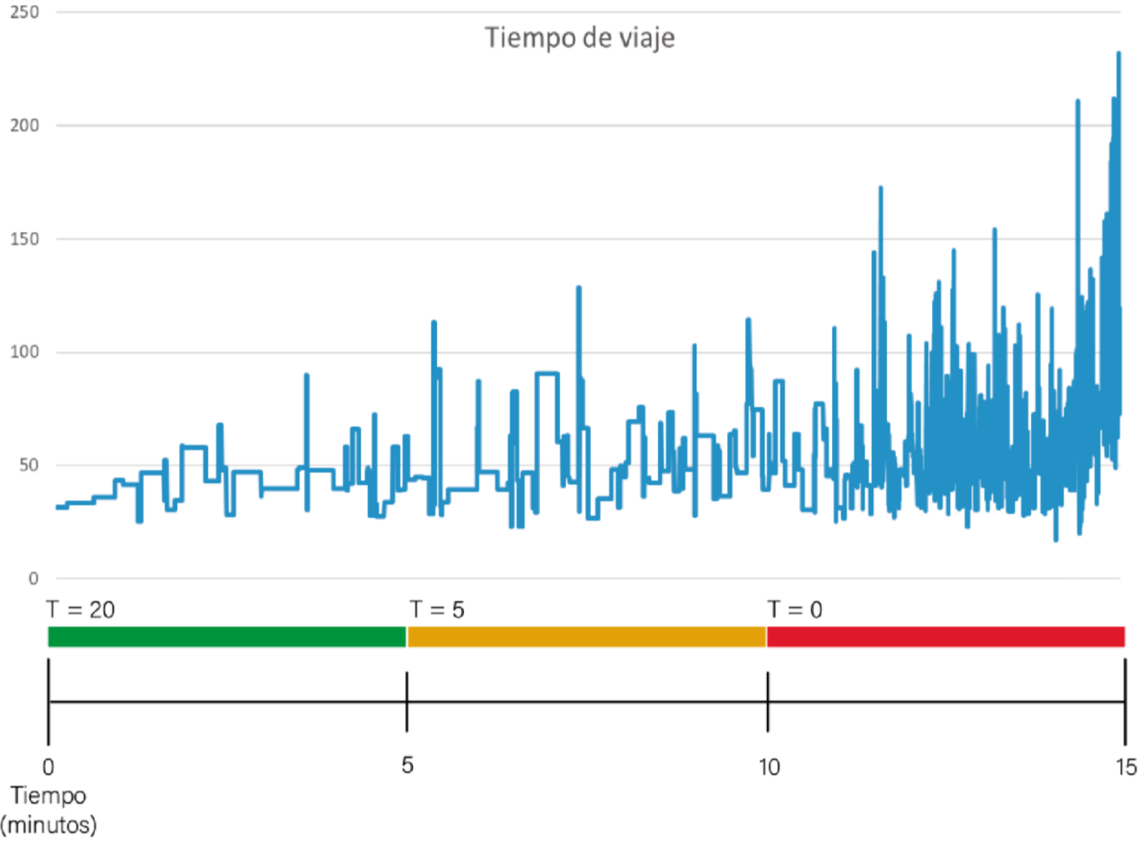

Ilustración 54: Gráfico del tiempo de viaje para cada vehículo en escenario de rotonda Paicaví

Finalmente, de los datos obtenidos para el tiempo de viaje particular se observa que si bien existe una variación, existe una tendencia al alza conforme el ciclo T aumenta. Notemos además como en el último tramo existen mucho más datos que en las dos primeras etapas que es debido a un mayor número de agentes vehiculares en ese instante.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

5.6.3.- Conclusiones finales

Acerca del proyecto se puede concluir que se logró el objetivo de simular correctamente el fenómeno de flujo de tráfico en un ambiente 3D. Fue posible observar distintos niveles de tráfico sobre una infraestructura que formaban los agentes vehiculares. Mediante esta construcción modelo-representativa se logró estimar valores como la densidad, flujo de tráfico por hora y tiempo de viaje vehicular.
Con respecto a las pruebas, se consiguió observar como los tres indicadores reaccionaban frente a distintos flujos vehiculares tanto de forma analítica como de forma visual (en el simulador) en los dos mapas. También podemos establecer que el ciclo de aparición T es inversamente proporcional a los tres indicadores, lo que en otras palabras significa que un mayor flujo entrante implica naturalmente un aumento sobre la densidad, flujo vehicular y tiempos de espera al cruzar un sector.
El sistema de tráfico desarrollado logra simular el fenómeno de tráfico vehicular y su comportamiento en distintas variables, gracias al motor de físicas Unity3D se puede continuar con el desarrollo del sistema siendo posible agregar más lógicas que modifiquen el comportamiento de la simulación, a fin de especificar un área de estudio y apoyar estos estudios mediante la simulación. Como línea futura se puede implementar la inteligencia artificial mediante el método aprendizaje continuo machine learning, donde su aplicación puede ser variada; por ejemplo una aplicación del machine learning sobre los agentes vehiculares nos permitiría simular decisiones individuales según cómo se comportan los humanos, permitiendo un comportamiento aún más fiel a la realidad y según esto poder establecer predicciones de accidentes, atascos entre otros gracias a que se cuenta con una estructura sobre la cual guiarse. También se podría agregar la posibilidad de incluir aspectos psicológicos individuales para añadir el cambio de carril y diferenciar entre conductores calmados y los más agresivos.
Otro hilo que implicó el desarrollo del sistema fue la optimización de recursos a la hora de implementar un sistema que simula muchas partículas (agentes vehiculares) con sus propias decisiones, por lo que en un futuro este tema puede ser abordado para que la cantidad de vehículos en escena, no perjudique el rendimiento de la velocidad de la simulación y asimismo se mantenga el orden vehicular a pesar de una pérdida de frames.

Respecto a la restricción de tiempo (relativamente 1 año) se puede concluir que fue un plazo suficiente para realizar un simulador de flujo vehicular en su forma básica (esto ya que se podría continuar con ramas futuras). En el quinto y actual capítulo pudimos ver en detalle el funcionamiento general del sistema, que nos permitió dar paso a las pruebas y con ello fuimos capaces de concluir distintos resultados en este mismo capítulo.

