UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Curso Académico 2010/2011

Proyecto de Fin de Carrera

WINDistrict 1.0

Autor: Pedro García De La Cruz

Tutor: Matteo Vasirani

Resumen

El proyecto surge a partir de la evolución en el avance de las tecnologías productoras de electricidad de forma renovable, por la instauración paulatina de estas energías renovables en nuestra sociedad y para combinar ambas en elementos que son utilizados a diario por posibles futuros usuarios.

En concreto, el proyecto se centra en las energías eólicas y su aplicación sobre los elementos de transporte que habitan en nuestras ciudades.

El objetivo del proyecto es calcular el consumo de energía que pueden consumir los vehículos en el transcurso del día a día y la cantidad de energía que se puede obtener en condiciones normales por un determinado campo eólico.

Para conseguir los objetivos planteados en el proyecto se ha procedido a la creación a escala de una posible ciudad, basando la creación de ésta en datos de ciudades reales.

Para la información vehicular se ha generado una cantidad de vehículos, los cuales tienen sus caminos propios, sus posiciones propias dentro de la ciudad como ocurre en la realidad y se ha procedido a la implementación de una Inteligencia Artificial que actúa como "cerebro" de los vehículos, comprobando y analizando las rutas que el mismo ha realizado, está realizando o va a realizar.

Para la información eólica se ha generado un campo eólico, con molinos suficientes para abastecer la necesidad de energía eléctrica que tienen los vehículos creados y con cierto margen para que éstos no queden desabastecidos.

Como complementos, y para mostrar la intención de realismo buscado en este proyecto, se han creado elementos cotidianos como:

- Terrenos, carreteras, señales de tráfico, aceras.
- Edificios de viviendas, tiendas, oficinas.
- Simulación del día, mediante un cielo, sol, luna, lluvias.

En cuanto a la funcionalidad de la plataforma, existe un acceso directo para Windows con el que poder realizar la simulación sin tener que abrir ningún programa externo, al igual que existe un acceso para abrir la ciudad a simular mediante el programa Unity3D. También existe la posibilidad de realizar archivos de acceso directo para Web, Apple, PS3, Wii y PSP modificando únicamente las direcciones de los ficheros utilizados por la simulación.

Índice

1. Introducción

Debido al notable crecimiento de técnicas que proporcionan energías renovables de forma limpia y ecológica, se necesitan datos fiables en los que basarse, para que los inversores que apuestan por estas tecnologías e incluso las mismas sociedades que las demandan, puedan contar con datos lo más fiables posible y adaptados a sus ciudades, para que éstos puedan contrastar si los resultados a obtener son los deseados.

Tenemos multitud de energías renovables, como pueden ser: energía hidráulica, solar, eólica, biomasa, etc. Pero todas ellas tiene algo en común, y es el principal problema que las abarca: su inestabilidad. Debido a esta inestabilidad, se necesitan datos fiables no sólo sobre el consumo eléctrico que puede realizar una sociedad, sino también, se necesitan datos fiables con los que poder contrastar si es posible el auto-abastecimiento de dichas sociedades mediante la utilización única de las energías renovables.

En este proyecto nos centramos en la energía eólica, la cuál es obtenida mediante la fuerza del viento, mediante la utilización de la energía cinética generada por las corrientes de aire. Se obtiene a través de unas turbinas eólicas que son las que convierten la energía cinética del viento en electricidad por medio de aspas o hélices que hacen girar un eje central conectado, a través de una serie de engranajes a un generador eléctrico.

El otro fundamento en el que nos centramos en este proyecto, es en los vehículos eléctricos, los cuáles son impulsados por uno o varios motores eléctricos. La energía que necesitan es, por tanto, energía eléctrica, sin influir la procedencia de ésta. El consumo por cada 100Km, es aproximadamente de 13.78Kw, disponiendo de una autonomía de 28.25Kw, por lo que uno de éstos vehículos eléctricos dispondrá de una autonomía aproximada de 200Km.

Debido a la información de la que se dispone, viene siendo muy práctico, la realización de simulaciones con datos lo más reales posibles para poder contrastar la viabilidad tanto de la energía eólica como energía renovable sustituta de la energía proveniente de fósiles, como de la utilización de vehículos eléctricos en el uso diario de cualquier sociedad actual.

2. Objetivos

El propósito principal de este proyecto es la simulación del tráfico de una ciudad y el campo eólico que ésta podría tener, para con esto, obtener datos sobre consumos y necesidades eléctricas que una ciudad puede tener en su día a día.

Se decidió la creación de ésta simulación al ser un reto interesante debido a sus múltiples objetivos:

- Diseñar en 3D una ciudad.
- Diseñar en 3D un campo eólico.
- Diseñar en 3D vehículos que compondrán el tráfico vehicular.
- Crear una I.A. encargada del tráfico vehicular.
- Crear una I.A. encargada de generar datos eólicos.
- Contabilizar datos obtenidos en tiempos de simulación.
- Mostrar gráficamente los datos obtenidos en tiempo real.
- Posibilidad de ejecutar la simulación en varias plataformas.
- Posibilidad, mediante un caso base de:
    - Creación de rutas a seguir por uno o varios vehículos.
    - Introducción de tráfico, siguiendo rutas base (o nuevas) en la simulación.
    - Generación de datos eólicos, de forma aleatoria o mediante datos reales.
    - Comprobación de rutas introducidas.

A pesar de ser un proyecto con muchos puntos a tratar, el objetivo fundamental radica en la combinación de todos los requisitos para conseguir obtener datos resultado de la simulación, que traten de ser lo más reales posible.

Para tratar de obtener los resultados buscados se han abordado los objetivos principalmente como si fueran dos únicos caminos:

- Parte diseño: Se ha utilizado el programa de diseño en 3D, 3D Studio Max.
- Parte escenario: Para la creación del escenario, las inteligencias artificiales, la obtención de datos e incluso la creación de un fichero simulación, se ha utilizado el programa de creación de videojuegos Unity3D.

3. Metodología

Considerando una metodología de desarrollo como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software; se ha decidido separar este proyecto en 3 fases, para explicar detalladamente la metodología en cada una de ellas. Porque a pesar de ser utilizada una metodología similar, se aplican técnicas diferentes al hacer un diseño gráfico en 3D que al implementar una inteligencia artificial.

3.1. Fase diseño 3D

Tareas

El objetivo de esta fase radica en la creación de objetos diseñados en 3D, los cuales formarán parte de la ciudad a desarrollar. Para ello, se necesitan las siguientes clases de objetos:

- Entramado de carreteras y aceras.
- Elementos de la ciudad como pueden ser: casas, coches, edificios deportivos, edificios de oficina, almacenes industriales, edificios de viviendas, señales de tráfico, tiendas, farolas.
- Molinos.

Salidas

Las salidas producidas en esta fase de diseño serán ficheros del tipo 3D (formato .3DS o formato .Max), donde cada uno de éstos ficheros será un objeto desarrollado.

Restricciones

La única restricción aplicada en esta fase consiste en el número de polígonos que componen el objeto. Cada objeto está compuesto por una serie de polígonos, pero el programa encargado de aceptar estos objetos para formar la ciudad (Unity3D) tiene una cantidad máxima de polígonos aceptados y por esto, los objetos deben tener una cantidad máxima para poder ser introducidos en la escena.

En este apartado, el objeto donde se han creado las carreteras, ha tenido que ser dividido en 4 sub-objetos, para que éstos no excedan el número de polígonos permitido.

Herramientas

La herramienta utilizada en esta fase de diseño ha sido el programa: 3D Studio Max.

Para el diseño en esta etapa, también se ha dispuesto de texturas con las que dar realidad a los objetos (por ejemplo, texturas para la carretera o las aceras), y de algunos objetos obtenidos a través de internet (como un estadio de fútbol, que forma parte de la ciudad).

3.2. Fase creación escenario

Tareas

El objetivo en esta fase consiste en la creación de una ciudad, unificando los objetos creados en la fase anterior y de la creación de las siguientes funcionalidades con las que poder hacer que dicha ciudad tenga movilidad:

- Creación de un terreno en el que poder situar la ciudad, dándole aspecto real mediante la creación de montañas, bosques (en este apartado, se ha decidido realizar el diseño del terreno con Unity3D, ya que dispone de una herramienta para ello la cuál soportará los objetos con sus leyes físicas como puede ser la gravedad).
- Introducción coherente de los elementos creados en la fase anterior, para dar un aspecto real a la hora de crear una ciudad y un campo eólico.
- Creación de una I.A. encargada de simular un viento que será el encargado de mover los molinos y dar información eólica durante la simulación.
- Creación de una I.A. la cuál será encargada de aceptar datos con los que poder crear rutas a seguir por los vehículos, introducir qué cantidad de vehículos se desea que realicen dichas rutas, durante qué horario se desea que realicen las rutas.
- Simular una atmósfera, generando así una sensación de realidad en el escenario, el cuál mostrará cómo transcurre el día mediante su rotación solar, generación aleatoria de lluvia y paso de las horas.
- Creación de código encargado de comprobar si las rutas introducidas son correctas.

