Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Librería en Unity3D para la generación de entornos viales realistas

Estudante: Sergio Fernández Barreiro
Dirección: Enrique Fernández Blanco
Dirección: Ismael Barbeito Vázquez

A Coruña, novembro de 2020.

A mi clan.

Agradecimientos

Me gustaría dedicar esta sección a todas aquellas personas que han estado presentes en mi vida durante el transcurso de estos años de universidad. No ha sido fácil, pero aquí estoy y vosotros habéis estado siempre conmigo. Gracias a toda mi familia por ayudarme a avanzar paso a paso en cada una de las etapas que ha supuesto mi desarrollo emocional y profesional, por ayudarme a convertirme en lo que soy actualmente y por saber que aunque no hubiese llegado hasta aquí, seguirían siempre pendientes de mí. Mi etapa de universitario cierra con este trabajo, el cual he tenido la suerte de poder elegir y realizar sobre un tema que me gusta. Ajenos a mi familia… a Ismael, persona no relacionada con la universidad que decidió dedicarme horas de su tiempo para ayudarme y guiarme durante la realización de este proyecto, así pues a Enrique por habérmelo presentado y por las clases más entretenidas que he tenido, no quiero tampoco olvidar a los demás profesores que me han guiado para obtener conocimientos necesarios para el desarrollo de este proyecto así como a todos los compañeros de promoción (o de otras) los cuales han hecho que el viaje fuese más llevadero.

Finalmente, a mi novia, Eugenia. Persona que siempre está a mi lado y no me ha fallado en los momentos decisivos de mi vida, tomando a veces el rol de madre o de amiga, dependiendo de mis necesidades, te quiero.

Resumen

Durante las últimas décadas, el entorno de los videojuegos y simuladores ha evolucionado de una manera que nadie pudo imaginar, alcanzando un exito sin precedente.

Los juegos en los que se incluyen grandes ciudades, simuladores, carreras o incluso trabajos propios de la vida real [1], son aquellos en los que más se extiende la aparición de vehículos, humanos u otros elementos con los cuales el usuario puede interactuar o simplemente disfrutar de la ambientación. Esto es lo que genera la sensación de realidad o inmersión que el usuario busca, haciendo que sea capaz de estar delante de una pantalla conduciendo un vehículo o actuando como si se sintiera en el interior del videojuego.

Esta librería es un complemento ideal para dotar al mundo virtual creado de vida propia, añadiendo un sistema sencillo de carreteras y vehículos que las manejará la IA de manera totalmente ajena al desarrollador y al usuario. Se permite de este modo que el desarrollador no tenga que hacerse cargo del tedioso trabajo de preparar un sistema de circulación para su entorno. Por otra parte, el establecer dicho sistema, otorgará al usuario de mayor inmersión en el mundo, es decir, el hecho de que el entorno disponga de vehículos artificiales con vida propia dará una mayor sensación de realismo. El resultado final de este trabajo es una herramienta que se ha denominado Easy Road System o ERS.

Abstract

During the last decades, game developing and simulators had evolved in a way that nobody could have imagined, reaching incredible success.

Games that include big cities, races, simulators or even real life jobs [1], are the ones with vehicles IA, humans or many other elements that you can interact with or simply enjoy the enviroment. This is what generates the sensation of reality or inmersion making the user feel as if he is part of the game.

This library is an ideal complement for giving the virtual world its own life, adding a simple road and vehicle system that will be handled by the AI in an autonomous way. This allows the developer to avoid the teious work of preparing a system for his environment. On the other hand, establishing this system will give the user a greater inmersion into the world. The fact that the environment has artificial vehicles with their own life, will encourage the user to feel the realism it wanted to. The final result of this work is a tool that has been called Easy Road System or ERS.

Palabras clave:

- Unity3d
- Tráfico
- Vial
- Carreteras
- Señales
- Vehículos
- Circulación
- Semáforo
- Librería

Keywords:

- Unity3d
- Traffic System
- Roads
- Signals
- Vehicles
- Circulation
- Traffic Light
- Library

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto

La sensación de inmersión dentro de un entorno virtual es la que en muchos casos determina el grado de satisfacción de un usuario con el mundo que le rodea. Así pues no se concibe un entorno sin ciertos elementos que otorgan lo comentado anteriormente, como pueden ser árboles dentro de un bosque, las piezas mecánicas dentro de un taller, basura, carteles u otros elementos dentro de una ciudad que harán que el ambiente se parezca lo máximo posible al que nos rodea.
Esta herramienta en concreto, se encargará de dotar a la escena una serie de elementos: carreteras, señales, vehículos... que pueden ser utilizados como el usuario crea más conveniente. Por ejemplo, en el caso de una ciudad, se encargaría de la colocación y utilización, si así se cree oportuno, de un entorno en el cual los elementos comentados anteriormente se manejen de manera aislada al usuario. De esta manera se logran dos objetivos importantes, que el desarrollador no tenga que estar pendiente de crear su propio sistema y que el proyecto final cumpla las expectativas del usuario.
La sociedad actual ha estipulado una serie de normas a seguir para la correcta circulación de vehículos. Estas normas vienen estipuladas dentro del código de circulación vial, en las cuales ERS (Easy Road System) se basa. De este modo, si un semáforo le dice a nuestro agente que debe detenerse, éste lo hará. Si identifica una señal con un nuevo límite de velocidad máxima, deberá acatarla.

1.2 Motivación

El tiempo y coste de un proyecto son un factor determinante a la hora de la realización de éste. Tanto un desarrollador independiente como un equipo grande, pueden tener problemas a la hora de lidiar con diversos aspectos que pueden ser fundamentales en cuanto al desarrollo y

que en un principio no pensaban encontrar. Si se lograse que el equipo encontrase una forma alternativa, sencilla y económica de solventar una situación que le está llevando más recursos de los previstos, podría plantearse utilizarla. Debido a esto, surge la idea de abstraer a los desarrolladores del tedioso trabajo de crear un sistema de circulación en su entorno, adaptable, intuitivo y sobre todo funcional. Es por eso que el desarrollo de esta librería facilitaría en un alto grado la implementación y mantenimiento del entorno ya comentado, tanto para alguien que se haya metido en el mundo para aprender como para un profesional.
Por todo esto, la creación de una librería totalmente gratuita, de facil acceso y uso, será un factor clave a la hora de que el desarrollador se decante por la utilización de una herramienta u otra.

1.3 Objetivos

El objetivo principal de este proyecto es: desarrollar una librería para Unity3D, que se encargue de implementar de manera sencilla un entorno de circulación vial.
En este proyecto, los agentes deberán de seguir unas pautas (normas de tráfico), la inmesa mayoría de ellas vendrán establecias en el código de circulación español.
Teniendo este objetivo en mente, para lograrlo se procede a identificar una serie de objetivos particulares y más concretos. Estos son:

- Generación de entornos viales realistas.
- Identificación del entorno (señales) que los agentes tienen que respetar.
- Colocación de carreteras, señales y agentes.
- Facilidad de integración de la librería en cualquier proyecto.
- Simplicidad de utilización mediante una interfaz.
- Posibilidad de añadir vehículos determinados por el usuario.
- Garantizar el funcionamiento autómata de cada agente, sin la implicación del usuario.

1.4 Estructura del documento

Para ilustrar el proceso de desarrollo, así como presentar los diferentes elementos de una forma estructurada, el presente trabajo se ha concretado en los siguientes temas:

1. **Introducción:** capítulo actual, en él están los elementos mencionados anteriormente, un resumen del contexto del proyecto, la motivación que existe detrás del mismo y los objetivos que se pretenden alcanzar con él.

2. **Alternativas:** es un análisis de las distintas opciones que se pueden elegir a la hora de realizar la librería, tanto del motor gráfico, como de otras librerías existentes.

3. **Fundamentos:** descripción de varios elementos que han tenido que establecerse previamente al desarrollo de la librería.

4. **Planificación:** es una vista de manera global del desarrollo inicial y final del proyecto. En éste capítulo se puede observar el tiempo previsto para la elaboración de Easy Road System así como los gastos que se han tenido durante su desarrollo.

5. **Desarrollo:** capítulo en el que se muestran las distintas iteraciones por las cuales la librería ha pasado, conteniendo cada una de las mismas un establecimiento de objetivos, un análisis, la propia descripción del desarrollo, las pruebas a las que ha sido sometida la librería y la planificación de la siguiente iteración.

6. **Conclusiones:** recapitulación de todo lo aprendido durante el transcurso del proyecto.

7. **Trabajo futuro:** conjunto de ideas *nice-to-have* de cara a versiones futuras.

8. **Lista de acrónimos:** lista de palabras utilizadas provenientes de otras.

9. **Glosario:** lista de términos utilizados en el proyecto.

10. **Bibliografía:** lista de referencias utilizadas en el proyecto.

Capítulo 2

Alternativas

2.1 El porqué de Unity3D

Unity3D es un motor gráfico que lleva activo desde el 2005. Gracias a esto, ha ido creando una comunidad cada vez más grande haciendo que sea más que común ver páginas de ayuda o foros en los que la gente se reúne para comentar ideas, problemas o cualquier otro tema.
El motor Unity3D provee un gran abanico de utilidades llamados componentes, que se pueden utilizar para añadir diversas características al entorno creado. De este modo, se consigue personalizar cada uno de los elementos del mundo virtual.
La forma que tiene Unity de "introducir" una clase dentro de un objeto es mediante la adición de *scripts*.
Un script es un componente que contiene código que será ejecutado durante el funcionamiento del proceso. Con lo cual, existen acciones que se lanzan en un instante determinado del proceso, al cargar un objeto, que se ejecute durante todo el rato que el objeto esté vivo, cuando el objeto desaparece o cuando se cumplen determinadas condiciones, como por ejemplo, que un objeto colisione contra otro.
Estos scripts mejoran su funcionamiento mediante la derivación de otros. Un ejemplo de esto es la clase **MonoBehaviour**, la cual contiene métodos que ejecutan el código en ciertas ocasiones como *Start()*, ejecutado nada más el objeto se carga o *Update()* el cual se ejecuta una vez por frame.
Debido a todas las características mencionadas anteriormente, se ha elegido Unity3D ante otras opciones como Unreal Engine o jMonkey. Para más información consultar el anexo B.