Por último, se concluye que el proyecto consideró distintos tópicos de las ciencias de la computación como el paradigma orientado a objetos que considera un sistema sobre un motor gráfico, el estudio de cómo funcionan los gráficos para proyectar una imagen, y en general distintas técnicas lógicas a la hora de construir un software, lo que es suficiente para concluir que la informática se puede aplicar para cualquier rama de estudio conocido, mediante su correcta aplicación, es posible apoyar en áreas que se requiera realizando aplicaciones según sea el objetivo final de un proyecto.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Referencias

[1] (Von Neumann, J. (1995), "Method in the physical sciences", en Bródy F., Vámos, T. (editores), The Neumann Compendium, World Scientific, p. 628; previamente publicado en The Unity of Knowledge, editado por L. Leary (1955), pp. 157-164, y Tambien en John von Neumann Collected Works, editado por A. Taub, Volume VI, pp. 491-498.)

[2] Treiber, M., & Kesting, A. (2013). Traffic flow dynamics: Data, models and simulation. Heidelberg: Springer.)

[3] Elon Musk, J.Rogan [PowerfulJRE]. (6 de septiembre de 2018). Joe Rogan Experience #1169 - Elon Musk [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=ycPr5-27vSI.

[4] (Fallas, J. (2003). Proyecciones cartográficas y Datum ¿Qué son y para qué sirven? Laboratorio de Teledetección y Sistemas de Información Geográfica PRMVS-EDECA. Universidad Nacional. Heredia, Costa Rica.

[5] CNN Chile (28 de diciembre de 2019). Cifra de muertes en accidentes de tránsito asciende a 1.555 durante 2019. Recuperado de https://www.cnnchile.com/pais/muertes-accidentes-de-transito-2019_20191228/

[6] Unity Technologies (1 de septiembre de 2020). Unity – Scripting API: Physics.Raycast. Recuperado de https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Anexos: Glosario

a. Renderizar: se refiere al proceso de generar imagen fotorrealista o no fotorrealista a partir de un modelo 2D o 3D (o modelos en lo que colectivamente podría llamarse un archivo de escena) por medio de programas informáticos.
b. Ray Cast: se refiere a una técnica donde se proyecta un rayo hasta que intersecta sobre una superficie que puede entregar información del objeto al que pertenece esta superficie, entre otros.
c. Mesh (Mallas poligonales): es una superficie creada mediante un método tridimensional generado por sistemas de vértices posicionados en un espacio virtual con datos de coordenadas propios.
d. UV Mapping: consiste en aplicar texturas 2D a superficies que conforman una figura tridimensional.
e. LINQ: es un conjunto de extensiones integradas en el lenguaje C#, que nos permite trabajar de manera cómoda y rápida con colecciones de datos, como si de una base de datos se tratase.
f. Gameobject: es el concepto más importante en el Editor de Unity. Cada objeto en su juego es un GameObject, desde personajes y objetos coleccionables hasta luces, cámaras y efectos especiales.
g. Coroutines: Una coroutine (corrutina) es una función que puede suspender su ejecución (yield) hasta que la instrucción sobre el yield se ejecuta.
h. Colliders (Colisionadores): definen la forma de un objeto para los propósitos de colisiones físicas. Un collider, el cual es invisible, necesita no estar con la misma forma exacta que el mesh del objeto y de hecho, una aproximación a menudo es más eficiente e indistinguible en el juego.

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Anexos: Clases principales implementadas

Lectura de archivo OSM y generación procedural

Clase MapReader

```
using System.Collections.Generic;
using System.Xml;
using UnityEngine;
7 referencias
class MapReader : MonoBehaviour
{
    public Dictionary<ulong, OsmNode> nodes;
    public List<OsmWay> ways;
    public OsmBounds bounds;
    public GameObject groundPlane;

    [Tooltip("El recurso txt que contiene los datos OSM")]
    public string resourceFile;

    3 referencias
    public bool IsReady { get; private set; }

    // Use this for initialization
    0 referencias
    void Start()
    {
        System.Globalization.CultureInfo customCulture = (System.Globalization.CultureInfo)System.Threading.Thread.CurrentThread.CurrentCulture.Clone();
        customCulture.NumberFormat.NumberDecimalSeparator = ".";
        System.Threading.Thread.CurrentThread.CurrentCulture = customCulture;
        Time.timeScale = 1f;
        nodes = new Dictionary<ulong, OsmNode>();
        ways = new List<OsmWay>();

        var txtAsset = Resources.Load<TextAsset>(resourceFile);

        XmlDocument doc = new XmlDocument();
        doc.LoadXml(txtAsset.text);

        SetBounds(doc.SelectSingleNode("/osm/bounds"));
        GetNodes(doc.SelectNodes("/osm/node"));
        GetWays(doc.SelectNodes("/osm/way"));
        float minx = (float)MercatorProjection.lonToX(bounds.MinLon);
        float maxx = (float)MercatorProjection.lonToX(bounds.MaxLon);
        float miny = (float)MercatorProjection.latToY(bounds.MinLat);
        float maxy = (float)MercatorProjection.latToY(bounds.MaxLat);
        groundPlane.transform.localScale = new Vector3((maxx - minx) / 2, 1, (maxy - miny) / 2);
        IsReady = true;     //determina cuando se comienzan a generar elementos del sistema de tráfico
    }

    1 referencia
    void GetWays(XmlNodeList xmlNodeList)
    {
        foreach (XmlNode node in xmlNodeList)
        {
            OsmWay way = new OsmWay(node);
            ways.Add(way);
        }
    }

    1 referencia
    void GetNodes(XmlNodeList xmlNodeList)
    {
        foreach (XmlNode n in xmlNodeList)
        {
            OsmNode node = new OsmNode(n);
            nodes[node.ID] = node;
        }
    }

    1 referencia
    void SetBounds(XmlNode xmlNode)
    {
        bounds = new OsmBounds(xmlNode);
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Clase Road Maker