- Creación de una I.A. encargada de generar el tráfico, según un horario determinado.

Salidas

La salida obtenida y deseada en esta fase, consiste en un escenario, el cuál será utilizado para almacenar en ficheros los datos que queremos que sean simulados en la siguiente fase.

Restricciones

En esta fase aparecen las siguientes restricciones:

- Objetos de la ciudad formados por un número máximo de polígonos (65.000).
- Limitación del número de vehículos a formar parte en la simulación (aproximadamente 30.000 vehículos), debido a la eficiencia del programa.

Herramientas

En esta fase se ha utilizado la herramienta Unity3D, y múltiples de sus opciones como son:

- "The Chronosphere Prefab", que simula una atmósfera y el transcurso de los días.
- Physics, dando gravedad a los objetos. Siendo útil esto para los vehículos, dotándolos de credibilidad a la hora de simular su circulación.
- Collider, consistente en la creación de una malla de colisión. Esto es, el objeto que contiene esta funcionalidad, dispone de la opción de poder chocar contra otros objetos haciendo así que pueda ver a otros objetos con los que pueda chocar o que otros objetos le puedan ver a él.
- Light, consistente en la aparición de una luz. Esto ha sido útil a la hora de crear ambiente lumínico mediante farolas, el sol o luces para el estadio de fútbol.

3.3. Fase diseño simulación

Tareas

En esta fase, una vez creados los objetos que componen el escenario y la ciudad con sus funcionalidades, la tarea es ir comprobando que todo funciona correctamente e ir modificando los datos que sean necesarios.

Se ha decidido realizar una fase especial para la comprobación de la funcionalidad de la ciudad, esto ha sido debido a la creación de las rutas.

Para la creación de las rutas, ha sido necesario crear esferas que van por las carreteras. Estas esferas serán puntos que compondrán un camino, pero una vez obtenido el mismo, son innecesarias en la simulación. Por ello se ha decidido crear un caso base con dichos puntos, para la creación o modificación de caminos, y este caso el cuál únicamente se encarga de la simulación.

La utilización de cámaras también es realizada en esta fase, para sobrecargar menos el caso base y también porque es innecesario un cambio de cámara a la hora de crear caminos.

Otra importante utilidad de esa fase consiste en la selección de los datos del campo eólico, esto es, el script encargado de generar una I.A. con datos eólicos (script adjunto al objeto “Campo Eólico”) permite dos formar de simular el viento: cargando datos desde ficheros que contendrán la velocidad, dirección y potencia durante cada tiempo; o cargando datos aleatoriamente pero con cierta coherencia (esto es, genera aleatoriamente un viento, pero con rangos para que esa aleatoriedad tienda hacia datos más reales).

Salidas

La salida será un fichero ejecutable, que efectuará la simulación de manera que no sea necesario acceder a Unity3D.

Importante indicar, que la salida elegida es un fichero ejecutable bajo Windows, pero existe la posibilidad de generar ficheros ejecutables bajo las plataformas de Web, Mac, iOS, Android, X-Box 360, PS3, Wii, pero modificando las direcciones de acceso de los ficheros utilizados en la simulación.

La salida para la simulación consistirá en la creación de un fichero (llamado “Datos Simulacion.txt”) con los datos obtenidos durante la misma que contendrá los siguientes campos:

- Número de coche
- Distancia recorrida por el mismo
- Consumo eléctrico efectuado durante el recorrido, teniendo en cuenta si está parado, ya que también ahí está consumiendo electricidad.
- Batería disponible

Restricciones

Esta fase se basa prácticamente en comprobar el funcionamiento de la anterior y en la generación de un fichero ejecutable, por lo que las restricciones que podrían aparecer han sido comprobadas en la fase anterior.

Herramientas

Las mismas que en la fase anterior.

4. Descripción informática

En esta sección, nos centramos en los aspectos técnicos de desarrollo del proyecto, pero unificando las fases en una. Esta unificación es debido, a que durante la fase de diseño de objetos en 3D no existe funcionalidad alguna, únicamente se busca una finalidad (en este caso, un objeto) el cuál sí pasará a formar parte de un grupo que busca una funcionalidad.

4.1. Especificación de requisitos

A continuación se especificarán las restricciones que tiene el sistema, tanto para la simulación creada como para la creación de simulaciones. Esto es, para la simulación el usuario únicamente tendrá que ejecutarla y observar los resultados que se van obteniendo (ya que se trata de una simulación, no de un sistema que interactúe con el usuario) teniendo las únicas opciones de cambiar de cámara o terminar la simulación, pero también existe la funcionalidad de poder generar e introducir datos y crear su propia simulación a través del caso base para Unity3D y se determinarán los requisitos de ésta también.

Para la siguiente especificación, en el caso de la creación de simulaciones, únicamente se analizarán restricciones del caso base, esto es, situaciones para la ciudad existente. Si el usuario desea crear una nueva ciudad, únicamente debería introducir los elementos gráficos que componen a ésta, los puntos deseados para formar los caminos y la funcionalidad será la misma que para el caso base. Luego sus requisitos serían los mismos, solo que con objetos diferentes o en posiciones diferentes.

R.1: Ejecución de simulación

- **Tipo**: Funcional
- **Descripción**: Ejecutar la simulación de la ciudad, obteniendo los resultados tanto en pantalla como en fichero.
- **Entrada**: Datos provenientes de ficheros para la realización de caminos para los vehículos. Datos provenientes de ficheros (en caso de utilizarlos, ya que se puede generar aleatoriamente) para la generación del viento que mueve los molinos.
- **Salida**: Ejecución gráfica de la simulación, con el funcionamiento normal en un día en la ciudad creada.

R.2: Creación de simulación

- **Tipo**: Funcional
- **Descripción**: Crear un fichero para ejecutar una simulación. Para ello el usuario debe abrir el fichero "WDistrict.unity" que contiene la simulación y construir la simulación correspondiente.
- **Entrada**: Escenario sobre el que se efectuará la simulación de datos, compuesto por la ciudad y sus elementos.
- **Salida**: Fichero que ejecutará la simulación de la ciudad sin necesidad de acceder al programa Unity3D.

R.2.1: Campo eólico

- **Tipo**: Funcional
- **Descripción**: El usuario deberá elegir la opción que desea tener para la generación del viento: de forma aleatoria o cargando datos desde fichero.
- **Entrada**: En caso de no seleccionar datos aleatorios, ficheros con datos del viento.
- **Salida**: Funcionalidad del viento sobre los molinos.

R.2.2: Selección horaria

- **Tipo**: Funcional
- **Descripción**: El usuario deberá seleccionar la hora en la que desea que empiece la simulación, tanto en el contador de tráfico (objeto "Trafico", seleccionar hora y minuto sobre su script) como en la Chronosphere (objeto ChronosphereControls, seleccionar Hour Start de su script). Importante indicar, que la misma terminará cuando lo indica el usuario o a las 23:59 del reloj de la simulación.
  Se ha decidido que el usuario no pueda determinar la duración de una simulación (excepto modificando el script "ContadorGeneral.cs") para obtener resultados fiables, dejando que los vehículos desarrollen sus trayectos a velocidades que permitan una contabilización de datos reales. Por ello, si el usuario desea que la simulación tenga un tiempo determinado, debería iniciar la simulación a una hora, sabiendo que la simulación terminará cuando el reloj de la misma marce las 23:59 y todos los vehículos hayan realizado su trayecto.
- **Entrada**: Hora de inicio que indicará la simulación.
- **Salida**: Reloj de simulación iniciado a los datos de entrada.

R.2.3: Introducción elementos de ciudad

- **Tipo**: No funcional
- **Descripción**: El usuario tiene la posibilidad de introducir elementos de ciudad en la simulación, para ello tendría que importarlos a la carpeta que contiene el proyecto e insertarlos en la posición que desee de la ciudad.
- **Entrada**: Objetos para la ciudad.
- **Salida**: Ciudad que contendrá los objetos insertados.

R.2.4: Introducción vehículos

- **Tipo**: Funcional
- **Descripción**: A pesar de que los vehículos son elementos de la ciudad, este requisito se indica a parte, ya que necesita un trato especial. El usuario debe introducir el vehículo en la escena, y agregarle el script "RealizaCamino.cs", ya que éste se encargará de mover dicho objeto durante la simulación.
- **Entrada**: Objeto vehículo.
- **Salida**: Ciudad con el nuevo objeto vehículo.

R.3: Terminar simulación

- **Tipo**: Funcional
- **Descripción**: Durante la simulación, ésta podrá correr a pantalla completa o en una ventana. En el primer caso, el usuario necesitará presionar la tecla "ESC" o esperar la terminación de un día completo en la simulación (duración aproximada de 40 minutos). Al terminar dicha simulación, ya sea opción del usuario o finalización de la misma, se almacenarán los datos generados en un fichero destino.
- **Entrada**: Tecla "ESC"
- **Salida**: Terminación del programa y almacenamiento de datos resultado en fichero destino.