2.2 Estado del arte

Actualmente existen varias herramientas o librerías que se encargan de solventar todos los quebraderos de cabeza que implementar un sistema de circulación ocasiona.

Pese a que Unity3D posee una gran comunidad por detrás, no existe ninguna librería gratuita que cubra con todo lo anteriormente dicho. Hay ejemplos de librerías muy completas con precios totalmente desorbitados, otras no tan completas con precios más o menos asequibles para la mayoría de la gente y otras que no cumplen la mayoría de las cosas que prometen pese a pagar por ellas. De este modo, la librería contiene muchos de los aspectos más relevantes de cada una de las herramientas al alcance de los usuarios, buscando siempre que el desarollador se ahorre el esfuerzo de tener que cogerlas una por una e implementarlas por su cuenta. También contiene diversas características que no están presentes en ninguna de las otras herramientas como puede ser el reconocimiento por parte de los agentes de señales de velocidad en vez de tener una velocidad estática todo el rato o que simplemente varíe segun una variable presente en la vía por la que circule.

2.3 Otras opciones

Proyectos viales de Unity							
Nombre	Editor	Sencillez	Accesorios	Vehículos	Peatones	Full	Precio
Urban Traffic System	S	S	S	S	S	N	357€
EasyRoads3D	S	S	S	S	S	N	50€
Unity Road Tool	S	S	S	S	N	N	N/D
Road Traffic System	S	N	N	S	N	S	23€
Easy Road System	S	S	S	S	N	S	0€

Tabla 2.1: Tabla comparativa de algunas librerías similares a Easy Road System.

Pese a la existencia de varios proyectos que tratan sobre la creación de entornos viales, se han tomado varios ejemplos recogidos de la Unity Store o de páginas personales a modos de proyecto que nunca llegaron a ver la luz.

A continuación se describirá brevemente cada uno de los componentes de la tabla:

- **Nombre**: nombre del proyecto.
- **Editor**: Existencia de un editor propio en la librería.
- **Sencillez**: Facilidad de colocación y configuración de elementos en la escena.
- **Accesorios**: Existencia de semáforos o señales funcionales.
- **Vehículos**: Existencia de vehículos que se puedan configurar de manera sencilla.

- **Peatones**: Existencia de peatones que se puedan configurar de manera sencilla.
- **Full**: Describe si la versión descargada está completa o se necesita descargar paquetes adicionales.
- **Precio**: Cantidad de dinero a desembolsar en caso de compra. *(N/D significa que no se puede descargar)*

En la tabla 2.1, cabe destacar el Urban Traffic System, un sistema muy completo de circulación en el cual vienen incluidos peatones o la posibilidad de conducir tú mismo el vehículo. La gran pega es que no está al alcance del todo el mundo, ya que sólo la versión básica cuesta un desembolso de 360€.
Por otra parte, el EasyRoads 3D es una solución bastante más económica pero con algunas carencias que puede que se traduzcan en una posible pérdida de interés por parte del usuario, como la ausencia de vehículos.
El Unity Road Tool es una herramienta que contiene bastantes características interesantes, tales como un editor en 3D impresionante. El problema es que no está disponible debido a que es un proyecto privado.
Road Traffic System es una solución económica, pero que no solventa todos los problemas a los cuales este proyecto está enfocado. Un montón de comentarios negativos en el apartado de reseñas, falta de semáforos (sólo son decorativos), complicaciones de montaje...

Dicho esto, se puede afirmar que no existe ninguna librería que sea completa en todos los ámbitos y que sea alcanzable por todos los públicos, así pues ésta se ha vuelto la motivación de este proyecto.

2.4 Software utilizado:

Para el desarrollo de ERS ha sido necesaria la utilización de diversos programas:

- *Unity3D* [2]: es el motor gráfico utilizado, es el único software totalmente necesario para el desarrollo de la librería.
- *Adobe Illustrator*: es una herramienta de edición de imágenes, ha sido utilizada para el diseño de las texturas que se han utilizado en la librería.
- *TeXworks*: ha sido el programa encargado de traducir las instrucciones introducidas en LaTeX para construir este documento.
- *Visual Studio Code*: es el entorno de desarrollo que Unity3D facilita con su instalación.

Nota:el ordenador dispone de acceso a internet como recurso para la obtención de información.

Capítulo 3

Fundamentos

3.1 Modelo de ciclo de vida

Existen multitud de ciclos de vida para el desarrollo de un proyecto, como pueden ser secuenciales, en cascada, en espiral, en V, metodologías ágiles... Lo primero, indicar que todos los ciclos de vida pueden ser aptos para cualquier tipo de proyecto, teniendo siempre presente las prioridades, limitaciones y capacidades del mismo. A continuación se exponen los diversos argumentos que se tuvieron en cuenta para la elección del mismo:

- **Número de desarrolladores** 1 desarrollador
- **Tiempo** Limitado, aproximadamente mes y medio.
- **Conocimiento del problema del desarrollador** bajo-medio
- **Conocimiento del entorno del desarrollador** medio-bajo

Tras poner las cartas sobre la mesa, se descartan varios de los ciclos de vida, como los que requieren a más de una persona, por ejemplo aquellos que utilizan revisiones a pares. Finalmente valorando los pros y contras de cada una de las opciones, los ciclos de vida que más fuerza cobran son scrum y espiral.

3.1.1 Scrum

Este modelo de ciclo de vida, está basado en metologías ágiles [3]. Mediante el uso de buenas prácticas se basa en el desarrollo incremental del producto utilizando bloques temporales fijos y cortos, como puede ser una semana o dos. Scrum está ideado para equipos pequeños de 3 a 9 personas dándole prioridad a los que más valor le aporta al cliente. Se realizan adaptaciones diarias a los objetivos. Todo esto busca resultados rápidos, flexibilidad, mitigación de riesgos, productividad y calidad.

3.1.2 Espiral

Se trata de un modelo de desarrollo de software [4] en el cual se repiten las etapas repetidas veces, aumentando y refinando cada vez más tanto el objetivo como el proyecto. Está basado en cuatro estapas:

- Determinar objetivos
- Analizar los riesgos
- Desarrollo y pruebas
- Planificación

Así pues, une varias características de varios modelos, el análisis de riesgos se hace de manera explícita y clara, añade la posibilidad de tener en cuenta mejoras para la siguiente iteración [4].

3.1.3 Decisión

Pese a que ambos ciclos de vida son totalmente válidos para este tipo de proyecto, se sopesaron los pros y contras de cada de uno de éstos, optando finalmente por la utilizacióndel ciclo de vida en *espiral*. La razón final entre la utilización de uno u otro es que scrum está pensado para equipos y no para desarrolladores solitarios. Esto hace que este modelo encaje mejor con el tipo de proyecto.

3.1.4 Desglose del modelo en espiral

El desarrollo del proyecto constará de las siguientes fases aquí listadas:

1. Determinación de objetivos a alcanzar, requisitos y análisis de riesgos.
2. Análisis, diseño e implementación de la librería, que contenga los objetivos más básicos alcanzables, así pues definición del movimiento de los agentes.
3. Análisis, diseño e implementación de una herramienta capaz de gestionar la adición de elementos circulables y normas a respetar (carreteras y señales)
4. Análisis, diseño e implementación de elementos más complejos de la librería, como pueden ser atascos, adelantamientos, situaciones excepcionales…
5. Optimización y pulimiento de los diversos métodos a utilizar e interfaces generadas.

A parte de las fases que han sido listadas más arribas, se realizará la documentación del mismo. Ésta no se encuentra listada entre las tareas dado que es una tarea constante que se irá realizando durante todo el proceso de elaboración a lo largo de las diferentes etapas.

3.2 Algoritmos de búsqueda

Antes de comenzar esta sección, es necesario aclarar algunos conceptos básicos que se van a utilizar, como la definición de algoritmos de búsqueda, los distintos tipos que hay y cuales son útiles para este proyecto...
Así pues, un algoritmo de búsqueda puede definirse como un conjunto de instrucciones lanzadas de manera ordenada que tienen como objetivo encontrar un elemento determinado dentro de un conjunto. Aplicado en este proyecto sería un código que es capaz de encontrar la carretera que el agente necesita después de recorrer otras carreteras. Existen miles de problemas parecidos a los que esta herramienta se enfrenta [5], concretamente, los grafos. Un grafo es un diagrama de puntos que están relacionados entre sí mediante líneas. Es usado para la resolución de distintos problemas, entre ellos de lógica, combinatoria...
Por lo tanto, el principal objetivo de la herramienta, de manera muy resumida, será ofrecer una solución algorítmica a un problema de grafos. Existen infinidad de algoritmos para la solución que buscamos desde seleccionar puntos del grafo de manera aleatoria hasta encontrar la solución correcta (que no tiene porqué ser la óptima), hasta los pensados de manera totalmente específica para este tipo de situaciones. Esta es una lista de algunos algoritmos que se podrían considerar para la resolución al problema de nuestras carreteras:

- Algoritmo Depth-First Search
- Algoritmo Breadth-First Search
- Algoritmo de Dijkstra
- Algoritmo A*

Se han elegido los algoritmos más populares y adecuados al problema actual [6], y se han analizado y comparado uno a uno para seleccionar el óptimo para el presente proyecto.
Sobre ellos comentar lo siguiente: los cuatro algoritmos no están enfocados a lo mismo, pero realizando distintas variaciones sobre ellos podría conseguirse el resultado esperado. Es decir, el de Dijkstra por ejemplo, busca encontrar el camino más corto en un grafo con pesos, mientras que el Depth-First Search busca encontrar el primero y sin pesaje.
De aquí se puede sacar algo más, en la navegación de un punto del mapa al otro, el objetivo siempre es llegar, no importa el cómo, pero un factor relevante es el tiempo invertido a la hora

de hacerlo. Así pues, se podría llegar a considerar la adición de pesos en las carreteras para llegar a un punto determinado.