```
class RoadMaker : InfrastructureBehaviour
{
    public Material roadMaterial;
    private PathFinder script;
    private int _globalWayCount = 0;
    public Material _1L;
    public Material _2L;
    public Material _3L;

    public GameObject flowSegment;


    public bool roadsReady = false;
    /// <summary> Create the roads.
    0 referencias
    IEnumerator Start()
    {
        // Wait for the map to become ready
        while (!map.IsReady)
        {
            yield return null;
        }
        script = PathFinder.instance;
        // Iterate through the roads and build each one
        foreach (var way in map.ways.FindAll((w) => { return w.IsRoad; }))
        {
            var roadTexture = roadMaterial;
            switch (way.Lanes)
            {
                case (1):
                    roadTexture = _1L;
                    break;
                case (2):
                    roadTexture = _2L;
                    break;
                case (3):
                    roadTexture = _3L;
                    break;
                case (4):
                    roadTexture = _3L;
                    break;
                default:
                    roadTexture = _1L;
                    break;
            }

            CreateObject(way, roadTexture, way.Name);
            createRoads(way);
            yield return null;
        }
        roadsReady = true;
    }
```

```
// check if a point overlaps another
1 referencia
public int getID(Vector3 point)
{
    Node nearestNode = null;

    foreach (var node in script.graphData.nodes)
    {
        if (node.Position == point)
        {
            nearestNode = node;
            return node.autoGeneratedID;
        }
    }

    return -1;
}

//creates a set of Qpathfinder nodes in a given way
2 referencias
void AddNode(Vector3 position, OsmNode node,int addIndex = -1 ){
    Node nodeAdded = new Node(position);

    nodeAdded.isTrafficLights = node.isTrafficSignals;
    if (addIndex == -1)
        script.graphData.nodes.Add(nodeAdded);
    else
        script.graphData.nodes.Insert(addIndex, nodeAdded);

    script.graphData.ReGenerateIDs();

   // QPathFinder.Logger.LogInfo("Node with ID:" + nodeAdded.autoGeneratedID + " Added!");
}

//we're going to use this counter to keep track of
private int nodesBuilt = 0;
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
// OVERLAP : cuando queremos crear un nodo donde ya existe un nodo anterior.
1 referencia
private void createRoads(OsmWay way)
{
    int previousNode = 0;
    bool isOneWay = way.isOneWay;
    string name = way.Name;
    int lanes = way.Lanes;

    for (int i = 0; i < way.NodeIDs.Count; i++)
    {
        OsmNode p1 = map.nodes[way.NodeIDs[i]];
        Vector3 ipos = p1 - map.bounds.Centre;
        //overlapId nos dira si existe algun nodo en la ubicacion que decimos, -1 si no encontro ninguna, "id" si se logró
        int overlapId = getID(ipos);

        if (i !=0)  //significa que estamos en cualquier nodo >1 por lo que podemos construir path al anterior
        {
            if (overlapId == -1)  //no hubo overlap, podemos crear el nodo
            {
                AddNode(ipos,p1, -1);
                nodesBuilt++;
                if (previousNode == 0)
                {
                    var currentPathCount = script.graphData.paths.Count;
                    AddPath(-1, nodesBuilt -1, nodesBuilt,name,isOneWay,lanes);
                }
                else
                {
                    var currentPathCount = script.graphData.paths.Count;
                    AddPath(-1, previousNode, nodesBuilt, name, isOneWay, lanes);
                    previousNode = 0;
                }

            }
            else //hubo overlap, creamos el path desde el nodo anterior a el overlapId
            {
                if (previousNode == 0)
                {
                    var currentPathCount = script.graphData.paths.Count;
                    AddPath(-1, nodesBuilt, overlapId, name, isOneWay, lanes);
                    previousNode = overlapId;
                }
                else
                {
                    var currentPathCount = script.graphData.paths.Count;
                    AddPath(-1, previousNode, overlapId, name, isOneWay, lanes);
                    previousNode = overlapId;
                }
            }
        }
        else        //Estamos en el primer nodo, solo colocar referencia a el.
        {
            if (overlapId == -1){
                AddNode(ipos,p1,-1);
                nodesBuilt++;
            } else
            {
                previousNode = overlapId;
            }
        }
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
3 referencias
protected override void OnObjectCreated(OsmWay way, Vector3 origin, List<Vector3> vectors, List<Vector3> normals, List<Vector2> uvs, List<int> indices)
{
    for (int i = 1; i < way.NodeIDs.Count; i++)
    {

        OsmNode p1 = map.nodes[way.NodeIDs[i - 1]];
        OsmNode p2 = map.nodes[way.NodeIDs[i]];

        Vector3 s1 = p1 - origin;
        Vector3 s2 = p2 - origin;
        Vector3 diff = (s2 - s1).normalized;
        // https://en.wikipedia.org/wiki/Lane
        // According to the article, it's 3.7m in Canada
        int lanes = 0;
        if (way.Lanes > 3) {  lanes = 3; }
        else { lanes = way.Lanes; }

        var cross = Vector3.Cross(diff, Vector3.up) * 2.5f * lanes;

        // Create points that represent the width of the road
        Vector3 v1 = s1 + cross;
        Vector3 v2 = s1 - cross;

        Vector3 v3 = s2 + cross;
        Vector3 v4 = s2 - cross;

        vectors.Add(v1);
        vectors.Add(v2);
        vectors.Add(v3);
        vectors.Add(v4);



        uvs.Add(new Vector2(0, 0));
        uvs.Add(new Vector2(1, 0));
        uvs.Add(new Vector2(0, 1));
        uvs.Add(new Vector2(1, 1));

        normals.Add(Vector3.up);
        normals.Add(Vector3.up);
        normals.Add(Vector3.up);
        normals.Add(Vector3.up);

        int idx1, idx2, idx3, idx4;
        idx4 = vectors.Count - 1;
        idx3 = vectors.Count - 2;
        idx2 = vectors.Count - 3;
        idx1 = vectors.Count - 4;

        // first triangle v1, v3, v2
        indices.Add(idx1);
        indices.Add(idx3);
        indices.Add(idx2);