R.4: Selección de cámaras

- **Tipo**: Funcional
- **Descripción**: Durante la ejecución de la simulación, el usuario podrá cambiar la perspectiva de imagen seleccionando alguna de las 12 cámaras disponibles. Esto mostrará 11 puntos diferentes de la ciudad (ya que a escala real trataría de unos 70Km cuadrados), y una tercera cámara que mostrará el campo eólico.

- **Entrada**: Pulsación de las teclas ‘0’, ’1’ ,’2’ ,’3’, ’4’, ’5’, ’6’, ’7’, ’8’, ’9’, ’m’, ’c’
- **Salida**: Muestra perspectivas de la ciudad.

R.4.1: Desactivando cámara

- **Tipo**: No funcional
- **Descripción**: Durante la creación de una simulación, el usuario debería tener únicamente una cámara activada (estado por defecto), para no perder eficiencia en la creación de la simulación.
- **Entrada**: Desactivación de todas las cámaras, excepto una.
- **Salida**: Ninguna.

R.5: Seleccionando gráficos

- **Tipo**: Funcional
- **Descripción**: Al iniciar la simulación, aparecerá un menú en el que el usuario podrá seleccionar la resolución gráfica y el tamaño de la ventana de la simulación.
- **Entrada**: Datos de resolución y tamaño de ventana.
- **Salida**: Simulación acorde con los datos de entrada.

R.6: Introducción datos simulación

- **Tipo**: Funcional
- **Descripción**: El usuario tiene la opción de introducir datos en la simulación, tales como las carreteras a recorrer, las rutas, cantidad de vehículos que realizarán las rutas, direcciones de los carriles de las carreteras, creación de rutas base e incluso la reparación de rutas ya creadas. Para ello deberá abrir el fichero “CasoBase.unity” con Unity3D.
- **Entrada**: Datos funcionales para la simulación.
- **Salida**: Datos funcionales para la simulación.

R.6.1: Almacenar puntos de carretera

- **Tipo**: No funcional
- **Descripción**: El usuario deberá crear tantas “Spheres” como necesite a lo largo de la carretera que desea sea transitable, de tal forma que formen el trayecto de la misma. Estas “Spheres” deberán estar contenidas en un objeto padre por cada carril, indicando el nombre “Way” si la primera esfera será la salida del carril, o “AWay” si la primera esfera es la última del carril. Estos objetos, deberán estar a su vez

contenidos en otro llamado “RoadN”, siendo n el número que se desea dar a la carretera. Una vez realizado esto, se deberá adjuntar el script “OrdenaDirecciones.cs” al cualquier objeto y ejecutar Editor, para así que el script se encargue de crear ficheros en los que simulará el trayecto de la carretera con su dirección indicada.
También existe un script “CompruebaDirecciones.cs”, que será el encargado de comprobar que los nombres de las carreteras y sus direcciones son correctos.

- **Entrada**: Puntos a recorrer de una carretera.
- **Salida**: Ficheros con las carreteras simulando su sentido.

R.6.2: Introducir rutas base

- **Tipo**: No funcional
- **Descripción**: Si el usuario desea introducir una ruta, que sirva como base para otras (por ejemplo si quiere crear varias rutas que empiezan en lugares diferentes, pero tienen un punto intermedio en común y el resto de su camino hasta terminar es igual), adjuntará el script “IntroduceRutaBase.cs” a un objeto activo, le agregará al mismo los puntos que desea formen parte del trayecto. Para no tener que adjuntar todos los puntos a recorrer, el usuario puede introducir el punto inicial y final, de cada carretera que desea que el vehículo/s recorra.
- **Entrada**: Puntos de salida de cada carretera, y puntos de fin de las mismas.
- **Salida**: Fichero con el recorrido introducido.

R.6.3: Introducir rutas

- **Tipo**: No funcional
- **Descripción**: El usuario puede introducir el número de vehículos que desea que carguen la ruta indicada, ajuntando ésta de la misma forma que en el punto anterior. Si existe un atajo de ida se cargará primero la ruta base seguida del nuevo trayecto a recorrer, si el atajo es de vuelta se cargará primero la ruta indicada seguida de la ruta base, si no existe atajo únicamente se cargará un fichero con los datos de la ruta a recorrer.
- **Entrada**: Puntos de ruta a recorrer y posible ruta base.
- **Salida**: Fichero con ruta a recorrer, un fichero por cada vehículo que va a realizar la ruta y ficheros (generados de

forma aleatoria) que indicarán en cada franja horaria qué vehículo será el que inicie su camino.

R.6.4: Reparación de rutas

- **Tipo**: No funcional.
- **Descripción**: Existe la posibilidad de que el usuario haya introducido un atajo de forma errónea, para ello dispone del fichero "ReparaRutas.cs", el cuál comprobará que los atajos introducidos seguidos de su ruta tengan una coherencia.
- **Entrada**: Ninguna.
- **Salida**: Reparará los ficheros donde vea que existe una incoherencia.

4.2. Diseño

Esta fase se encarga de dar soporte a los requisitos obtenidos en la fase anterior, obteniendo así una abstracción del proceso de implementación y una entrada apropiada para dicha fase.

El proyecto, como se ha mencionado anteriormente tiene las fases de diseño en 3D, la creación de una simulación y la propia simulación. El objetivo de dicho proyecto, es el resultado de la simulación, y es por esto que se explicará el diseño en base a la ejecución y actuación de la simulación y no basándose en los casos de diseño gráfico o creación de otras simulaciones.

A continuación se muestra el diagrama de carga de la simulación:

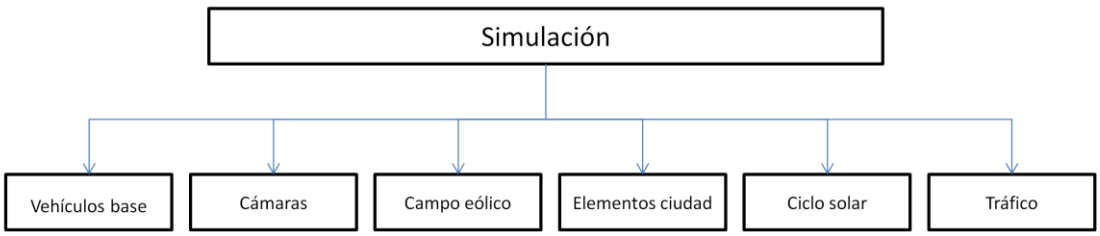

Figura 1. Diagrama carga de simulación

Como se puede observar, el diagrama anterior muestra únicamente la forma que tiene la simulación creada de cargar los datos y objetos, esto es debido a que la simulación genera los objetos y los sitúa en un escenario gráfico, pero durante la ejecución de la simulación cada objeto tendrá funcionalidades distintas y con interactuaciones con otros objetos distintas.

Ahora se procede al análisis determinado de cada uno de estos grupos de objetos.

4.2.1 Vehículos base

Los vehículos base son objetos hijos de “Coches” y tendrán un nombre del tipo “CocheB N”, donde N>0.

Estos objetos tendrán adjuntos un script encargado de realizar el camino que le tocaría (RealizaCamino.cs), pero en realidad estos vehículos no se moverán por la simulación. Sirven como objetos base, los cuales serán clonados por el objeto “Tráfico”, quedando dichos objetos en su situación inicial.

Disponen de las características “Physics” y “Collider”, para poder introducir al objeto unas cualidades reales dotándole de peso y propiedades físicas. Para ello debe acceder a los componentes de “Physics” y “Collider”, accediendo a éstos en el momento de su creación y durante toda la simulación, al tratarse de propiedades que debe almacenar el objeto durante la misma.

A continuación se muestra un diagrama donde se indica la relación del objeto con otros del escenario:

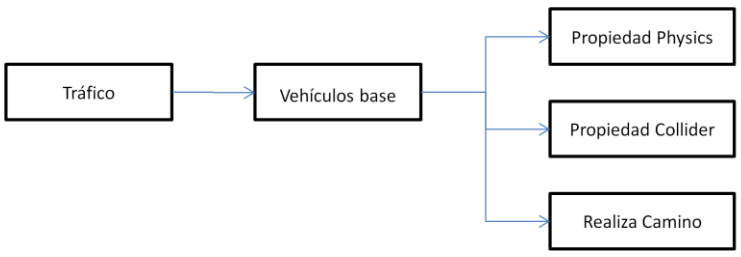

Figura 2. Diagrama vehículos base

4.2.2 Cámaras

Este objeto es un objeto padre de cada objeto “Camera”, con un total de 11 (la cámara 12 forma parte del objeto “Ciclo solar”) cámaras.

Se ha optado por el diseño de un objeto padre que contenta todas las cámaras por motivos de eficiencia y organización. El objeto padre accede al script encargado de mover las cámaras según los datos introducidos por el usuario (MoviendoCamara.cs), obteniendo así el resultado de cambios de cámara durante la simulación.