3.2.1 Algoritmo Depth-First Search

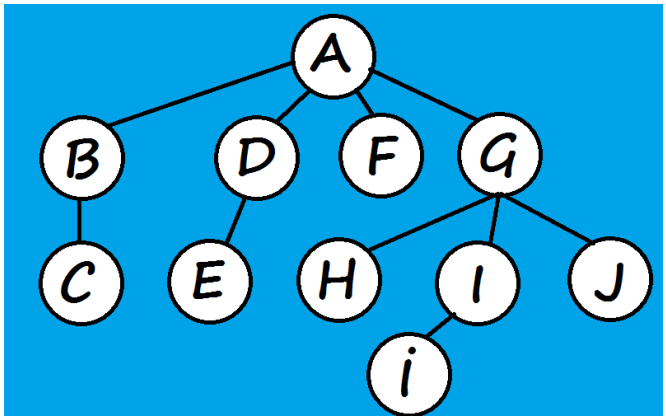


Figura 3.1: Ejemplo de grafo. Fuente: Wikimedia

El algoritmo Depth-First Search (DFS), analiza cada uno de los nodos hasta encontrar el primero que cumpla los requisitos determinados. Este algoritmo recorrerá el primer nodo en profundidad siempre, es decir el que está más a la izquierda, si este tiene un hijo, lo recorrerá después siguiendo el mismo patrón, primero el que encuentre en profundidad. Suponiendo un ejemplo sobre la figura 3.1 en el que se quiera encontrar la D, por ejemplo. El recorrido del algoritmo sería el siguiente: A -> B -> C -> D.

3.2.2 Algoritmo Breadth-First Search

El algoritmo Breadth-First Search o (BFS), tambien analiza todos los nodos hasta encontrar el primero que cumpla las condiciones requeridas. Éste también empieza por el primero que encuentra a la izquierda, pero la diferencia que tiene con el algoritmo DFS, es que primero debera recorrer todos los nodos en anchura antes que en profundidad. Esto supone que hasta que no analice todos los nodos disponibles en determinado nivel, no comenzará con el siguiente. Así pues, repitiendo el ejemplo del grafo (figura 3.1), en el que se busca la C, el recorrido del algoritmo sería el siguiente: A -> B -> C.

3.2.3 Algoritmo de Dijkstra

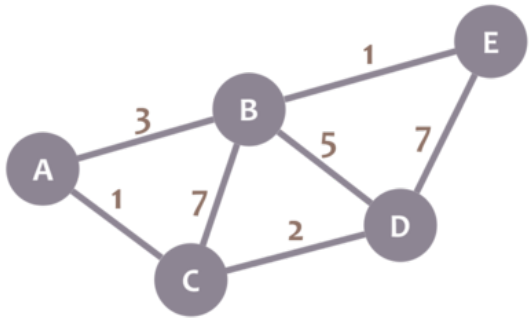


Figura 3.2: Ejemplo de grafo. Fuente: GeeksforGeeks

El algortmo de Dijkstra surge como una solución para encontrar el camino mínimo entre dos puntos. Así que realizará diversas comprobaciones para comprobar que sea el correcto. Véase la figura 3.2.
Este algoritmo se basa en que para ir de un punto a otro, no tenemos porqué tomar siempre el mismo camino, si no que existen diversas alternativas a tomar y obviamente, no va a llevar el mismo tiempo realizar un recorrido de 1km que uno de 100km, suponiendo obviamente la misma velocidad. Para ello hay que comprobar las diversas alternativas que existen y tomar la mejor. En el caso de que se pueda llegar a un mismo nodo, por diversos caminos, se eliminará aquel que en su recorrido tenga una mayor distancia. Supongamos entonces un ejemplo partiendo desde el Nodo 0, hasta llegar al nodo 4.

- Camino 1: 0 -> 1 (4) -> 2(12)
- Camino 2: 0 -> 7 (8) -> 6 (9) -> 5 (11)
- Camino 3: 0 -> 1 (4) -> 2 (12) -> 8 (14)
- Camino 4: 0 -> 1 (4) -> 2 (12) -> 3 (19)
- Camino 2: 0 -> 7 (8) -> 6 (9) -> 5 (11) -> 4 (21)

Así pues, el algoritmo se va encargando de mantener en memoria cuanto pesa el recorrido total del grafo y va descartando aquellos cuyo peso supera el mínimo hasta cierto nodo. En un grafo pequeño se puede plantear con buenos resultados, pero a la hora de escalarlos al proyecto, el tiempo computacional se disparará.

3.2.4 Algoritmo de búsqueda A*

Este algoritmo, aparece también como solución a encontrar el camino más corto entre dos nodos. Para ello recurrirá a tres variables, en este caso, g, h y f, que variarán en función de la posición de cada nodo. En donde 'g' es la distancia al nodo inicial, 'h' al nodo final y 'f' el valor de la suma de ambos. Así pues la principal diferencia con Dijkstra es que el valor de éste viene dado por la expansión desde el nodo raiz mientras que en A* el valor vendrá dado por las funciones anteriormente definidas. Cabe decir, que el algoritmo A* tiene un hueco especial en los videojuegos ya que a nivel de utilización de nodos, tiene un tiempo de respuesta muchísimo más bajo que el algoritmo de Dijkstra ya que a con un número grande de nodos, éste comienza a dispararse.

3.2.5 Elección del algoritmo

Después de haber valorado los distintos algoritmos para ERS, se ha decidido que se utilizará el algoritmo A* usando cada carretera como un nodo del algoritmo. De este modo también se podría añadir una heurística personalizada basada en la velocidad máxima permitida de la carretera, así se daría prioridad a las autopistas antes que a las vías convencionales por ejemplo.

Capítulo 4

Planificación

En este capítulo se describirá la planificación a la que ha sido sometida el proyecto, así como un análisis de la misma.

4.1 Toma de requisitos

En base a los objetivos establecidos en la sección 1.3, se procede a establecerla toma de requisitos tanto funcionales como no funcionales:

4.1.1 Requisitos funcionales

- **RF1**: La librería debe de incluir una interfaz gráfica propia.
- **RF2**: La interfaz gráfica debe de ser sencilla.
- **RF3**: La interfaz dispondrá de dos colores para indicar si se puede colocar el elemento.
- **RF4**: Deben existir señales que regulen el tráfico.
- **RF5**: Los vehículos deben personalizarse facilmente.
- **RF6**: Los vehículos deben respetar dichas señales
- **RF7**: Los vehículos actuarán independientemente del usuario.
- **RF8**: Los vehículos regularán su velocidad en función de la situación.
- **RF9**: Los semáforos alternarán sus colores entre el rojo, amarillo y verde.

4.1.2 Requisitos no funcionales

- **RNF1**: El desarrollador debe ser capaz de utilizar la librería solamente con el manual del usuario. Ha de ser sencilla.
- **RNF2**: Un usuario debe ser capaz de utilizar Easy Road Systems en menos de una hora.
- **RNF3**: La interfaz gráfica debe ser de fácil uso.
- **RNF4**: La librería debe disponer de algún método de ayuda en línea.

4.2 Planificación

4.2.1 Planificación inicial

En primer lugar, después de haber establecido un ciclo de vida para el proyecto, se ha determinado una planificación inicial conteniendo los pasos que el proyecto seguiría. Para ello se han tenido en cuenta diversos factores:

- Solamente se utilizará una persona (un desarrollador y gestor al mismo tiempo).
- Se utilizará un calendario estándar pero sin festivos, es decir, se trabajará de lunes a viernes.
- La jornada laboral durará 8 horas.
- Se establecerán diversos hitos, que serán una tarea a modo de control del estado del proyecto en cada una de sus fases.
- El proyecto se dividirá principalmente en 3 etapas: Introducción, Desarrollo y Conclusiones.
- Durante el desarrollo, cada iteración requerirá un mayor tiempo de desarrollo.
- Cada una de las tareas incluyen la documentación del proyecto. Durante las iteraciones del ciclo de vida, se tomará en cuenta lo documentación como una tarea a parte.