        // second         v3, v4, v2
        indices.Add(idx3);
        indices.Add(idx4);
        indices.Add(idx2);
    }
}
```

```
4 referencias
void AddPath(int addIndex = -1, int from = -1, int to = -1,string nom ="", bool isOneway=true, int lanes=0)
{
    if (from != -1 && to != -1)
    {
        if (from == to)
        {
          // QPathFinder.Logger.LogError("Preventing from adding Path to the same node.");
            return;
        }
        Path pd = script.graphData.GetPathBetween(from, to);
        if (pd != null)
        {
            QPathFinder.Logger.LogError("We already have a path between these nodes. New Path not added!");
            return;
        }
    }
    Path newPath = new Path(from, to,nom,isOneway,lanes);
    if (addIndex == -1)
        script.graphData.paths.Add(newPath);
    else
        script.graphData.paths.Insert(addIndex, newPath);
    script.graphData.ReGenerateIDs();

    QPathFinder.Logger.LogInfo("Path with ID:" + newPath.autoGeneratedID + " Added");
}
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Inteligencia artificial

Clase SimpleCarAi – Determina comportamiento individual de los agentes

```
2 referencias
public class SimpleCarAi : MonoBehaviour
{
    public float speed = 10f;
    public float torque = 1f;
    public bool isBraking = false;
    private Path _track;
    private List<Vector3> route;
    private int currentNode = 0;
    public int currentLane = 0;
    public float dtc;


    private Vector3 _closestTrackPoint = Vector3.positiveInfinity;
    RoadMaker go;
    [HideInInspector]
    public PathFinder script;

    private int score = 0;
    private float timeSinceStart = 0f;
    public float timeAlive = 0f;

    [Header("Sensores")]
    public float sensorLenght = 5f;
    public Vector3 frontSensorPosition = new Vector3(0f, 0.2f, 0.5f);
    public float frontSideSensorPos = 0.2f;
    public float frontSensorAngle = 30f;
    private bool turnDecider = true;

    0 referencias
    private void Start()
    {
        script = PathFinder.instance;
        go = GameObject.Find("Map").GetComponent<RoadMaker>();

        StartCoroutine(KeepTrack());
        StartCoroutine(ShouldTurn());
        timeSinceStart = Time.time;
    }

    1 referencia
    IEnumerator ShouldTurn()
    {
        var time = Random.Range(1f, 5f);
        while (true)
        {

            if (Random.value > 0.5)
            {
                turnDecider = true;
            }
            else turnDecider = false;
                yield return new WaitForSeconds(time);
        }
    }
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
0 referencias
public Path getTrack()
{
    return _track;
}

1 referencia
IEnumerator KeepTrack()
{
    yield return new WaitUntil(() => go.roadsReady);

    while (true) {
        var prevTrack = _track;
        _track = script.graphData.getClosestPath(transform.position);

        if (_track != null && go.roadsReady == true)
        {
            var a = script.graphData.nodes[_track.IDOfA - 1].Position;
            var b = script.graphData.nodes[_track.IDOfB - 1].Position;
            var dir = (b - a);
           var angle = Vector3.Angle(transform.forward, dir);

          //  var angle = Vector3.SignedAngle(transform.forward,dir,Vector3.up);
            if (prevTrack != null)
            {
                if(angle > 75f)
                {
                    if (!(currentLane == 1 && !turnDecider))
                    {
                        _track = prevTrack;
                        a = script.graphData.nodes[_track.IDOfA - 1].Position;
                        b = script.graphData.nodes[_track.IDOfB - 1].Position;
                    }

                }
            }

            _closestTrackPoint = script.graphData.ClosestPointOnLine(a, b, transform.position);

        }
        yield return null;
    }
}

0 referencias
public void SetRoute(List<Vector3> wayNodes)
{
    route = wayNodes;
}

//we are keeping the car separated in lanes
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
1 referencia
private void KeepDistance()
{
    if (_track != null)
    {
        //var angle = Mathf.Atan2(b.y-a.y,b.x-a.x)*180f/Mathf.PI;
        //var angle = AngleDir(transform.forward, _closestTrackPoint, Vector3.up);
        //var angle = (b.x - a.x) * (transform.position.y - a.y) - (b.y - a.y) * (transform.position.x - a.x);
        //var angle = Vector3.SignedAngle(transform.forward, _closestTrackPoint, Vector3.up);
        Vector3 localPos = transform.InverseTransformPoint(_closestTrackPoint);
        var dist = (transform.position - _closestTrackPoint).sqrMagnitude;
        var angle = 30f;
        bool side = false; // true if right, false if left
        if (localPos.x < 0.00f) side = false;
        else if (localPos.x > 0.00f) side = true;

        //determine which lane we should be driving on

        currentLane = whichLaneIsMyLane(_track.lanes, side, dist);

        RotationV2(currentLane, side, dist);


    }
}


//bool side : true for right - false left
1 referencia
private int whichLaneIsMyLane(int _thisTrackLanes, bool side, float distanceToCenter)
{
    if (_thisTrackLanes == 1) return 1;
    if (_thisTrackLanes == 2)
    {
        if (side)
        {
            return 2;
        }
        else
        {
            return 1;
        }
    }
    if (_thisTrackLanes == 3 || _thisTrackLanes==4)
    {
        if (distanceToCenter < 5f) return 2;
        if (distanceToCenter < 100f)
        {
            if (side) return 3;
            else return 1;
        }
    }
    return 0;

}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
1 referencia
private void Sensor()
{
    RaycastHit hit;
    Vector3 sensorStartPos = transform.position;
    sensorStartPos += transform.forward * frontSensorPosition.z;
    sensorStartPos += transform.up * frontSensorPosition.y;
    if (Physics.Raycast(sensorStartPos, transform.forward, out hit, sensorLenght * 1.5f))
    {
        if (hit.collider.CompareTag("Intersection"))
        {
            if (hit.collider.GetComponent<lightsState>() != null)
            {
                if (!onIntersection)
                {
                    Debug.DrawLine(sensorStartPos, hit.point);
                    if (hit.collider.GetComponent<lightsState>().state == -1) isBraking = true;
                    if (hit.collider.GetComponent<lightsState>().state == 0) isBraking = true;
                    if (hit.collider.GetComponent<lightsState>().state == 1) isBraking = false;
                    return;
                }
                else
                {
                    isBraking = false;
                }
                return;
            }

        }
        else if (hit.collider.CompareTag("Vehicle"))
        {
            isBraking = true;
            return;
        }
    }
    //front right sensor
    sensorStartPos += transform.right * frontSideSensorPos *2f;
    if (Physics.Raycast(sensorStartPos, transform.forward, out hit, sensorLenght*1.5f))
    {
        if (hit.collider.CompareTag("Intersection"))
        {
            if (hit.collider.GetComponent<lightsState>() != null)
            {
                if (!onIntersection)
                {
                    Debug.DrawLine(sensorStartPos, hit.point);
                    if (hit.collider.GetComponent<lightsState>().state == -1) isBraking = true;
                    if (hit.collider.GetComponent<lightsState>().state == 0) isBraking = true;
                    if (hit.collider.GetComponent<lightsState>().state == 1) isBraking = false;
                    return;
                }
                else
                {
                    isBraking = false;
                }
                return;
            }

        }
        else if (hit.collider.CompareTag("Vehicle"))
        {
            isBraking = true;
            return;
        }
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
        }
        //front left  sensor