Diagrama de relación del objeto con otros del escenario:

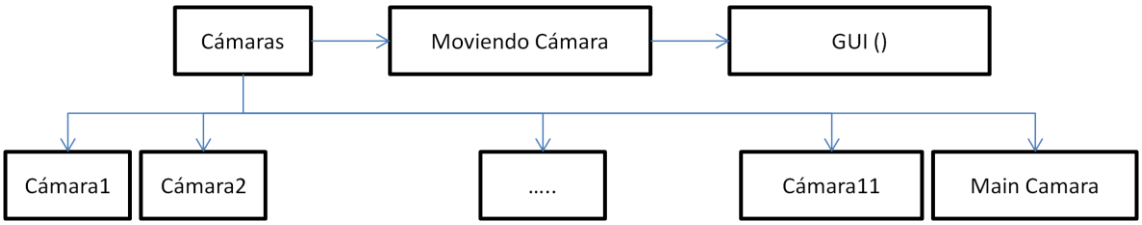

Figura 3. Diagrama cámaras

4.2.3 Campo eólico

Este objeto es padre de los objetos “Molino”, con un total de 15 hijos. Dispone de acceso al script “CampoEolico.cs”, mediante el cual puede acceder a E/S de datos de ficheros y al GUI para mostrar la información por pantalla, también dicho script accede a los hijos de este objeto, pero al tratarse de hijos del mismo en el diagrama se indicará únicamente el acceso del padre ya que éste ahorra el acceso a sus hijos al ser él quien llama al script.

A continuación se muestra su diagrama con respecto a la simulación:

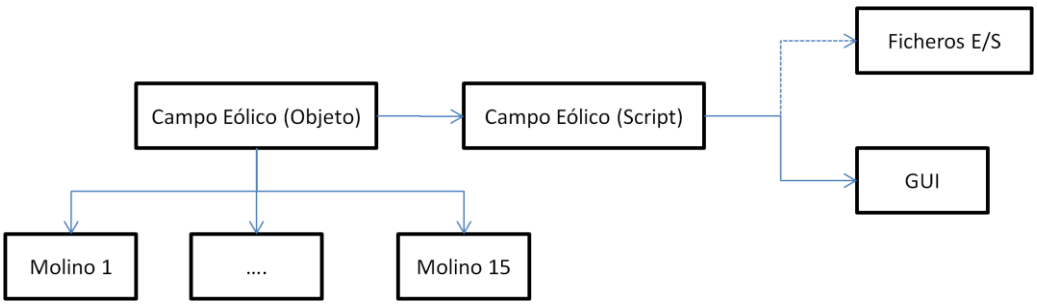

Figura 4. Diagrama Campo eólico

4.2.4 Elementos de ciudad

Estos son objetos estáticos, los cuales (excepto los objetos hijo de "Farolas Ciudad" y "Señales") no son accedidos ni acceden a otros con otros objetos de la misma.

En esta clase, también se incluyen los objetos Roads, Terrain y "The Chronosphere Prefab".

A pesar de ser objetos estáticos, tenemos que algunos de ellos si son accedidos por otros objetos o que tienen propiedades especiales, estos son:

Farolas Ciudad: Objetos accedidos a través del objeto "Contador General", que accede a éstos para apagar su luz si ya es de día.

Señales: Objetos que disponen de la propiedad "Collider", para que los vehículos actúen en consecuencia al cruzarse con alguno de éstos objetos.

Roads: Disponen de las propiedades "Physics" y "Collider", para dar a las carreteras una actuación realista.

Terrain: Dispone de la propiedad "Terrain Collider", este componente da al objeto una propiedad de terreno haciendo que éste se comporte al igual que lo haría un terreno real.

The Chronosphere Prefab: Este es un componente de Unity3D, el cuál simulará un cielo, tanto de día como de noche, e incluso en su cámara adjunta puede simular una lluvia que se producirá de forma aleatoria.

4.2.6 Tráfico

Este objeto es el encargado de gestionar el tráfico de la ciudad, para ello accede a los vehículos base y clona tantos vehículos como necesita para la simulación (obteniendo los datos de ficheros horarios). Una vez clonados los vehículos, los va activando cuando les toca hacerlo. También gestiona un contador horario, que se encarga de simular la hora en la simulación.

El diagrama del objeto con los elementos de la simulación es el siguiente:

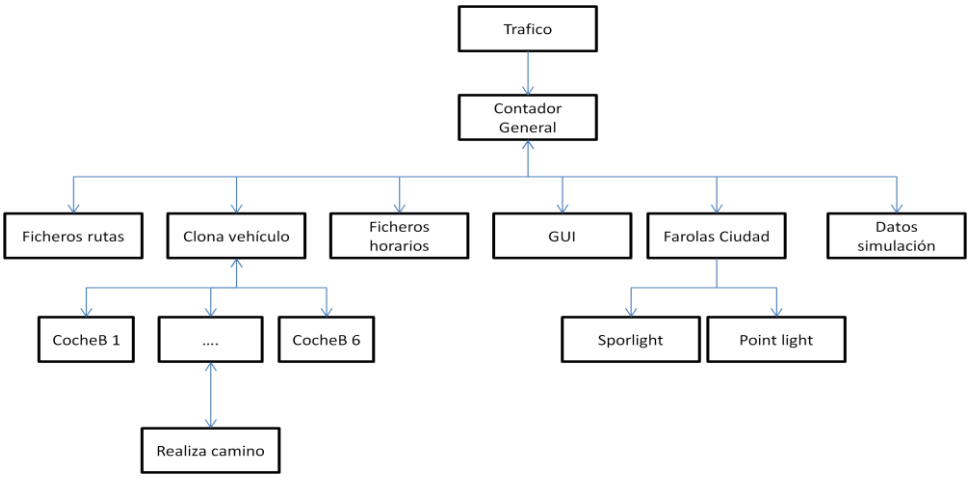

Diagrama 5. Diagrama Tráfico

4.3. Implementación

En el punto anterior se ha explicado el método de creación de los objetos y las relaciones que entre éstos existe. Debido a las características de Unity3D y del proyecto, la creación de los objetos que componen el escenario y la relación que dan la funcionalidad de la simulación se ven explicadas en el punto anterior.

Por ello en esta fase nos centraremos exclusivamente en los scripts realizados, que son los que dan funcionalidad a la simulación, y se explicará de forma resumida la funcionalidad de cada función.

4.3.1. "CampoEolico.cs"

Script utilizado durante la simulación.

- *GeneraVelocidadPotencia()*: Genera aleatoriamente una velocidad y la potencia generada por ésta.
- *GeneraDireccion()*: Genera aleatoriamente una dirección del tipo: Norte, Sur, Este, Oeste.
- *Start()*: Función que se ejecutará al iniciar la simulación. Capturará los molinos que componen el campo eólico, comprueba si debe generar los datos aleatoriamente o los recibe por fichero y actúa en consecuencia a ello.
- *Update ()*: Función que se ejecuta cada Frame llamando a ActuaCampoEolico.
- *MueveAspas()*: Rota las aspas del molino según la velocidad del viento.
- *GiraMolino()*: Rota el molino sobre sí mismo, según la dirección del viento.
- *ActuaCampoEolico()*: Mueve los molinos en factor a su dirección, y gira sus aspas a velocidad del viento.
- *GeneraDatos()*: Genera nuevos datos en su llamada para viento, potencia y dirección si se desea generar viento aleatorio.

- *CargaDatosFichero()*: Carga los datos con los que introducir la velocidad, dirección y potencia del viento desde los ficheros origen.

- *SiguienteDatoFichero()*: Si obtiene los datos de ficheros, esta función se encarga de pasar de un dato al siguiente de forma cíclica.

- *OnGUI()*: Muestra por pantalla los datos referentes al campo eólico.

4.3.2. "CompruebaDirecciones.cs"

Este script será utilizado únicamente para la creación de escenarios, no para la simulación.

- *ComprobarDirecciones()*: Comprueba si los objetos padre que contienen los puntos que forman la dirección de una carretera, tienen el nombre correcto.

- *Start()*: Función que llamará a ComprobarDirecciones.

4.3.3. "IntroduceRuta.cs"

Script utilizado únicamente para la creación de rutas, no para la simulación.

- *obtieneRecorrido()*: Dado los puntos de entrada, obtiene el recorrido que los va uniendo.

- *nomFichSalida()*: Función que buscará un nombre libre para la ruta. Sera del tipo "Ruta X", donde X es un entero.

- *datosCoche()*: Función que creará ficheros con los nombres de los coches, para llevar un contador de los mismos.

- *AlmacenaDatos()*: Almacena la ruta en un fichero con nombre Ruta X. También almacena en los ficheros "Hora X Minuto Y" el nombre de la ruta y coche, para que sean cargados en la Hora X : Minuto Y.

- *recogeAtajo()*: Recoge un camino dado por el nombre de entrada nombreAtajo. Este camino será un camino pre-definido, para ahorrar tiempo a la hora de crear caminos.

- *calculaHorasAleatorias()*:Genera aleatoriamente las horas en las que los vehículos realizarán el trayecto.