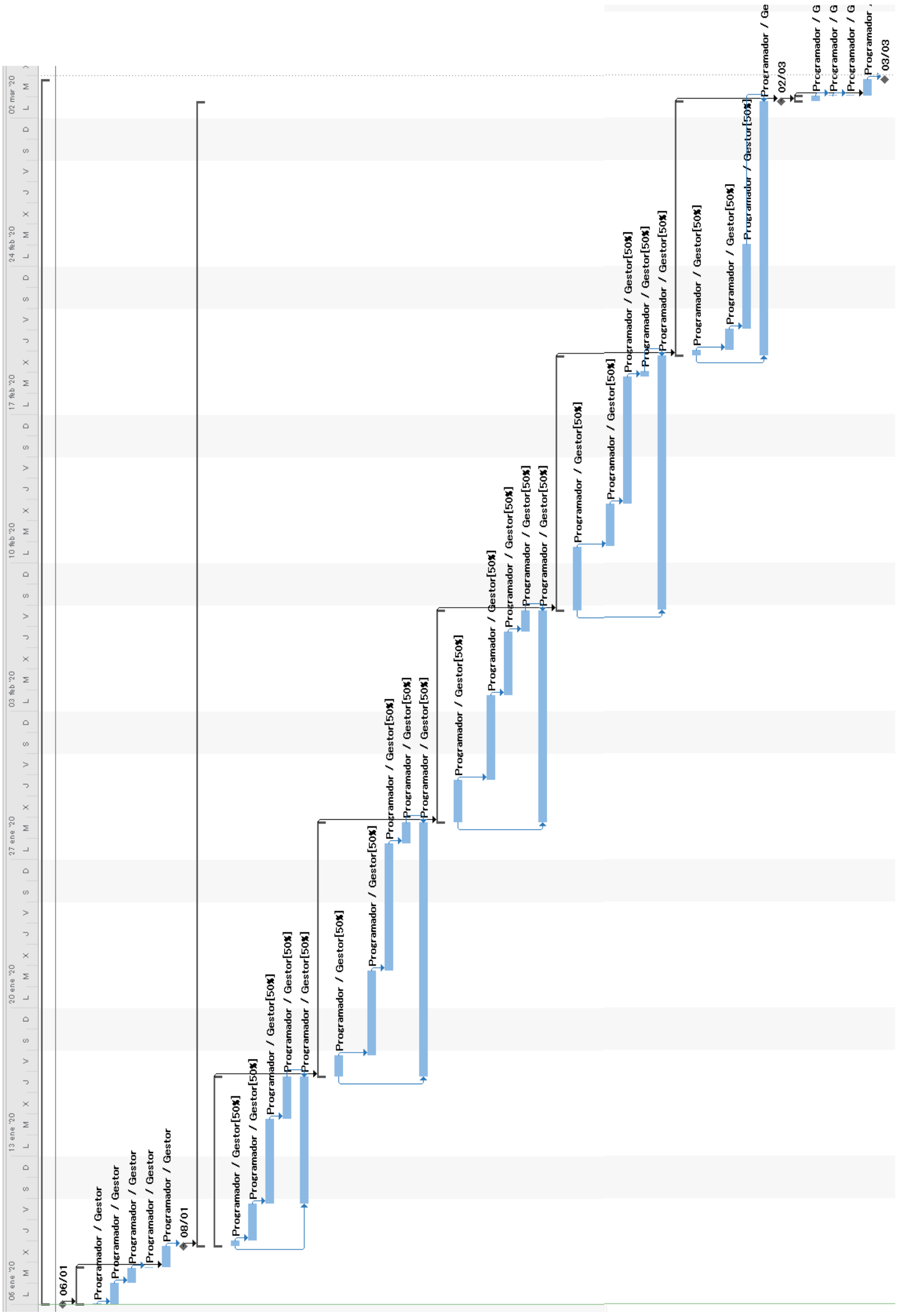


Figura 4.1: Diagrama de Gantt inicial

Al tener en cuenta lo descrito más arriba, se realizó una estimación del proyecto de aproximadamente unas 308 horas. Las cuales se repartían mayoritariamente durante la fase de desarrollo, en el transcurso de las iteraciones.
Por lo tanto, el desarrollo sería la fase clave del proyecto, mientras que la introducción y las conclusiones se volverían una parte menor en cuanto al tiempo empleado.

4.2.2 Resultados finales

Pese a haber realizado una valoración inicial de los tiempos implicados en cada una de las tareas del proyecto, la primera aproximación resultó ser optimista. Los tiempos necesarios para las tareas situadas fuera de la etapa de desarrollo, fueron bastante acertados, sim embargo, aquellos situados en esta etapa no.

En casi todas las iteraciones se produjo el mismo error, el tiempo dedicado a la documentación ocuparía el mayor tiempo del desarrollador. Finalmente, ésto no fue así, lo que ocuó más tiempo fue la etapa de desarrollo de cada iteración.

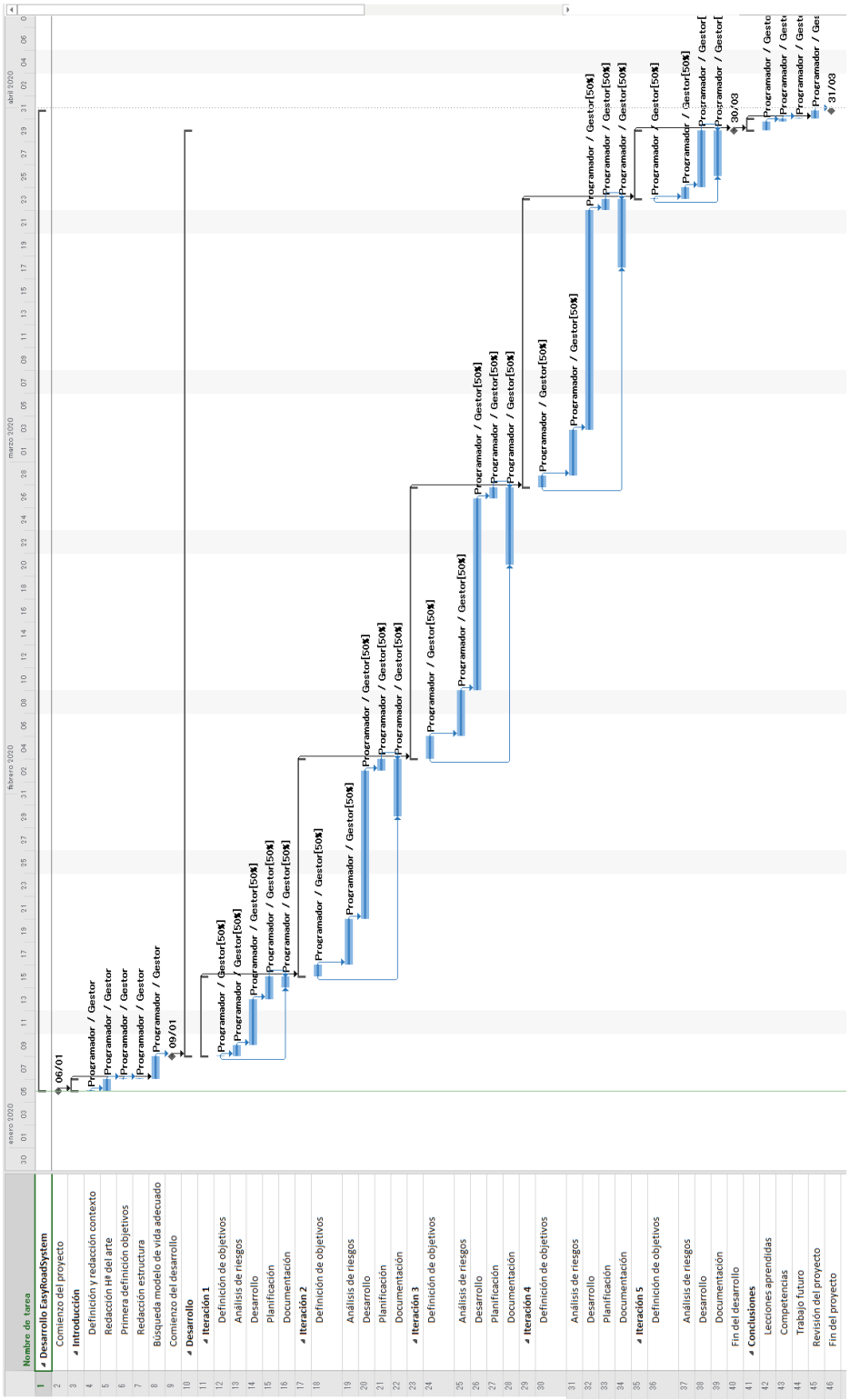


Figura 4.2: Diagrama de Gantt final

Como se ha mencionado anteriormente, esta situación no se vivió en todas las iteraciones:

- **Iteración 1**: las situaciones predichas fueron casi idénticas a las reales. Habiendo una variación mínima en cuanto a horas requeridas.

- **Iteración 2**: se produjo una variación de cuatro horas, el desarrollo tardó más de lo previsto y la documentación se acabó en menos tiempo del establecido.

- **Iteración 3**: se produjo una variación de cincuenta horas aproximadamente, el desarrollo se alargó mucho más de lo previsto y la documentación se predijo con demasiada holgura, siendo necesarias muchas menos horas de las previstas.

- **Iteración 4**: casi idéntica a la iteración 3.

- **Iteración 5**: acabó antes de lo previsto, el desarrollo sufrió un retraso de ocho horas pero la documentación sufrió un adelanto de 16 horas.

Debido a todo lo mencionado en esta sección, el tiempo invertido total ha sido de aproximadamente de 321 horas.

4.3 Evaluación de costes

El único recurso asociado a este proyecto, es una persona polivalente (denominada Programador / Gestor en los diagramas de Gantt), la cual tiene una tarifa de 30€/hora.