        sensorStartPos -= transform.right * frontSideSensorPos * 2;
        if (Physics.Raycast(sensorStartPos, transform.forward, out hit, sensorLenght*1.5f))
        {
            if (hit.collider.CompareTag("Intersection"))
            {
                if (hit.collider.GetComponent<lightsState>() != null)
                {
                    if (!onIntersection)
                    {
                        Debug.DrawLine(sensorStartPos, hit.point);
                        if (hit.collider.GetComponent<lightsState>().state == -1) isBraking = true;
                        if (hit.collider.GetComponent<lightsState>().state == 0) isBraking = true;
                        if (hit.collider.GetComponent<lightsState>().state == 1) isBraking = false;
                        return;
                    }
                    else
                    {
                        isBraking = false;
                    }
                    return;
                }

            }
            else if (hit.collider.CompareTag("Vehicle"))
            {
                isBraking = true;
                return;
            }
        }

        isBraking = false;
    }

    private bool rotateV2 = true;
    1 referencia
    private void RotationV2(int currentLane, bool side, float distanceToCenter)
    {
        dtc = distanceToCenter;
        if( !isBraking) {
        Vector3 a = script.graphData.nodes[_track.IDOfA - 1].Position;
        Vector3 b = script.graphData.nodes[_track.IDOfB - 1].Position;
        Vector3 dir = (b - a);
        var cross = Vector3.Cross(transform.forward, dir);
        if (_track.lanes == 1)
        {
            if (currentLane == 1)
            {
                //1 pista - AI en pista 1
                if (cross.y > -0.01f && cross.y < 0.01f) return;      //  <--- "same" direction, dont rotate
                if (distanceToCenter > -0.5f && distanceToCenter<0.5f) {
                    if ((cross.y < -0.01f || cross.y > 0.01f))
                        transform.rotation = Quaternion.LookRotation(dir);
                    //rotateV2 = false;
                    return;
                }

                //rotateV2 = true;
                if (side)
                {
                        if (distanceToCenter > 0.5f)
                        {

                            transform.Rotate(new Vector3(0f, 1f, 0f));
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
                    return;
                }

            }
            else
            {
                if (distanceToCenter > 0.5f)
                {
                    transform.Rotate(new Vector3(0f, -1f, 0f));

                    return;
                }

            }
        }
        else return;

    } else if (_track.lanes == 2)
    {
        if (currentLane == 1)
        {
            if (cross.y > -0.01f && cross.y < 0.01f) return;
            if (distanceToCenter > 5.5f && distanceToCenter <6.4f)
            {
                if (cross.y < -0.01f || cross.y > 0.01f)
                    transform.rotation = Quaternion.LookRotation(dir);
                return;
            }

            if (!side)
            {
                if(distanceToCenter<=5.9f ) transform.Rotate(new Vector3(0f, 1f, 0f));
                if(distanceToCenter>=6.1f ) transform.Rotate(new Vector3(0f, -1f, 0f));
                return;
            }
        }
        else if (currentLane == 2)
        {

            if (cross.y > -0.01f && cross.y < 0.01f) return;
            if (distanceToCenter > 5.5f && distanceToCenter<6.4f)
            {
                if (cross.y < -0.01f || cross.y > 0.01f)
                    transform.rotation = Quaternion.LookRotation(dir);
                return;
            }

            if (side)
            {
                if (distanceToCenter <= 5.9f) transform.Rotate(new Vector3(0f, -1f, 0f));
                if (distanceToCenter >= 6.1f) transform.Rotate(new Vector3(0f, 1f, 0f));
                return;
            }
        }
        else return;

    }else if (_track.lanes == 3 || _track.lanes==4)
    {
        if (currentLane == 1)
        {
            if (cross.y > -0.01f && cross.y < 0.01f) return;
            if (distanceToCenter > 17.5f && distanceToCenter <18.5f)
            {
                if (cross.y < -0.01f || cross.y > 0.01f)
                    transform.rotation = Quaternion.LookRotation(dir);
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
                    return;
                }

                if (!side)
                {
                    if(distanceToCenter>=18.5f) transform.Rotate(new Vector3(0f, -1f, 0f));
                    if(distanceToCenter<=17.5f) transform.Rotate(new Vector3(0f,  1f, 0f));
                    return;
                }
            }
            else if (currentLane == 2)
            {
                if (cross.y > -0.01f && cross.y < 0.01f) return;
                if (distanceToCenter > -0.5f && distanceToCenter < 0.5f)
                {
                    if (cross.y < -0.01f || cross.y > 0.01f)
                        transform.rotation = Quaternion.LookRotation(dir);
                    return;
                }

                if (side)
                {
                    transform.Rotate(new Vector3(0f, 1f, 0f));
                }
                else
                {
                    transform.Rotate(new Vector3(0f, -1f, 0f));
                }
            }
            else if( currentLane == 3)
            {
                if (cross.y > -0.01f && cross.y < 0.01f) return;
                if (distanceToCenter > 17.5f && distanceToCenter < 18.5f)
                {
                    if (cross.y < -0.02f || cross.y > 0.02f)
                        transform.rotation = Quaternion.LookRotation(dir);
                    return;
                }

                if (side)
                {
                    if(distanceToCenter>=18.5f) transform.Rotate(new Vector3(0f, 1f, 0f));
                    if(distanceToCenter<=17.5f) transform.Rotate(new Vector3(0f, -1f, 0f));
                    return;
                }
            }
        }
    }
}

private bool rotating = false;
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
0 referencias
private void FixedUpdate()
{
    timeAlive = Time.time - timeSinceStart;
    Drive();
    KeepDistance();
    Sensor();
    if (transform.position.y > 3f)
    {
        Destroy(gameObject, 0.7f);
        // FindObjectOfType<DataManager>().vehiclesInScene--;
    }

    if (onIntersection && isBraking)
    {
        count += Time.deltaTime;

    } else count = 0f;

    if (count > 3f) Destroy(gameObject, 1f);
}

0 referencias
private void OnDrawGizmos()
{
    if (_closestTrackPoint != null)
    {
        Gizmos.color = Color.magenta;
        Gizmos.DrawLine(transform.position, _closestTrackPoint);
    }
}

//just moves forward
1 referencia
private void Drive()
{
    if(!isBraking )
    transform.Translate(speed * Vector3.forward * Time.deltaTime);
}

private bool onIntersection = false;
0 referencias
private void OnCollisionStay(Collision collision)
{
    if (collision.gameObject.tag.Equals("Intersection"))
        {
        onIntersection = true;
        }
}

0 referencias
private void OnCollisionExit(Collision collision)
{
    if (collision.gameObject.tag.Equals("Intersection"))
    {
        onIntersection = false;
    }
}
    //DEPRECATED
    0 referencias
    private void Rotation(int currentlane, bool side, float distanceToCenter)...

    //returns -1 when to the left, 1 to the right, and 0 for forward/backward
    0 referencias
    public float AngleDir(Vector3 fwd, Vector3 targetDir, Vector3 up)...

    float count = 0;
    0 referencias
    private void MoveCar(float horizontal, float vertical, float dt)...
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Elementos de la simulación

Clase NodesVisualizer – Crea elementos del simulador tales como intersecciones, entradas, salidas e indicadores de flujo.