- *Start()*: Calcula aleatoriamente las horas en las que se van a realizar los trayectos. Comprueba si existe un atajo, en cuyo caso lo utiliza y en caso contrario únicamente almacena el trayecto indicado.

4.3.4. "IntroduceRutaBase.cs"

Script utilizado únicamente para la creación de rutas, no para la simulación.

- *obtieneRecorrido()*: Calcula los puntos del recorrido que se ha introducido al script.
- *almacenaPuntos()*: Almacena los puntos del recorrido en el fichero de salida.
- *Start()*: Llama a la función almacenaPuntos.

4.3.5. "OrdenaDirecciones.cs"

Script utilizado únicamente para la creación de rutas, no para la simulación.

- *AlmacenarDireccion()*: Guarda los puntos del carril en el fichero destino.
- *InvertirPosiciones()*: Invierte las posiciones de los arrays con los puntos del carril, para darle la dirección correcta.
- *Ordenar()*: Crea un fichero del tipo RoadX [AR] WayY.txt, donde ordenará los puntos de forma que simule la dirección del carril de cada carretera. El primer punto del fichero solo podrá acceder al siguiente, a menos que sea una rotonda donde el fichero será RoadX WayY R.txt.
- *Start()*: Llamará a la función Ordenar.

4.3.6. “ReparaRutas.cs”

Script utilizado únicamente para la creación de rutas, no para la simulación.

- *reparaFicherosIda()*: Comprueba si el fichero con la ruta tiene un recorrido de ida con datos correctos.

- *almacenaRutasBase()*: Se encarga de almacenar las rutas base en arrays para su posterior utilización.

- *reparaFicheroDiferenteRutaBase()*: Repara los ficheros cuya ruta base no es la correcta.

- *Start()*: Llamará a la función que el usuario le solicite, ya que éste puede necesitar reparar ficheros de ida, o ficheros con diferentes rutas base.

4.3.7. “MoviendoCamara.cs”

Script que será utilizado durante la simulación.

- *cambiarPosicion()*: Activa la cámara que el usuario solicita.
- *Start()*:Captura las cámaras existentes en la simulación, y deja activada únicamente la primera.

- *Update()*: Comprueba cada Frame si el usuario solicita cambiar de cámara.

- *OnGUI()*: Muestra en la simulación una ventana con el número de cámara activa.

4.3.8. “RealizaCamino.cs”

Script que será utilizado por cada vehículo durante la simulación, a diferencia de los demás scripts, éste actualizará sus datos cada “FixedUpdate”, esto es porque el vehículo que lo llama debe terminar de realizar sus datos antes de que termine el Frame, y por ello se utiliza “FixedUpdate”.

- *creaRuta()*: Almacena una ruta.

- *RotateTowards()*: Gira el vehículo mirando al siguiente punto.

- *camina()*: Camina si delante de él no hay ningún obstáculo, en caso de ser así, avanza según la velocidad de la carretera en la que se encuentre.
- *FixedUpdate()*:Comprueba cada Frame si el vehículo está activo, en cuyo caso llama a camina().

4.3.9. "ContadorGeneral.cs"

Script que será el encargado de realizar el tráfico de la simulación, de mostrar y almacenar los resultados obtenidos e incluso de ir avanzando en el reloj de la simulación.

- *cargaRutas()*: Carga las rutas desde su ficheros en arrays e inicializa el array que contiene las franjas horarias y los vehículos que utilizan cada una de ellas. Esta carga se realiza en arrays estáticos para obtener efectividad durante la simulación cargando la mayor cantidad de datos posible al inicio.

- *sumaHora()*: Suma una hora al contador horario, siendo ésta cíclica 0-23.

- *sumaMinuto()*: Suma un minuto al contador horario, siendo éste clic lico 0-59.

- *obtieneCocheBase()*: Obtiene el nombre de un coche base.

- *cargaVehiculo()*: Clona un vehículo, le introduce una ruta en su script, lo desactiva (ya se activará en su Hora:Minuto), se posiciona en la salida y se le indica un nombre.

- *creaVehiculos()*: Comprueba los ficheros horarios "Hora X Minuto Y.txt" compuestos cada uno de ellos por la ruta y el coche que la utilizará, obtiene los datos y los carga en un array bidimensional que simula la hora y el minuto, y se almacena en cada array [Hora][Minuto] los coches que deben actuar en esa franja horaria y la ruta que realizarán.
  Una vez cargado cada coche, accede al script de éste y le indica a su variable activo=true (para decirle que cuando pueda, circule), y

le guarda el nombre y el objeto Trafico para así tener la mayoría de datos cargados al inicio de la simulación y ahorrar cargas innecesarias durante la simulación.

- *activaVehiculos()*: Si en [Hora][Minuto] existen vehículos que deben salir a escena, se sitúan sus nombre en una fila para empezar a salir cuando puedan. Esta función únicamente se ocupa de colocarlos en la lista de salida.

- *sacaVehiculos()*: Si existen vehículos en la fila para salir, comprueba si su posición está libre y en caso afirmativo lo saca y finaliza la función, en caso negativo comprueba si existe otro vehículo para salir y realiza el mismo proceso.
  Se ha decidido sacar un único vehículo cada Frame por motivos de eficiencia, ya que esta función se activará cada frame y sobrecargarla intentando sacar todos los vehículos existentes en la fila provoca sobrecarga en cada frame de la simulación.

- *destruyeVehiculo()*: Función que eliminará de la lista de coches activos, el coche que la invoque. El coque que invoca esta función, la indica la distancia y consumos realizados en su trayecto.

- *tiempoDinamico()*: Se encarga de avanzar el tiempo durante la simulación, pero haciendo que durante las horas puntas el paso de este tiempo sea algo más lento.
  En cada minuto que avanza, llama a la función activaVehiculos para activar los que deban hacerlo.
  Si la hora está entre las 6:00 y las 21:00 llama a apagaFarolas, en caso contrario, si están apagadas, llama a enciendeFarolas.

- *OnGUI()*: Muestra por pantalla el nombre del proyecto, el reloj de la simulación y los datos del tráfico.

- *capturaFarolas()*: Captura en un array las luces de todas las farolas de la ciudad.

- *apagaFarolas()*: Apaga las farolas de la ciudad.

- *enciendeFarolas()*: Enciende las farolas de la ciudad.

- *Start()*: En la carga de la simulación, llama a las funciones cargaRutas, creaVehiculos y capturaFarolas, y llama cada 2 segundos a la función tiempoDinamico.
  Esta función realiza la carga de todos los datos posibles al inicio de la simulación, e invoca cada 2 segundos a tiempoDinamico para ir avanzando durante la simulación con coherencia.

- *Update()*: Cada frame comprueba si hay vehículos por sacar a escena, y en caso de ser así, llama a sacaVehiculos.
  Si el usuario pulsa la tecla "ESC" o la simulación ha llegado a las 23:59 y todos los vehículos han terminado sus trayectos, finaliza la simulación. Al finalizar la simulación almacena los datos obtenidos durante la misma.

4.4. Funcionalidad

Una vez obtenida la especificación de requisitos, el diseño y la implementación del proyecto, se pasa a explicar la funcionalidad separando el proyecto en tres fases: creación de objetos, creación de escenario y creación de simulación.

Esta fase se realiza debido a las características del proyecto, y mostrar así al usuario las posibilidades de las que dispone a parte de la ejecución de la simulación.

4.4.1. Creación de objetos

En esta fase, se explica al usuario cómo puede crear o modificar un objeto que posteriormente podrá utilizar en un escenario.

El programa utilizado para el diseño de objetos en 3D ha sido 3DStudio Max, el cual no es gratuito, por lo que el usuario puede optar por utilizar el programa freeware Blender.

Debido a la complejidad en el diseño 3D, en esta memoria únicamente se mostrará un breve resumen sobre las opciones de las que dispone el usuario con el programa 3D Studio Max para la modificación de los objetos existentes en el proyecto.