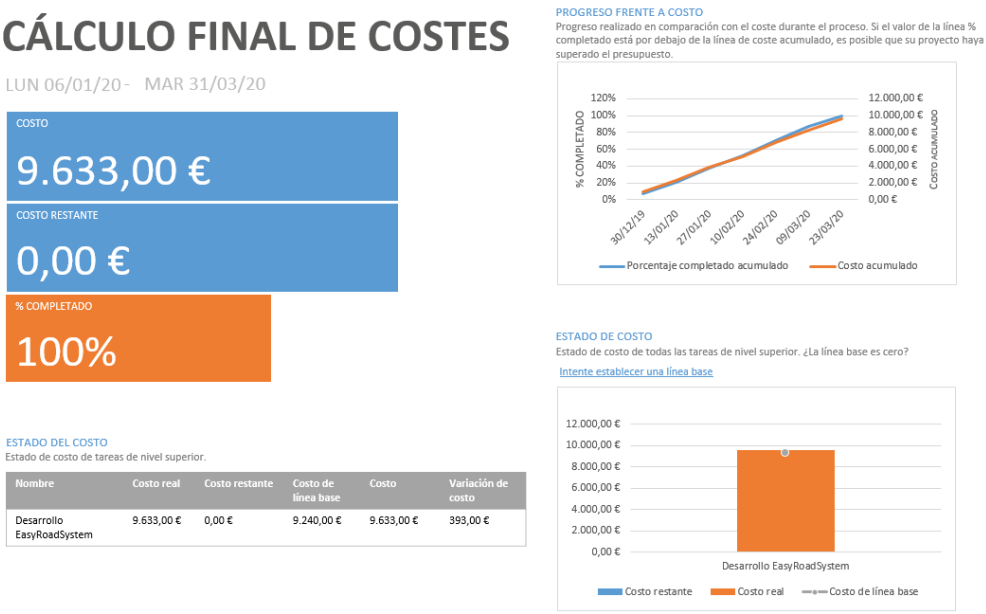


Figura 4.3: Cálculo de costes para el desarrollo de EasyRoadSystem

Así pues, como se observa en la figura 4.3, la aplicación ha requerido 9633€ para su desarrollo, existiendo un sobrecosto de 393€ de lo calculado en un principio. Esto fue debido a la falta de experiencia del propio gestor al calcular el tiempo necesario para algunas de las tareas.

Capítulo 5

Desarrollo

En este capítulo se encuentran los diversos objetivos alcanzados en cada una de las fases del ciclo de vida del proyecto, un análisis de todas las decisiones tomadas, las estructuras utilizadas y diversas explicaciones de los elementos que han sido necesarios.

5.1 Iteración 1

Primera iteración de las cinco totales del proyecto. Ésta será un poco distinta a las demás ya que es la inicial y no contiene una planificación anterior como ocurre con las siguientes. Será la iteración más difícil en cuanto a objetivos se refiere, pues dependiendo de estos definiremos la estructura a seguir durante el transcurso de todo el proyecto.

5.1.1 Objetivos

- Adquisición conceptos clave sobre el entorno
- Definición posibles clases a desarrollar
- Estructuración del punto anterior mediante OOP

5.1.2 Análisis de riesgos

La implementación del core del proyecto es, en todos los casos, una decisión de grandísimo impacto. Los errores en esta fase pueden provocar un gran sobrecoste en el proyecto. Por lo tanto, es necesario tratar con especial cuidado cualquiera de las decisiones tomadas durante el transcurso de esta iteración.

5.1.3 Desarrollo

Debido a todo lo mencionado en el anterior punto, se ha procedido con cautela a la hora de la primera investigación de los objetivos de esta iteración.

En cuanto a la adquisión de conceptos claves sobre el entorno:

Lo primero que se ha tenido en cuenta, son una serie de elementos clave que son propios de el ámbito del proyecto. Esto se ha traducido en la siguiente lista de conceptos clave de circulación:

- **Vehículo / agente:** Cada uno de los elementos que se desplazan sobre la carretera.
- **Carretera:** Se trata de la estructura mínima sobre la que los agentes deben desplazarse. Constituye la unidad mínima de circulación.
- **Código de circulación:** Reglamento que establece un control sobre cómo debe circular cada uno de los agentes sobre la carretera.

Estos conceptos se encuentran fuertemente vinculados entre sí, ya que un vehículo dependerá de una carretera para su correcto funcionamiento y del mismo modo, su circulación se rigirá según las normas estipuladas en el código de circulación [7]. Esto es debido a que en el mundo real, no se permite que los vehículos circulen a su libre albedrío por las carreteras. Easy Road System busca la mayor similitud entre ambos mundos, así que es lógico que sus agentes también deban respetar unas normas.

5.1.4 Planificación

En este apartado se decidió intentar desarrollar una clase padre para cada uno de los dos elementos principales nombrados anteriomente; el vehículo y la carretera.
De este modo, será sencillo crear una variante en funcionamiento de cada uno de ellos si así se requiere en un futuro (se pueden ver las clases creadas en los diagramas UML referidos más abajo, figura 5.2 y figura 5.3).
Teniendo en cuenta lo comentado en el párrafo anterior, se necesita una estructura sencilla que permita que los vehículos circulen sobre las carreteras. Así pues, hará falta una clase Road que será la padre para desarrollar otro tipo de clases, una clase Vehicle, que haga de padre para cualquier tipo de vehículo y algún método para lograr un movimiento fluído de los agentes. Se hablará más adelante de la elección de un algoritmo para sufragar este problema.

5.2 Iteración 2

En la segunda iteración, se llevará a cabo la implementación de las pautas marcadas en la Iteración 1. Se intentará lograr que todos los objetivos se cumplan de manera exitosa.

5.2.1 Objetivos

- Implementación estructura básica de carretera.
- Movimiento básico de los agentes.
- Implementación de estructura de control de carreteras.

5.2.2 Análisis de requisitos

- Las carreteras se han de colocar dentro de la escena.
- Los agentes deben de ser capaces de reconocer las carreteras.
- Los agentes deben de poder moverse por encima de las mismas.
- Debe existir un elemento, que se encargue de controlar el buen funcionamiento de agentes y carreteras.

5.2.3 Análisis de riesgos

Una vez aclarados los objetivos de esta iteración, se ha realizado una comprobación de los riesgos que puede generar y posibles alternativas para éstos.
En cuanto a la estructura básica de la carretera, tal y como se ha planteado, hará posible el movimiento de los vehículos sobre ella. También sera importante permitir la especialización de esta clase para alternar la colocación de las distintas variantes que puedan existir. Gracias a esto, se podrá variar el comportamiento de los agentes según se encuentren en un tipo de carretera u otro.

En cuanto a los vehículos, habrá que identificar el modo de desplazamiento correcto para simular la manera en la que se mueven en la vida real, mediante el uso de fuerzas (como son las aceleraciones y las fricciones a las que se ven sometidos) u otra alternativa como puede ser un desplazamiento de su posición sin que las leyes de la física actúen sobre ellos. Ya que el proyecto busca el mayor símil con una condución realista, la decisión tomada es que éstos se desplacen mediante fuerzas. Esto sin duda llevará a complicaciones más adelante y que el proyecto lleve más tiempo del esperado, pero se ha considerado como algo sumamente importante para el usuario final. Ésto es debido a que basar el movimiento de los agentes en la

actuación de fuerzas, implica un mayor número de cálculos que no serían necesario en caso de escoger la otra alternativa.

5.2.4 Análisis

En esta primera etapa de desarrollo de la herramienta, se marcará como objetivo cumplir con los requisitos funcionales RF7 y RF8 [4.1.1]. Para ello, se creará un sistema que se encargue de mover los agentes sin que el usuario actúe y se utilizarán diversas medidas (como la comprobación de las distancias entre vehículos) como primer medio de control de su velocidad.
También se afrontará indirectamente el requisito no funcional [4.1.2 RNF2 intentando simplificar lo máximo posible las clases utilizadas.

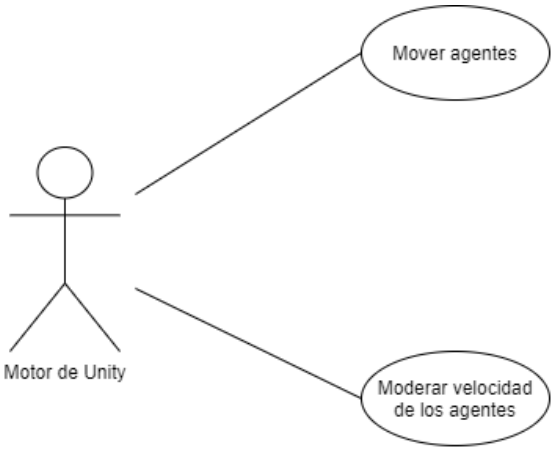


Figura 5.1: Diagrama de casos de uso de la iteración 2

5.2.5 Desarrollo

Se comienza la implementación de ambas estructuras, las carreteras irán controladas por una clase Road, mientras que los vehículos por una clase Vehicle. Del mismo modo, se ha creado una clase RoadManager, que se encargará de gestionar las carreteras que debe de cruzar cada uno de los vehículos para alcanzar su meta.
Por lo tanto en

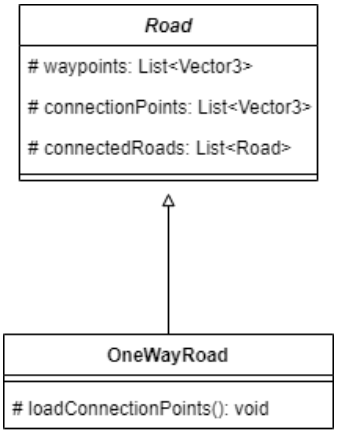


Figura 5.2: Primera versión clase Road

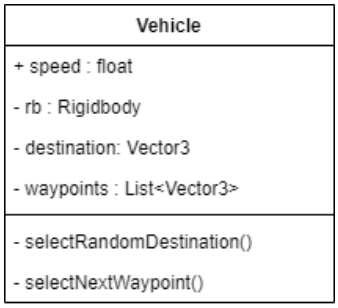


Figura 5.3: Primera versión clase Vehicle

En la clase Road (figura 5.2), podemos observar la creación de una estructura que contiene los puntos de control por los que debe de pasar el vehículo para atravesarla. Esta clase, se colocará sobre un objeto (*GameObject*) del motor gráfico Unity3d actuando como si fuese una característica más del mismo. Se situa sobre un componente ya mencionado anteriormente, el *script*. Así pues tendremos un plano cuadrado (con posibilidad de deformarse) que representará una instancia de Road. En cuanto a la clase, se ha diseñado con una lista de waypoints, que no son más que puntos en un plano de 3 dimensiones por los cuales el vehículo debe de pasar para continuar su marcha. También posee connectionPoints, los cuales serán los vectores de posición de entrada de un vehículo en el objeto. Por último, el objeto posee una lista de las carreteras que son adyacentes al mismo. Para identificar una carretera como adyacente, cuando el objeto se carga, manda una señal a otra clase, llamada RoadManager.
El método loadConnectionPoints() se encargará de identificar cada uno de los puntos por los cuales una carretera se puede conectar con otra.