```
public class NodesVisualizer : MonoBehaviour
{
    15 referencias
    enum nodeTypes {_normal,_in,_out,_selected,_mixto,_intersection}

    private PathFinder script;
    private RoadMaker go;
    private MapReader map;

    public GameObject []nodeIndicators;
    private List<Node> intersections = new List<Node>();
    public List<Node> flujosEntrantes = new List<Node>();
    public List<Node> flujosSaliente = new List<Node>();
    private List<Node> flujosMixtos = new List<Node>();


    // Start is called before the first frame update
    0 referencias
    void Start()
    {
        go = GameObject.Find("Map").GetComponent<RoadMaker>();
        map = GameObject.Find("Map").GetComponent<MapReader>();
        script = PathFinder.instance;
        StartCoroutine("Nodes");
    }
    0 referencias
    IEnumerator Nodes()
    {
        yield return new WaitUntil(() => go.roadsReady == true);
        var nodesCount = script.graphData.nodes.Count();
        if (nodesCount != 0) {
            for(int i = 1; i < nodesCount; i++)
            {
                var typeOfNode = nodeType(script.graphData.nodes[i - 1].autoGeneratedID);
                Quaternion direction = getNodeDirection(script.graphData.nodes[i - 1].autoGeneratedID);

                switch (typeOfNode)
                {
                    case nodeTypes._in:
                        var flujo = script.graphData.nodes[i - 1];
                        flujosEntrantes.Add(flujo);
                        GameObject obj = Instantiate(nodeIndicators[0], script.graphData.nodes[i - 1].Position + Vector3.up, direction, gameObject.transform) ;

                        CarSpawner node = obj.GetComponent<CarSpawner>();
                        node.id = script.graphData.nodes[i - 1].autoGeneratedID;

                        int lanes = getLanes(script.graphData.nodes[i - 1].autoGeneratedID);
                        node.setLanes(lanes);
                        break;
                    case nodeTypes._intersection:
                        if(script.graphData.nodes[i - 1].isTrafficLights)
                        {
                            Instantiate(nodeIndicators[1], script.graphData.nodes[i - 1].Position + Vector3.up*0.01f, direction, gameObject.transform);
                        }else
                            Instantiate(nodeIndicators[3], script.graphData.nodes[i - 1].Position + Vector3.up, direction, gameObject.transform);


                        break;
                    case nodeTypes._mixto:
                        Instantiate(nodeIndicators[2], script.graphData.nodes[i - 1].Position + Vector3.up, direction, gameObject.transform);
                        break;
                    case nodeTypes._normal:
                        Instantiate(nodeIndicators[3], script.graphData.nodes[i - 1].Position + Vector3.up*0.1f, direction, gameObject.transform);

                        break;
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
                case nodeTypes._out:
                    var _flujoSaliente = script.graphData.nodes[i - 1];
                    flujosSaliente.Add(_flujoSaliente);

                    Instantiate(nodeIndicators[4], script.graphData.nodes[i - 1].Position + Vector3.up, direction, gameObject.transform);

                    break;
                case nodeTypes._selected:
                    Instantiate(nodeIndicators[5], script.graphData.nodes[i - 1].Position + Vector3.up, direction, gameObject.transform);

                    break;
                default:
                    Instantiate(nodeIndicators[3], script.graphData.nodes[i - 1].Position + Vector3.up, direction, gameObject.transform);

                    break;
            }
        }
    }

}

1 referencia
int getLanes (int node)
{
    foreach (var n in script.graphData.paths)
    {
        if (n.IDOfA == node || n.IDOfB == node)
        {
            return n.lanes;
        }
    }
    return 0;
}