Los objetos se encuentran en la carpeta Diseño 3D, separados por su utilidad en el proyecto:

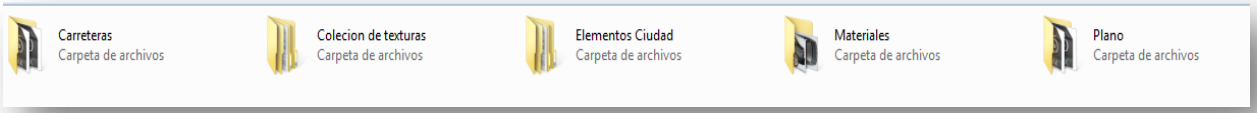

Figura 6. Descripción carpeta “Diseño 3D”

El usuario dispone de las carreteras que forman la ciudad, de una colección de texturas que poder aplicar a los objetos, de materiales de los objetos creados, de un plano el cuál ha sido utilizado (basándose en un mapa real de la ciudad de “Las Rozas de Madrid”) y de una colección de objetos que componen la ciudad.
En esta breve explicación, se mostrará un vehículo disponible en el escenario y las opciones de modificado que tiene el usuario:

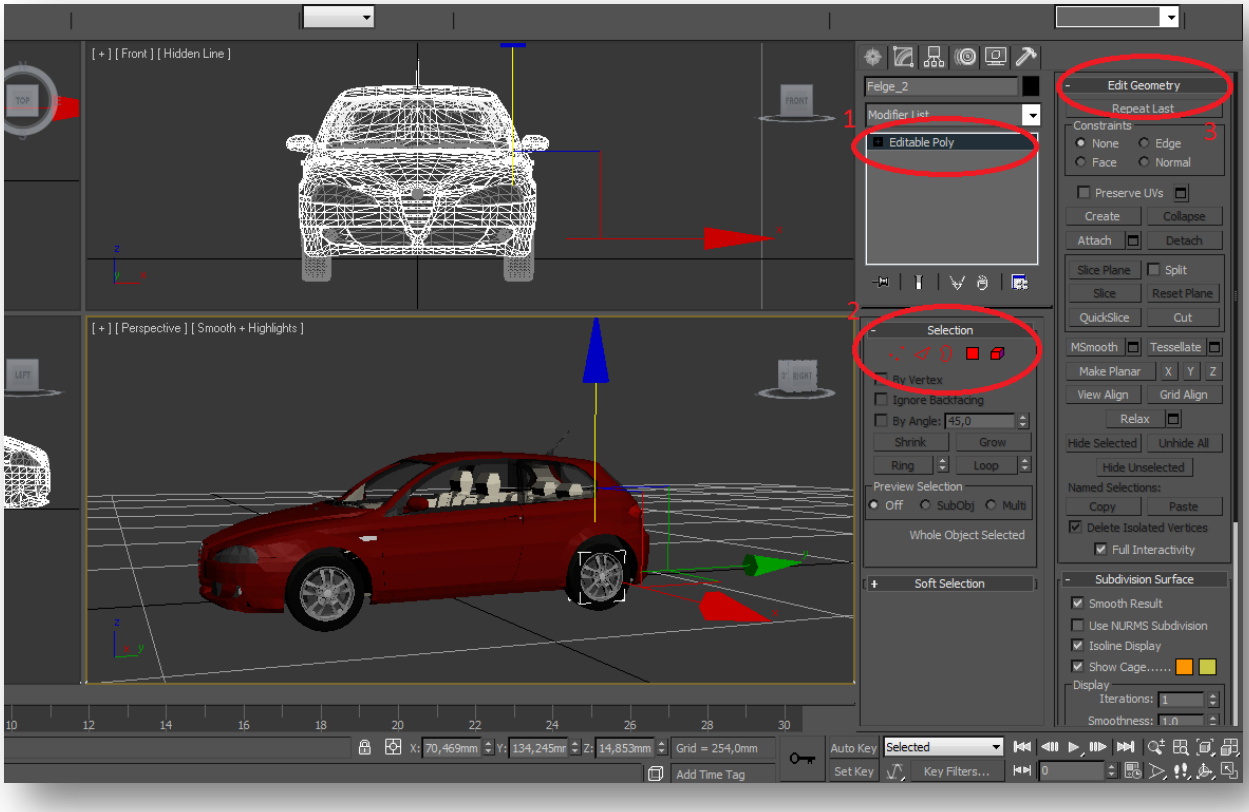

Figura 7. Diseño con 3D Studio Max

Como se observa en la figura 2, se va a explicar las tres principales características para modificar un objeto, pero 3D Studio Max dispone de múltiples opciones para el tratamiento de los objetos.

4.4.1.1. Editable Poly

El usuario dispone tres formas de editar el objeto: Poly, Mesh, Patch. Si éste no está familiarizado con la terminología de diseño en 3D, es recomendable que utilice Editable Poly, así puede tratar al objeto como un compuesto de polígonos, siendo ésta la forma más común de tratar los objetos.

En este ejemplo (y en la mayoría de los objetos del proyecto), un objeto está compuesto por varios objetos, y cada objeto que trate el usuario lo hará de la forma "Editable Poly". Si se desea modificar el tratamiento de un objeto a Mesh o Patch, únicamente debe seleccionar el tipo de tratamiento.

4.4.1.2. Selection

Esta parte trata el objeto según desea el usuario, esto es, el usuario puede desear seleccionar una parte en concreto del objeto, todas… Por ello, Selection permite al usuario tratar el objeto de las siguientes formas:

. ***Vertex***: Permite capturar únicamente los vértices del objeto, para así trabajar con ellos.
. ***Edge***: Permite capturar únicamente las líneas que componen al objeto y trabajar con ellas.
. ***Border***: Permite capturar las líneas que componen los bordes de los elementos del objeto y trabajar con ellas.
. ***Polygon***: Permite capturar los polígonos de un objeto y trabajar con ellos.
. ***Element***: Permite capturar el objeto completo y trabajar con éste.

Estas opciones consisten en dar al usuario la opción de tratar el objeto por partes, ya que para poder modificar de una forma correcta un objeto se necesita hacerlo ya sea moviendo un

vértice, cambiando de posición una línea, o incluso eliminando o duplicando un polígono.

4.4.1.3. Edit Geometry

Esta opción permite al usuario, una vez ha decidido la forma de tratar el objeto (es decir, tratar sus vértices, polígonos, líneas…), le da múltiples facilidades para trabajar con éste. Algunas opciones para el tipo Polygon son:

- Mostrar vértices
- Suavizar
- MSmooth, que suaviza elementos con curvas añadiendo una serie de vértices intermedios.
- Attach, unifica el polígono con otro/s.
- Cortar
- Editar triangulación
- Insertar
- Etc.

Esta es la parte más complicada en el diseño 3D, ya que, a las múltiples opciones que ofrece 3D Studio Max se le añaden plugins que ofrecen una mayor cantidad de opciones de tratado de un objeto.

4.4.2. Creación de escenarios

En esta fase se estudia la creación de los casos base, y la utilización de los scripts y diseños anteriormente citados.
Para esto, el usuario debe acceder a la carpeta Simulacion\Assets\Scenes del proyecto, y seleccionar el archivo CasoBase.unity.
Indicar que para poder abrir el caso base, el usuario deberá tener instalado el programa Unity3D, siendo éste de licencia gratuita.

En esta fase se mostrará principalmente cómo utilizar los scripts explicados anteriormente, para ello se dispone de un objeto que tiene adjunto cada uno de los scripts y éstos están inicialmente desactivados, para que el usuario pueda seleccionar el script que desea actúe al ejecutar el programa.

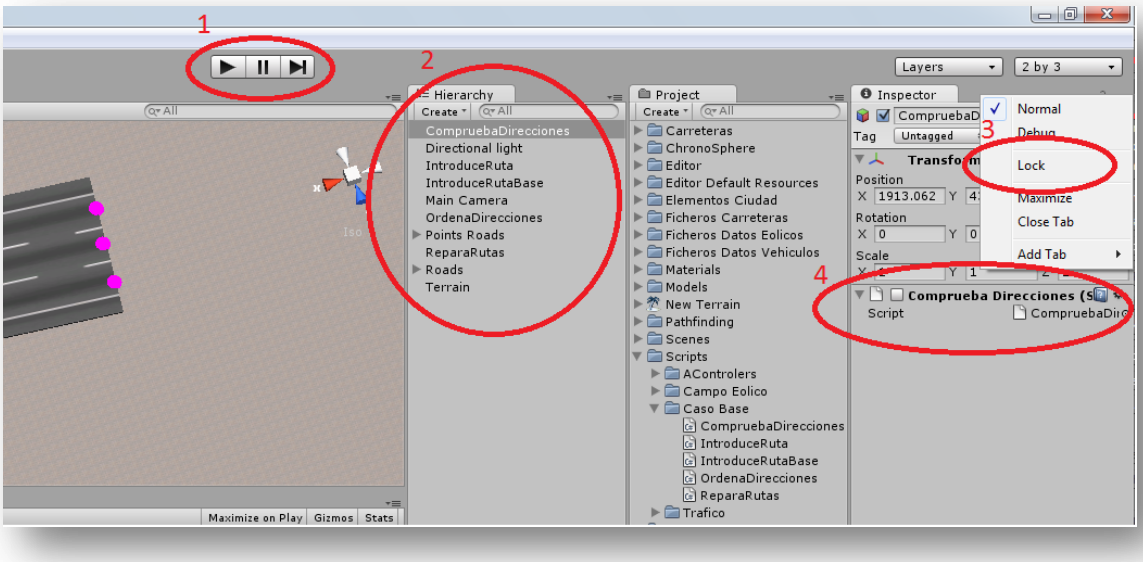

Figura 8. Utilidad Unity3D CasoBase

A continuación se explica el funcionamiento de los puntos indicados en la figura 3:

1. Ejecución, parada y finalización del escenario indicado
2. Objetos que componen el escenario (seleccionado objeto CompruebaDirecciones)
3. Bloquea esa pestaña, opción utilizada para arrastrar objetos desde la pestaña “Hierarchy” al script.
4. Script adjunto al objeto CompruebaDirecciones, el cual está desactivado como se observa en su pestaña de activación.