En cuanto a la figura 5.3 se observa que contiene varios atributos. Como la velocidad con la que el vehículo se desplazará de un punto a otro, un objeto Rigidbody, un vector de

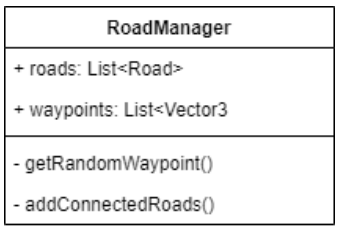


Figura 5.4: Primera versión clase RoadManager

posición que es el *waypoint* hacia el que se dirige, y una lista de *waypoints* que son las carreteras por las que tiene que pasar para alcanzar su posición final. En cuanto a los métodos, selectRandomDestination() hará una llamada al RoadManager, que se encargará de elegir una carretera al azar y devolver la posición hacia la que se tiene que dirigir el vehículo. El otro método, selectNextWaypoint(), comprobará si el vehículo ha llegado al proximo waypoint, lo eliminará de la lista y marcará como destino el próximo.

Por último, se ha implementado la clase RoadManager (figura 5.4). Esta clase tiene el objetivo de ser el nexo de unión entre las carreteras y los vehículos. Cuando la clase se carga en memoria, obtiene una lista de todas las clases Road que hay en la escena y de ella carga todos los waypoints que contienen para utilizarlos posteriormente. Esta clase posee dos métodos:

- **getRandomWaypoint():** el cual eligirá un punto al azar de la lista de carreteras y se lo devolverá al vehículo que se lo haya pedido.
- **addConnectedRoads():** una vez que la clase carga todas las carreteras en memoria, realiza una comprobación de aquellas que tienen los puntos de conexión próximos entre sí, marcándolos como adyacentes a la carretera colindante. Esto se va notificando a cada una de las carreteras para que tengan una lista de aquellas a las que están conectadas [5.1].

```
1  public void addConnectedRoads(){
2          foreach (var road in
       GameObject.FindGameObjectsWithTag("Road")) {
3              roads.Add(road.GetComponent<Road>());
4          }
5          foreach (var r1 in roads) {
6              foreach (var r2 in roads) {
7                  if (r1 == r2)
8                      continue;
9                  foreach (var cp1 in r1.getConnectionPoints()) {
10                     foreach (var cp2 in r2.getConnectionPoints()) {
11                         if (Vector3.Distance(cp1, cp2) <= 3) {
```

```
12                                  r1.addConnectedRoad(r2);
13                              }
14                          }
15                      }
16                  }
17              }
18          }
```

Listing 5.1: Método encargado de conectar las carreteras entre sí.

El umbral de distancia entre ambos puntos (en este caso 3), determinará la distancia máxima a la que se tienen que encontrar esos puntos para que estén conectadas. En esta iteración no ha tenido uso, pero si que será interesante más adelante, a la hora de definir un algoritmo que indique el camino a tomar.

Al utilizar un elemento mínimo como base, se permite que varias carreteras se enlacen unas con otras de manera sencilla, teniendo siempre en cuenta la dirección del punto de salida básico (verde), que concuerda con el eje X positivo del objeto. Se puede observar un claro ejemplo de lo que se acaba de comentar más adelante (figura 5.7).

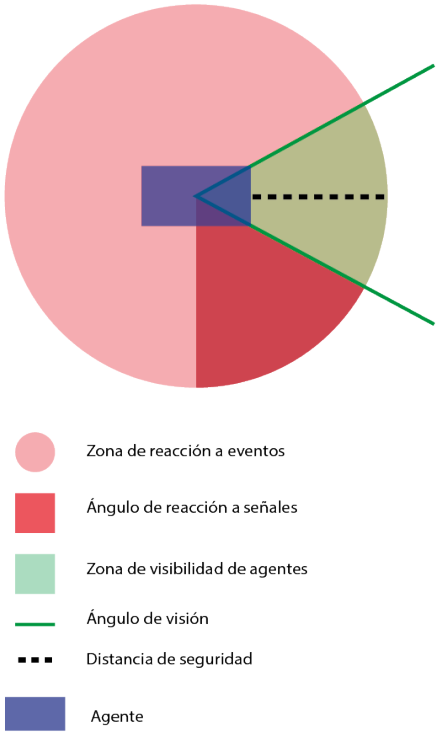


Figura 5.5: Esquema básico de una entidad vehículo

5.2.6 Pruebas

Tests de la iteración 2		
Nombre del test	Resultado esperado	Resultado recibido
Carreteras visibles en la escena	S	S
Los vehículos se sitúan sobre las carreteras	S	S
Los vehículos pueden desplazarse	S	S
Los vehículos siguen perfectamente las carreteras	N	N
Existe una estructura que controle las carreteras	S	S
Los vehículos reciben el camino a recorrer	N	N

Tabla 5.1: Tabla que contiene la descripción de los test realizados durante la iteración 2.

5.2.7 Planificación

En este sprint se han cumplido todos los objetivos que se pedían, sin embargo, durante la implementación de los mismos, se detectan las siguientes ventanas de mejora:

- Los agentes se mueven de una carretera a otra, pero no siguen un orden determinado, es decir, el destino de cada uno de los vehículos se alcanzará en la primera iteración del algoritmo. Es necesario la implementación de un algoritmo que se encargue de gestionar ésto, este problema se solventará en el próximo sprint mediante la adición de un método para alcanzar una posición siguiendo una ruta.
- Según el código actual del método addConnectionRoads() [5.1], los agentes podrán colisionar unos con otros a la hora de tener un camino fijado, ya que estas carreteras no vienen marcadas por una dirección.

5.3 Iteración 3

Se da comienzo a la siguiente iteración. En ella, se definen nuevos objetivos que intentan resolver los problemas de navegacion de agentes que se detectaron con anterioridad.

5.3.1 Objetivos

- Mejora de la estructura básica de Road.
- Implementación y reconocimiento de otros agentes.
- Elección e implementación del algoritmo de movimiento de los agentes.
- Recopilación de información sobre las señales que los agentes pueden reconocer.

5.3.2 Análisis de requisitos

- Los agentes deben de ser capaces de reconocer a otros agentes y detenerse cuando estén proximos entre sí.
- Los agentes deben de seguir las carreteras gracias a la clase RoadManager.
- Se debe proveer un algoritmo que marque el recorrido de carreteras a recorrer por los agentes.
- Establecer una o más señales que un agente pueda reconocer.

5.3.3 Análisis de riesgos

Esta iteración será la encargada de realizar mejoras en la Clase Road y en la clase Vehicle, aunque también se buscará un algoritmo óptimo de búsqueda y alguna forma de realizar el reconocimiento de señales de tráfico.
Los tres primeros objetivos están muy vinculados unos con otros, pues para la utilización de ciertos algoritmos de navegación, puede que sea necesario añadir algún parámetro a estas clases. Por lo tanto el riesgo que supone que realizar uno de los dos primeros objetivos, antes de haber investigado el tercero, es inútil, así pues el objetivo con mayor prioridad pasará a ser la elección del algoritmo.

5.3.4 Análisis

Esta iteración afrontará una aproximación hacia el requisito funcional [4.1.1] RF4 mediante la creación de la primera señal vial.

Por otra parte se deberá mantener el código de las clases de la librería simple para seguir cumpliendo el RNF2 [4.1.2].

5.3.5 Desarrollo

5.3.5.1 Mejora de la clase road

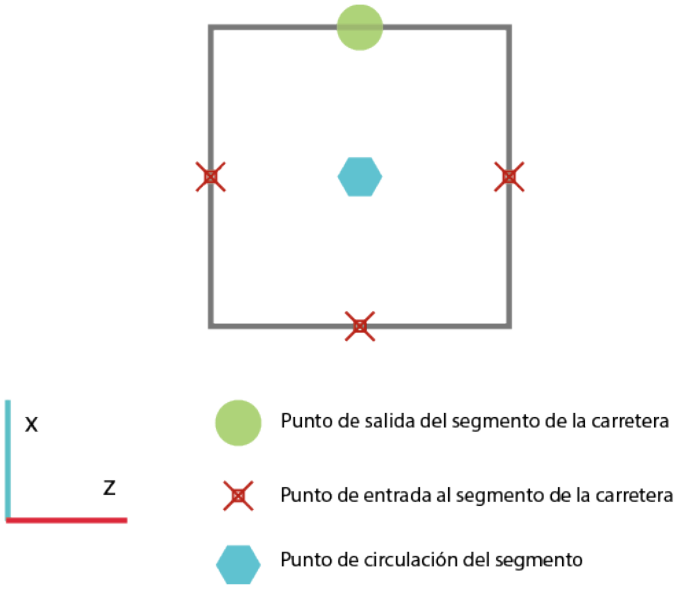


Figura 5.6: Entidad carretera

En la imagen 5.6, se muestra el esquema básico utilizado al instanciar una carretera. La librería utiliza un elemento (*prefab*) con forma rectángular. En él se utiliza el punto central de la figura (hexágono azul) como un waypoint, por el cual los agentes deben de circular. Las marcas en cruz indican los lugares por los cuales una entidad puede acceder a la carretera, mientras que el punto verde indica el punto por el cual el vehículo debe abandonar la carretera. De este modo, un vehículo accederá a la carretera por una de las cruces rojas, alcanzará el waypoint central y abandonará la carretera por el punto verde.
Al haber aplicado puntos de entrada y salida de la carretera, se crearán facilmente variaciones con los puntos de salida en otra posición para así poder variar la dirección del vehículo.