1 referencia
Quaternion getNodeDirection(int node)
{
    Vector3 direction = new Vector3();
    Quaternion rotation = new Quaternion();
    foreach (var n in script.graphData.paths)
    {
        var a = script.graphData.nodes[n.IDOfA-1];
        var b = script.graphData.nodes[n.IDOfB-1];
        if (n.IDOfA == node)
        {
            direction = b.Position - a.Position;
            rotation = Quaternion.LookRotation(direction);
            return rotation;

        }
        else if (n.IDOfB == node)
        {
            direction = a.Position - b.Position;
            rotation = Quaternion.LookRotation(direction);
            return rotation;
        }

    }
    return Quaternion.identity;
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Clase CarSpawner – Determina el comportamiento de las entradas y *spawns* vehiculares

```
    nodeTypes nodeType(int node)
    {
 2 referencias
public class CarSpawner : MonoBehaviour
{

    CarPooler carPooler;
    PathFinder script;
    RoadMaker go;

    public GameObject[] prefabs;

    private NodesVisualizer nd;
    public int id;
    public int lanes = 0;
    private List<Vector3> spawnPoints;                                          :Saliente)
    public GameObject l;
    public GameObject r;
    [Range(0, 100)]
    public float spawnCycle;
    private List<Vector3> route;
    private bool isBlocked = false;

    public GameObject dm;

    1 referencia
    public void setLanes ( int p)
    {
        lanes = p;
    }

    0 referencias
    public void setID( int _id)
    {
        _id = id;
    }

    // Start is called before the first frame update
    0 referencias
    void Start()
    {
        script = PathFinder.instance;
        carPooler = CarPooler.Instance;
        spawnPoints = new List<Vector3>();
        nd = GameObject.Find("NodeHandler").GetComponent<NodesVisualizer>();
        go = GameObject.Find("Map").GetComponent<RoadMaker>();
        StartCoroutine(CreateSpawns());

    }

            if(w.idOTA == id || w.idOTB == id)
            {
                count++;
            }
        }
        return count;
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
1 referencia
IEnumerator CreateSpawns()
{
    yield return new WaitUntil(() => lanes != 0 && go.roadsReady);
    if (lanes == 1){
        object[] param = { transform.position, 2 };
        StartCoroutine(SpawnV2(param));
    }else if(lanes == 2){
        Vector3 offset = new Vector3(2f, 0f, 0f);
        l.transform.localPosition = l.transform.localPosition + offset;
        r.transform.localPosition = r.transform.localPosition - offset;
        object[] param = { l.transform.position, 1 };

        StartCoroutine(SpawnV2(param));
        object[] param2 = { r.transform.position, 3 };
        StartCoroutine(SpawnV2(param2));
    }
    else if (lanes == 3 || lanes == 4)
    {
        object[] param = { l.transform.position, 1 };
        StartCoroutine(SpawnV2(param));
        object[] param2 = { transform.position, 2 };
         StartCoroutine(SpawnV2(param2));
        object[] param3 = { r.transform.position, 2 };
        StartCoroutine(SpawnV2(param3));
    }
}
6 referencias
IEnumerator SpawnV2(object []param) //[position to spawn, lane to spawn]
{
    yield return new WaitUntil(() => go.roadsReady);
    var init_wait = Random.Range(0f, 6f);

    yield return new WaitForSeconds(init_wait);
    while (true)
    {

        bool shouldSpawn = false;
        switch (param[1])
        {
            case 1: //left
                if(l.GetComponent<NotifyCollision>()!= null)
                    shouldSpawn = !l.GetComponent<NotifyCollision>().getIsBlocked();
                break;
            case 2: //center
                    shouldSpawn = !isBlocked;
                break;
            case 3: //right
                if (r.GetComponent<NotifyCollision>() != null)
                    shouldSpawn = !r.GetComponent<NotifyCollision>().getIsBlocked();
                break;
            default:
                shouldSpawn = false;
                break;
        }
        var spawnTime = Random.Range(spawnCycle - 1f, spawnCycle + 2f);
        var cooldown = 0.5f;
        if (shouldSpawn){
           Instantiate(generateVehicle(),(Vector3)param[0],transform.rotation,transform);
            FindObjectOfType<DataManager>().vehiclesInScene++;
            FindObjectOfType<DataManager>().TotalVehiclesInstantiated++;
        }
        yield return new WaitForSeconds(spawnTime + cooldown);
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