Una vez vista la funcionalidad básica del programa, se procede a explicar qué datos se deben insertar en cada script:

CompruebaDirecciones: Para utilizar este script, el usuario únicamente debe activarlo y ejecutar el programa.
No necesita la entrada de ningún dato.

IntroduceRuta: Para utilizar este script, el usuario debe introducir la cantidad de vehículos que desea que realicen la ruta, si ésta utiliza un atajo deberá indicar el nombre del fichero que contiene dicho atajo e indicar si se trata de un atajo para la ida de la ruta o para la vuelta.
Debe introducir los puntos que componen el trayecto de la siguiente forma:

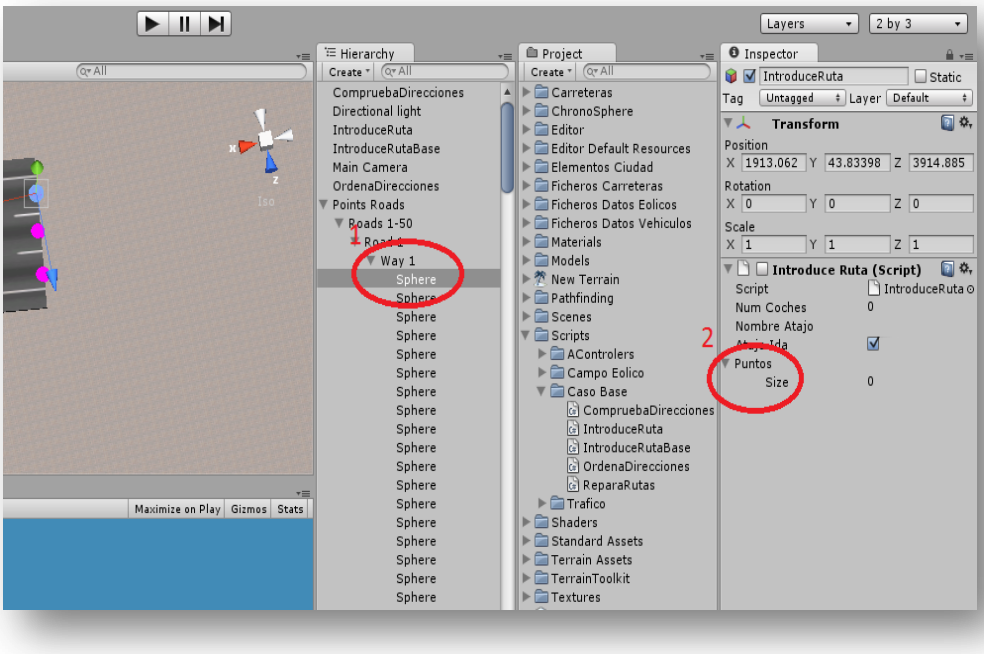

Figura 9. Adjuntar puntos

El usuario deberá seleccionar el punto obtenido en la pestaña “Hierarchy” como se muestra en la figura 4, en el punto circular 1. Seguido deberá arrastrar el punto al punto circular 2, el cuál contendrá los puntos de la ruta.

Antes de realizar este traspaso de puntos, el usuario debe bloquear (lock) la pestaña “Inspector” como se ha indicado en el punto anterior.

IntroduceRutaBase: En este script, el usuario debe introducir el nombre que desea tenga la ruta base destino y moverá los puntos que desea formen parte de dicha ruta como se ha explicado en el punto anterior.

OrdenaDirecciones: En este script, el usuario no debe introducir ningún dato. Únicamente debe activarlo y realizar una ejecución.

ReparaRutas: Como se ha indicado anteriormente, este script se encarga de analizar rutas donde el usuario cree que existe algún problema. Para ello, debe introducir el nombre de la ruta que cree está errónea, y los dos puntos donde cree que existe algún tipo de error.

4.4.3. Creación de una simulación

En esta fase se estudia la creación de una simulación, y la utilización de los scripts y diseños anteriormente citados que afectan a este diseño.
Para esto, el usuario debe acceder a la carpeta Simulacion\Assets\Scenes del proyecto, y seleccionar el archivo WDistrict.unity.

En esta fase, los scripts explicados anteriormente y adjuntos a sus objetos, estarán activados, debido a que se desea que durante la creación de la simulación éstos actúen. Por ello se explicará los casos para los que el usuario puede modificar ciertos aspectos, para crear su simulación y qué datos modificables tiene el usuario en dicha creación.

Campo Eolico: Este objeto tendrá adjunto el script "CampoEolico", donde el usuario puede indicar si desea que los datos sean cargados desde fichero o aleatoriamente.
El script muestra otros datos, pero éstos no son de entrada, por ello estos datos no son necesarios que sean modificados.

CocheB N: Donde N es un número de vehículo, este script tiene datos para introducir por el usuario, pero no son de entrada, ya que dichos datos serán cargados por el objeto Trafico al ejecutar la simulación.

Trafico: Este objeto dispone de dos datos de entrada, los cuáles son la hora y minutos. El usuario debe indicar en qué hora y minuto desea que empiece la simulación, teniendo en cuenta que la misma finaliza cuando el reloj de la simulación indica 23:59 y los trayectos de los vehículos han finalizado.

Una vez el usuario ha realizado la introducción de datos, éste debe generar el fichero destino que contendrá la simulación.
Para ello se sigue la siguiente figura:

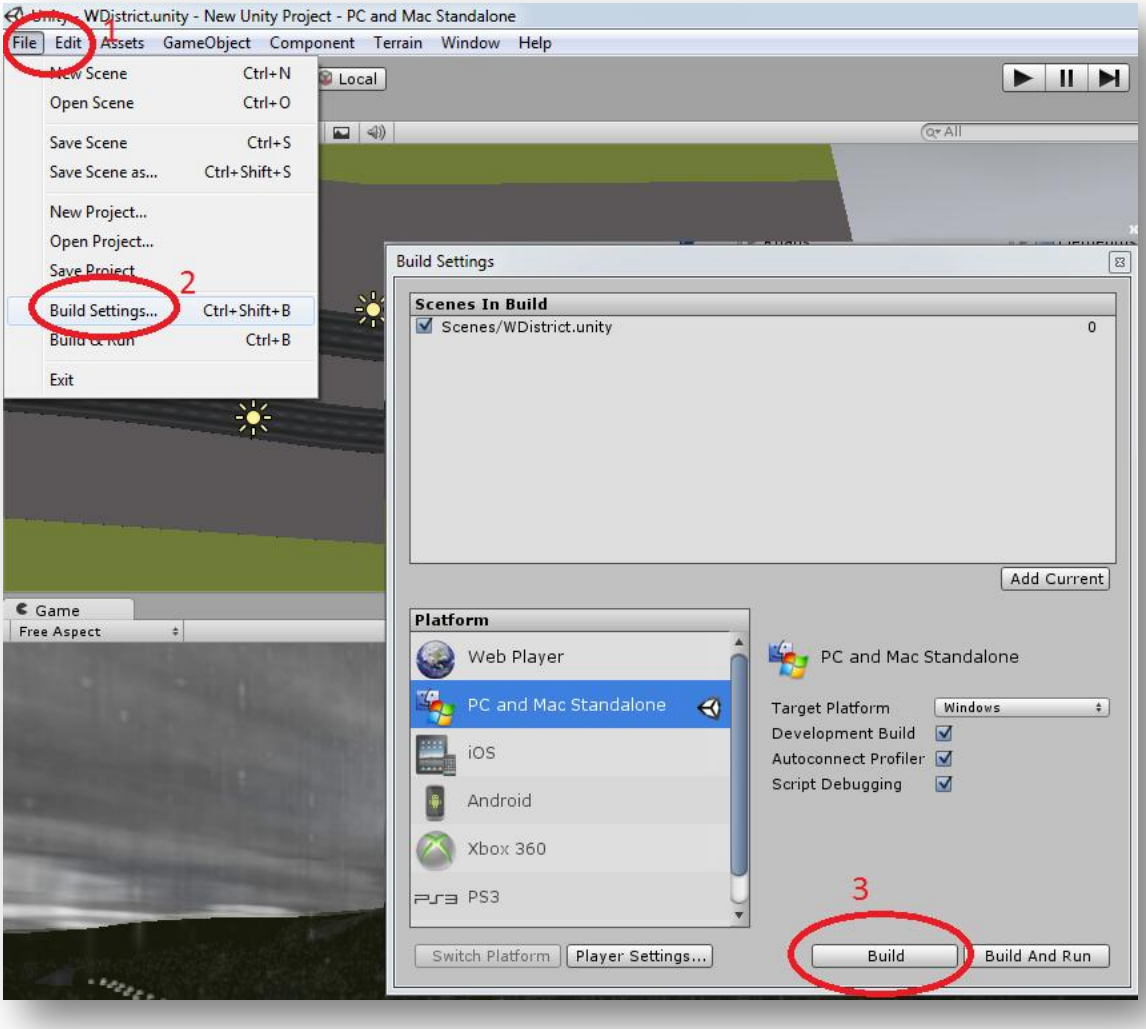

Figura 10. Creación de simulación

1. Accede a File
2. Selecciona Build Settings, para seleccionar opciones del fichero a generar. Este proyecto se ha creado para la plataforma PC, por lo que si el usuario desea otra plataforma debe cambiar la forma de acceder a los ficheros utilizados por la simulación.
3. Build, generará el fichero destino que ejecutará la simulación.