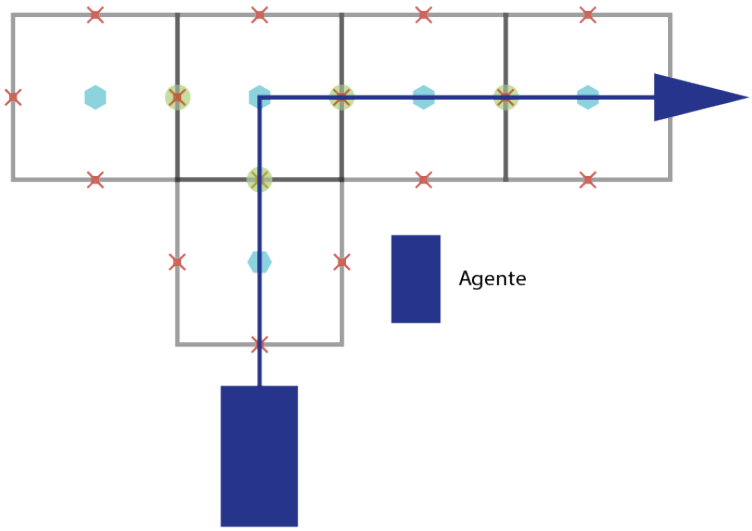


Figura 5.7: Esquema básico de circulación

En la imagen 5.7, observamos claramente este comportamiento con el ejemplo de un vehículo que llega desde la parte más baja de la representación. El vehículo comienza su ruta entrando por uno de los puntos de entrada de la carretera, en este caso el situado en el eje X negativo, posteriormente alcanzará el waypoint situado en el medio de la carretera y saldrá por el punto de abandono de la carretera. Llegados a este punto, volveremos a entrar en un bucle hasta la proxima instancia de la carretera. El vehículo se colocará en el waypoint azul del próximo prefab y abandonará éste por el punto de salida. Así hasta alcanzar su destino.

5.3.5.2 Elección e implementación del algoritmo de movimiento de los agentes

Tras haber elegido al algortimo A* como candidato final para la elección de un camino para los vehículos, se lleva a cabo la implementación de éste [5.2].

```
1 public List<Road> calculateTrip(Vector3 actualPos, Vector3
      destinationPos) {
2     Road actualRoad = getClosestRoadByPosition(actualPos);
3     Road destinationRoad = getClosestRoadByPosition(destinationPos);
4     List<Road> finalPath = new List<Road>();
5     List<RoadCost> openRoads = new List<RoadCost>();
6     List<RoadCost> closedRoads = new List<RoadCost>();
```

```
7       RoadCost startingRoadCost = new RoadCost(actualRoad,
        actualRoad, destinationRoad);
8       openRoads.Add(startingRoadCost);
9
10      while (openRoads.Count > 0) {
11          openRoads = openRoads.OrderBy(x => x.fCost).ToList();
12          RoadCost currentRoad = openRoads[0];
13          if (Vector3.Distance(currentRoad.road.transform.position,
        destinationRoad.transform.position) < 0.1) {
14              return finalPath =
        RoadsCostToRoads(getFinalPath(startingRoadCost, currentRoad,
        closedRoads));
15          }
16          openRoads.Remove(currentRoad);
17          closedRoads.Add(currentRoad);
18          var connectedRoads =
        RoadsToRoadsCost(currentRoad.road.getConnectedRoads(),
        actualRoad, destinationRoad);
19
20          foreach(var cr in connectedRoads) {
21              if (closedRoads.Contains(cr)) continue;
22              cr.parentRoad = currentRoad;
23
24              if (!openRoads.Contains(cr)){
25                  openRoads.Add(cr);
26              }
27          }
28      }
29
30      actualRoad.gameObject.GetComponent<Renderer>().material.color =
        Color.green;
31
        destinationRoad.gameObject.GetComponent<Renderer>().material.color
        = Color.red;
32      return finalPath;
33  }
```

Listing 5.2: Método de calcular el trayecto a realizar por cada vehículo.

- Se busca cual es la carretera que se haya más proxima al agente y aquella a la que debe ir.
- Se inicializan variables auxiliares (listas de carreteras)
- Se utiliza una estructura auxiliar de carretera que contiene la propia carretera junto con variables correspondientes para el cálculo de costes.

- Se coloca la carretera en la que se encuentra el agente en la lista de carreteras abiertas y se lanza un bucle en el cual se van comparando los valores de 'f' de las otras carreteras conectadas a ésta, ordenándose previmente por su mismo coste.
- Se elimina esta carretera de la lista de carreteras abiertas y se añade a la de cerradas.
- Se comprueba cada una de las carreteras colindantes a la que se acaba de cerrar y se añaden a la lista de carreteras abiertas.
- Se repite el proceso hasta alcanzar la carretera de destino.

```
List<RoadCost> getFinalPath(RoadCost startingRoad, RoadCost
    destinationRoad, List<RoadCost> closedRoads) {
    List<RoadCost> trip = new List<RoadCost>();
    RoadCost currentRoad = destinationRoad;
    while (currentRoad != startingRoad) {
        trip.Add(currentRoad);
        currentRoad = currentRoad.parentRoad;
    }
    trip.Reverse();
    return trip;
}
```

Listing 5.3: Método encargado de devolver el camino final a recorrer por el vehículo.

Una vez finalizado el proceso descrito anteriormente [5.3], se recorre el camino inverso y se van almacenando las carreteras recorridas en una lista. Cuando este proceso acaba, se invierte la lista y se devuelve al agente para que pueda recorrerla.

5.3.5.3 Recopilación de información sobre las señales que los agentes pueden reconocer

En el código vial existen muchísimas señales las cuales no tienen la misma prioridad una que otra. Por ejemplo, a estas alturas del desarrollo, sería totalmente ilógico utilizar una señal de prohibido circular los vehículos con una masa superior a X, antes de realizar un semáforo. Este dispositivo mencionado, se encargará de gestionar la circulación entre las carreteras, dando prioridad de paso a las carreteras que él crea oportuno.
Así pues lo más lógico para el proyecto es comenzar por crear una primera señal, que sirva como referencia para el resto. La decisión tomada es la de implementar más adelante un semáforo.

5.3.6 Pruebas

Tests de la iteración 3		
Nombre del test	Resultado esperado	Resultado recibido
La clase Road dispone de cuatro puntos de anclaje	S	S
La clase Road tiene un punto de entrada y varios de salida	S	S
Existe un waypont dentro de la clase Road	S	S
Los vehículos se desplazan utilizando fuerzas que se ven afectadas por la gravedad.	S	S
Los vehículos se dirigen a los waypoints objetivos que vienen determinados por cada clase Road	S	S
Los vehículos respetan la distancia de seguridad con el resto	S	S*
Se ha implementado el algoritmo A*	S	S
La clase RoadManager utiliza el algoritmo anterior para buscar una solución optima al recorrido a seguir por el vehículo.	S	S
La clase RoadManager detecta perfectamente todas las carreteras que están disponibles para usarse	N	S
El vehículo termina su recorrido en su waypoint final determinado por lo recibido anteriormente.	S	S

Tabla 5.2: Tabla que contiene la descripción de los test realizados durante la iteración 3.

*: En este caso se ha detectado un error en el test. Los agentes, por un error de variable, no respetan una distancia prudencial con otros vehículos. Esto se ha solucionado aumentando el umbral de distancia a la hora de detectar otros agentes.
Las pruebas realizadas en esta iteración incluyen también aquellas realizadas en la anterior.

5.3.7 Planificación

Después de haber terminado esta iteración, se ha conseguido que los agentes circulen de una carretera a otra sin niguna ayuda del usuario, pero su comportamiento aún dista mucho del idóneo.
Por otra parte, el algoritmo que detecta si una carretera se encuentra conectada a otra, no es lo suficientemente preciso en algunas ocasiones, será necesario la implementación de una estructura auxiliar que se encargue de solventar dicho problema.

5.4 Iteración 4

Penúltima de las iteraciones. En esta iteración se buscara facilitar, por parte de la herramienta, la creación de entornos para el usuario. También se busca que los agentes puedan identificar las señales de tráfico a las que se expongan y por último, conseguir un desplazamiento más natural de los vehículos involucrados en el proyecto.

5.4.1 Objetivos

- Diseño e implementación de la interfaz de usuario de la herramienta de creación de carreteras.
- Implementación del reconocimiento de señales por parte de los agentes.
- Mejora del movimiento de los agentes

5.4.2 Análisis de requisitos

- Se ha de crear una interfaz que simplifique la colocación de carreteras.
- Los agentes han de ser capaces de reconocer señales de tráfico.
- Los agentes deben de poder moverse de manera más natural a través de su recorrido.