```
    1 referencia
    GameObject generateVehicle()
    {
        var rand = Random.value;
        if(rand > 0.8f)
        {
            return prefabs[4]; //camion de carga
        }else if (rand >0.6f && rand <= 0.8f)
        {
            return prefabs[3];
        }else if (rand >0.5f && rand <= 0.6f)
        {
            return prefabs[2];
        }
        else if( rand >0.25f && rand <= 0.5f)
        {
            return prefabs[1];
        }
        else
        {
            return prefabs[0];
        }
    }
    //DEPRECATED - en desuso
    1 referencia
    int RandomizeOutNode()[...]
    0 referencias
    IEnumerator Spawn()[...]
    int originpoint = 0;
    0 referencias
    IEnumerator GenerateCars()[...]
    2 referencias
    private void SetPath(List<Vector3> nodes)[...]
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Clase IntersectionMaker – Regula los ciclos de los semáforos

```
0 referencias
public class IntersectionMaker : MonoBehaviour
{
    // Start is called before the first frame update
    public Material red;
    public Material yellow;
    public Material green;
    public GameObject[] faces; //1,2,3,4
    public float cycle = 15f;


    0 referencias
    void Start()
    {
        faces[0].GetComponent<MeshRenderer>().material = green;
        faces[1].GetComponent<MeshRenderer>().material = green;
        faces[2].GetComponent<MeshRenderer>().material = red;
        faces[3].GetComponent<MeshRenderer>().material = red;
        faces[0].GetComponent<lightsState>().state = 1;
        faces[1].GetComponent<lightsState>().state = 1;
        faces[2].GetComponent<lightsState>().state = -1;
        faces[3].GetComponent<lightsState>().state = -1;

        StartCoroutine(Semaforo_A());
        StartCoroutine(Semaforo_B());
    }

    1 referencia
    IEnumerator Semaforo_A()
    {
        while (true)
        {
            //green
            faces[0].GetComponent<MeshRenderer>().material = green;
            faces[1].GetComponent<MeshRenderer>().material = green;
            faces[0].GetComponent<lightsState>().state = 1;
            faces[1].GetComponent<lightsState>().state = 1;
            yield return new WaitForSeconds(cycle * 0.7f);
            //yellow
            faces[0].GetComponent<MeshRenderer>().material = yellow;
            faces[1].GetComponent<MeshRenderer>().material = yellow;
            faces[0].GetComponent<lightsState>().state = 0;
            faces[1].GetComponent<lightsState>().state = 0;

            yield return new WaitForSeconds(cycle * 0.3f);
            //red
            faces[0].GetComponent<MeshRenderer>().material = red;
            faces[1].GetComponent<MeshRenderer>().material = red;
            faces[0].GetComponent<lightsState>().state = -1;
            faces[1].GetComponent<lightsState>().state = -1;
            yield return new WaitForSeconds(cycle);
        }

    }

    1 referencia
    IEnumerator Semaforo_B()
    {
        while (true)
        {
            //red
            faces[2].GetComponent<MeshRenderer>().material = red;
            faces[3].GetComponent<MeshRenderer>().material = red;
            faces[2].GetComponent<lightsState>().state = -1;
            faces[3].GetComponent<lightsState>().state = -1;
            yield return new WaitForSeconds(cycle);
            //green
            faces[2].GetComponent<MeshRenderer>().material = green;
            faces[3].GetComponent<MeshRenderer>().material = green;
            faces[2].GetComponent<lightsState>().state = 1;
            faces[3].GetComponent<lightsState>().state = 1;
            yield return new WaitForSeconds(cycle * 0.7f);
            //yellow
            faces[2].GetComponent<MeshRenderer>().material = yellow;
            faces[3].GetComponent<MeshRenderer>().material = yellow;
            faces[2].GetComponent<lightsState>().state = 0;
            faces[3].GetComponent<lightsState>().state = 0;
            yield return new WaitForSeconds(cycle *0.3f);
        }

    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Clase TrafficFlow – Determina el comportamiento de los registradores de flujo de tráfico

```
public class TrafficFlow : MonoBehaviour
{

    private float totalVehicles = 0f;
    public float cycle = 60f;
    public float vehiclesCount = 0f;
    public float trafficFlow = 0f;

    public Material[] mats;

    // Start is called before the first frame update
    0 referencias
    void Start()
    {
        StartCoroutine(traffix());
    }

    1 referencia
    IEnumerator traffix()
    {
        while (true)
        {
            totalVehicles += vehiclesCount;
            vehiclesCount = 0;
            if (trafficFlow < 250f)
            {
                gameObject.GetComponent<MeshRenderer>().material = mats[0];
            } else if (trafficFlow < 1500f)
            {
                gameObject.GetComponent<MeshRenderer>().material = mats[1];
            }else if (trafficFlow< 1700f)
            {
                gameObject.GetComponent<MeshRenderer>().material = mats[2];
            }else if (trafficFlow < 2200f)
            {
                gameObject.GetComponent<MeshRenderer>().material = mats[3];
            }else gameObject.GetComponent<MeshRenderer>().material = mats[4];
            yield return new WaitForSeconds(cycle);
            trafficFlow = 3600f*vehiclesCount / cycle;

            if (FindObjectOfType<DataManager>() != null)
            {
                if(trafficFlow!=0)
                FindObjectOfType<DataManager>().AddToAverageFlow(trafficFlow);

            }
            else Debug.Log("ERROR");
        }
    }

    0 referencias
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag.Equals("Vehicle"))
        {
            vehiclesCount++;

        }
    }
}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Clase TrafficDensity – Determina el cálculo de la densidad de tráfico

```
1 referencia
public class TrafficDensity : MonoBehaviour
{
    private Mesh mesh;
    private Vector3 size;
    private int vehicleCount =0;
    public float density =0f;

    // Start is called before the first frame update
    0 referencias
    void Start()
    {

        if (gameObject.GetComponent<MeshFilter>() != null)
        {
            mesh = gameObject.GetComponent<MeshFilter>().sharedMesh;
            var actualSize = Matrix4x4.Scale(transform.localScale) * mesh.bounds.size;
            size = actualSize;
        }
        //StartCoroutine(Counter());
    }

    0 referencias
    private void Update()
    {
        if (size != null)
        {
            density = (vehicleCount * 1000f) / size.z;
            if (FindObjectOfType<DataManager>() != null)
            {
                if(density!=0)
                FindObjectOfType<DataManager>().AddToAverageDensity(density);

            }
            else Debug.Log("ERROR");
        }
    }

    0 referencias
    IEnumerator Counter()...


    0 referencias
    private void OnCollisionEnter(Collision other)
    {
        if (other.gameObject.tag.Equals("Vehicle"))
        {
            vehicleCount++;
        }
    }

    0 referencias
    private void OnCollisionExit(Collision other)
    {
        if (other.gameObject.tag.Equals("Vehicle"))
        {
            if(vehicleCount!=0)
            vehicleCount--;
        }
    }

}
```

Universidad del Bío-Bío. Sistema de Bibliotecas-Chile

Clase Despawn – Determina comportamiento cuando un vehículo sale de escena

```
public class Despawn : MonoBehaviour
{
    public GameObject dm;
    private void OnTriggerEnter(Collider collision)
    {
        if (collision.gameObject.tag.Equals("Vehicle"))
        {
            var vehicle = collision.gameObject;
            if (vehicle.GetComponent<SimpleCarAi>() != null)
            {
                FindObjectOfType<DataManager>().AddToAverage(vehicle.GetComponent<SimpleCarAi>().timeAlive);
            }
            Destroy(collision.gameObject);
            FindObjectOfType<DataManager>().vehiclesInScene--;
        }
    }
}
```