Importante indicar, que el usuario debe eliminar la simulación existente si desea construir otra, debido a problemas de Unity3D. También debe situar la simulación creada en la carpeta \Assets\, esto es debido a que los scripts creados que llaman a los ficheros a utilizar y el fichero con los datos obtenidos durante la simulación llaman a ésta dirección.
Si el usuario desea crear una simulación para otra aplicación o modificar la carpeta de destino, debe hacerlo en los scripts que utilizan dichos ficheros.

5. Conclusiones y trabajos futuros

Al finalizar el desarrollo del proyecto, se han alcanzado los objetivos propuestos, dando como resultado la simulación buscada que nos muestra los datos deseados. Por ello, se llega a la conclusión de que los objetivos buscados han sido cumplidos.

Para la realización del proyecto, se han abordado dos programas de los cuáles se desconocía su funcionamiento, ampliando así los conocimientos del autor sobre dos áreas (el diseño y la creación de escenarios con objetos inteligentes) completamente desconocidas para él.

Para la creación de objetos en 3D, he aprendido absolutamente todo sobre el diseño de objetos, modelado, introducir colores, luces, etc.

Para la creación de la ciudad y la simulación de la misma, a pesar de aprender creación de escenarios mediante Unity3D y el perfeccionamiento de la programación en C#, he aprendido mucho sobre la mejora en la eficiencia en scripts. Esto ha sido debido a problemas que la simulación creaba a la hora de simular los 20.000 vehículos y realizar sus movimientos durante cada frame, por ello, este proyecto ha mejorado también mi capacidad de eficiencia sobre la programación.

Como trabajos futuros cabría la posibilidad de crear nuevos objetos que formen parte de la ciudad, pero sobretodo buscar una mejora de eficiencia para el desarrollo de la simulación y así poder aumentar la cantidad de vehículos a cargar en la simulación.

Bibliografía

Wikipedia: http://es.wikipedia.org

Dariush Derakhshani, Randi L. Munn - *Introducing 3DS MAX 9: 3D for beginners* – Autodesk authorized publisher.

Autodesk – *Help: Autodesk 3Ds MAX 2010* – Autodesk

3DPoder - http://www.foro3d.com/foro3d.php

Jeff Ferguson, Brian Patterson, Jason Beres – *La biblia de C#* - Anaya

Paul Kimmel – *Advanced C# programming* – Obsorne

Foro UnitySpain - http://www.unityspain.com/

Foro Unity - http://forum.unity3d.com/

Scripting Unity - http://unity3d.com/support/documentation/ScriptReference/index.html

Apéndices

Como se ha mencionado anteriormente, la simulación creada funciona bajo el sistema operativo Microsoft Windows y no es necesario tener instalado el programa Unity3D.

Para su ejecución únicamente se debe ejecutar el fichero Simulacion.exe, situado en la carpeta \Simulacion\Assets\Simulacion.exe.

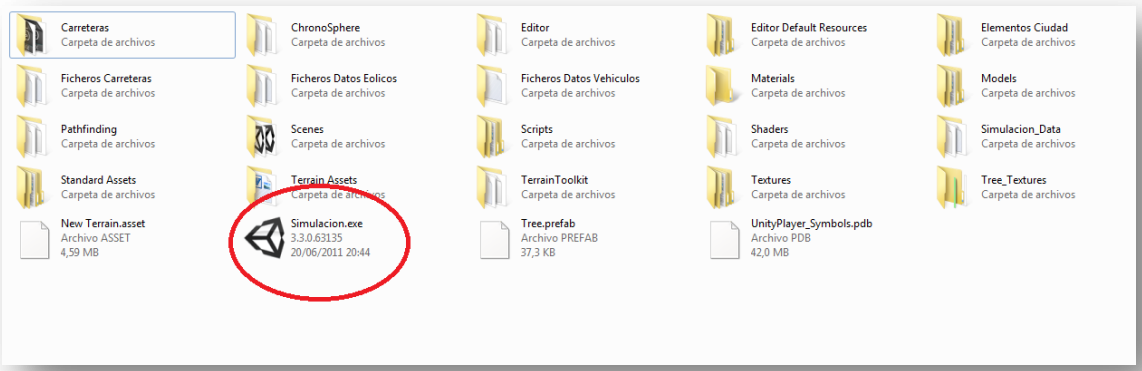

Figura 11. Simulacion.exe

En la figura 11, se puede observar el contenido de la carpeta Simulación. La obligatoriedad de crear la simulación bajo la carpeta \Assets, es debido a que los scripts que realizan llamadas a ficheros, lo realizan desde la posición donde está ubicado el ejecutable y desde ahí busca la carpeta donde están los ficheros necesarios. Luego si el usuario desea crear una simulación en una carpeta diferente, deberá añadir en la carpeta donde ubique el ejecutable de su simulación, las carpetas: “Ficheros Carreteras”, “Ficheros Datos Eolicos” y “Ficheros Datos Vehiculos”.

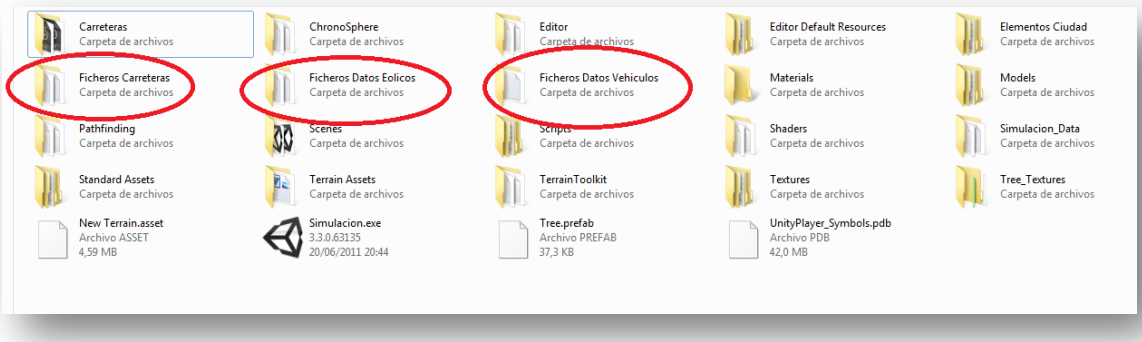

Figura 12. Carpetas a adjuntar

Para la ejecución de la simulación creada en este proyecto, se puede realizar de dos maneras:

- Ejecutando directamente el fichero Simulació.exe desde el DVD adjunto a este proyecto, como se indica en la figura 11.

- Copiando las carpetas “Ficheros Carreteras”, “Ficheros Datos Eolicos”, “Ficheros Datos Vehiculos”, “Simulacion_Data” y los ficheros “Simulacion.exe” y “UnityPlayer_Symbols.pdb”. Estos son las carpetas y ficheros que componen la simulación, el resto de carpetas contenidas en la carpeta padre “\Assets” forman parte del escenario creado con Unity, por lo que si el deseo del usuario es copiar únicamente la simulación creada, con las carpetas indicadas sería suficiente.

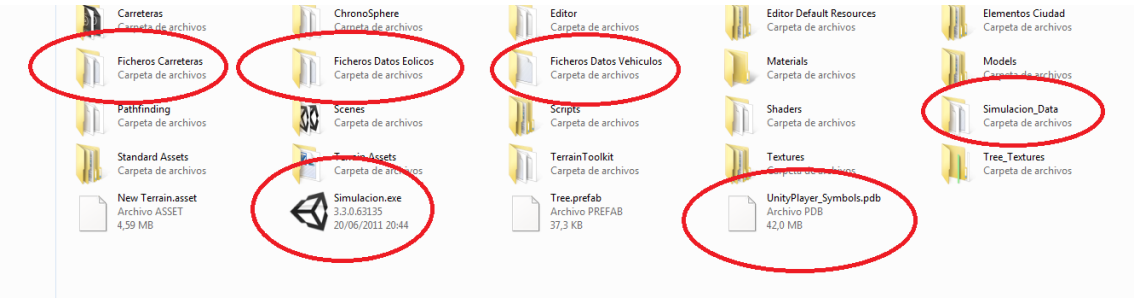

Figura 12. Carpetas necesarias para la ejecución de Simulacion.exe

Esto crea una independencia de la simulación creada, con el programa mediante el cual ha sido diseñado. El resto de carpetas forman parte del escenario creado con la utilidad Unity3D, por lo que si el usuario desea cargar la ciudad creada durante este proyecto necesitará las sub-carpetas de “Simulacion”, adjunta al DVD del proyecto.