5.4.3 Análisis de riesgos

El objetivo principal de esta iteración será facilitar la creación de las carreteras, principalmente, ya que actualmente el sistema no dispone de ningún sistema que se encargue del posicionamiento de cada una de las entidades, haciendo que sea un trabajo tedioso su colocación. interfiriendo así en el buen funcionamiento del sistema.
También es importante para la correcta inmersión del sistema, la implementación de una serie de señales que ordenarán a los agentes como deben funcionar. Así pues será importante la definición de semáforos, para comenzar, que regulen el tránsito, haciendo que los agentes adopten distintos estilos de conducción según el estado del semáforo (verde, amarillo, rojo).
Como punto final de esta iteración, se pretende mejorar el movimiento de los agentes. En el estado actual del proyecto, como se muestra en la figura 5.7, el funcionamiento del agente será ir desde un waypoint a otro hasta llegar a su destino. En cuanto éste alcance un cruce o una intersección, cambiará la dirección apuntando directamente hacia su próximo objetivo. Esto, de manera visual, presenta un cambio brusco en la dirección de los agentes, que es aceptable en los primeros momentos del desarrollo de la herramienta. El funcionamiento correcto sería una transición paulatina de la dirección del agente hasta finalmente apuntar a su próximo

objetivo.

5.4.4 Análisis

Esta etapa se enfocará en cumplir los requisitos funcionales restantes [4.1.1]. Para ello se creará la ventana de la herramienta Easy Road System, así como se buscará aumentar las interacciones entre los agentes o agente-semáforo.
Del mismo modo, se debe seguir manteniendo un código sencillo y comprenhensible. Así pues también se enfocará esta iteración en abordar el requisito no funcional [4.1.2] RNF3.

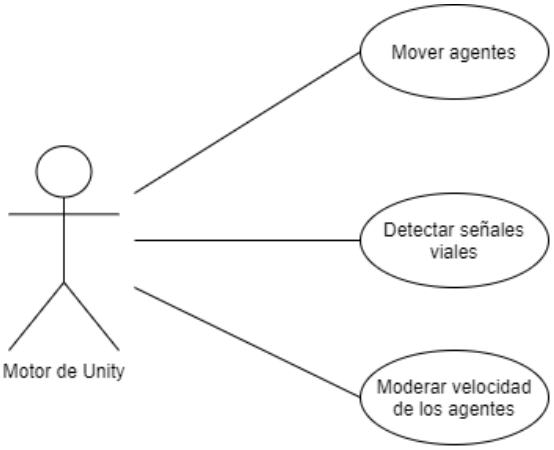


Figura 5.8: Diagrama de casos de uso de la iteración 4

5.4.5 Desarrollo

En cuanto al desarrollo de esta iteración, se abarcarán los objetivos comentados anteriormente. Para ello se ha valorado cada uno de ellos en cuanto a la ”cantidad de mejora” que aportará a la herramienta. Desde un punto de vista funcional, el proyecto funciona correctamente, pero no es intuitivo, así que la creación de una barra de herramientas que permita una mayor facilidad hacia el usuario a la hora de generar los entornos, será indispensable, tomando la máxima prioridad.
El reconocimiento de señales por parte de los agentes, también es uno de los objetivos que el proyecto aborda así pues, es el segundo a abordar. Se dejará así en último lugar la mejora de movimiento de los agentes, adaptándose a lo anteriormente creado.

5.4.5.1 Diseño e implementación de la interfaz de usuario de la herramienta de creación de carreteras

Uno de los objetivos principales de la herramienta es facilitar al usuario un sistema que situe de manera sencilla e intuitiva cada uno de los elementos necesarios para su uso. En anteriores iteraciones, ésto no se cumplía, volviéndose extremadamente tedioso la colocación de carreteras para un funcionamiento correcto, un error entre dos carreteras, supone el mal funcionamiento de la clase que se encarga de unir las carreteras entre sí.

Así pues, surgen diferentes aproximaciones de la ventana de ayuda de la aplicación. Es indispensable que la herramienta pueda identificar una carretera y ser capaz de colocar otra perfectamente pegada a ella. Esto se denomina como *snapping*. Gracias a él, se disminuirá de manera total, mientras el uso de la herramienta sea el adecuado, la probabilidad de fallos del sistema. Como la clase RoadManager, se encarga de identificar carreteras que se hayan próximas entre sí para su utilización, se debe minimizar el espacio que hay entre ellas para una buena detección, esta herramienta hará lo necesario para ello.

De este modo, existiría la opción de *snap*, que se encargará de situar las carreteras de manera inteligente y la opción manual que facilitará el posicionamiento de carreteras en el punto exacto que se desee. Principalmente orientada a la colocación de la primera carretera.

También la adición de un botón que se encargue de añadir elementos a la escena como la clase RoadManager (indispensable para el funcionamiento del sistema). Del mismo modo para una mayor sencillez a la hora de utilización, habrá un objeto que sirva como cursor (también llamado *preview*), así pudiendo ver la posición y rotación de la carretera usada.

Por lo tanto, para el desarrollo de la herramienta, se comenzará con el primer caso con el que el usuario se va a encontrar que será la escena vacía. El usuario tendrá que abrir el menú de ayuda de la librería, que se encuentra en la propia barra de herramientas de Unity con el título EasyRoadSystem. También se puede acceder a ella de manera alternativa pulsando la combinación de botones: Control + E

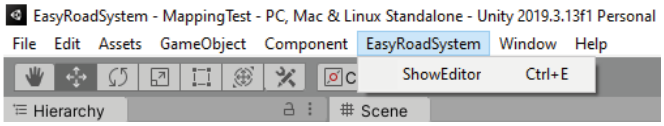


Figura 5.9: Barra de herramientas EasyRoadSystem

Unity3D contiene las herramientas necesarias para poder generar ventanas propias de la aplicación, facilitando tanto al usuario como al desarrollador el acceso a contenidos visuales de las herramientas desarrolladas. Todo esto se logra mediante la extensión de la clase de Unity, EditorWindow. En ella se sustituye el método OnGUI() para añadir los elementos que se quieran mostrar, vinculándolos a variables propias de la clase.

```
[...]
using UnityEditor;
public class RoadCreator : EditorWindow
{
  [...]

  void OnGUI() {
    [...]
      }
}
```

Listing 5.4: Ejemplo de la clase RoadCreator utilizando EditorWindow

Algo importante es el uso de UnityEditor como se puede ver en el código anterior [5.4]. Gracias a ello, se pueden acceder a los métodos necesarios como son GUILayout.Button() para crear botones o EditorGUILayout.Toggle() para crear checkboxes en caso de ser necesario. En la herramienta se hace uso de ambos métodos para la colocación de botones necesarios para la instanciación de el elemento de previsualización o para activar o desactivar el snapping.

```
void OnGUI() {
    EditorGUILayout.LabelField("EasyRoadSystem Brush");
    selectedPrefab = EditorGUILayout.ObjectField("Road Element",
    selectedPrefab, typeof(GameObject), false) as GameObject;
    if (selectedPrefab != null){
        prefabPath =
    PrefabUtility.GetPrefabAssetPathOfNearestInstanceRoot(selectedPrefab);
        if (gameObjectEditor == null)
            gameObjectEditor = Editor.CreateEditor(selectedPrefab);

        GUIStyle bgColor = new GUIStyle();
        bgColor.normal.background = EditorGUIUtility.whiteTexture;

    gameObjectEditor.OnInteractivePreviewGUI(GUILayoutUtility.GetRect(256,
    256), bgColor);
    }
    Repaint();

```

```
15      isSnapping = EditorGUILayout.Toggle("Snap to road", isSnapping);
16
17      if (GUILayout.Button("Set Preview (P)"))
18      {
19          setPreview();
20      }
21      if (GUILayout.Button("Set RoadManager"))
22      {
23          setRoadManager();
24      }
25      GUI.enabled = true;
26  }
```

Listing 5.5: Implementación del método OnGUI.

Figura 5.10: Ventana EasyRoadSystem

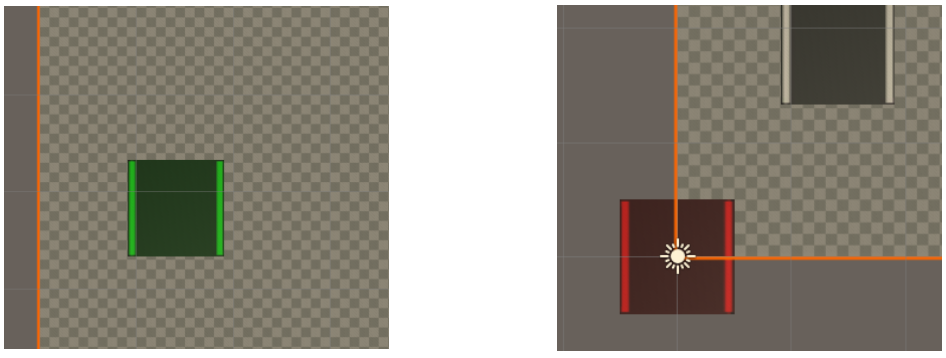
En la figura 5.10 se puede ver la ”traducción” del código descrito anteriormente en la ventana mostrada en Unity. Aquí se puede ver una primera versión de la ventana que el usuario utilizará como apoyo de su proyecto.

- **Road Element:** Se trata de un selector. Al hacer click sobre él, se abre otra ventana auxiliar en la cual se escoge el objeto a colocar dentro de la escena. Según qué tipo de objeto esté colocado en esta casilla, se visualizará en la ventana situada debajo del mismo. Al pulsar el botón "Set preview" o la tecla P, se colocará la previsualización dentro de la ventana del editor.

- **Snap to road:** Es una casilla checkbox. Esto quiero decir que puede estar seleccionada o no, en el código existe un booleano cuyo estado será true o false dependiendo del estado de esta elemento. Si está activado, el cual es por defecto, el objeto preview se pegará a una de las cuatro posiciones de la carretera cercana, pudiendo ser éstas, arriba, abajo, izquierda o derecha.

- **Set preview:** Es un botón. Al hacer click sobre este elemento, se eliminará el preview anterior en caso de que existiese. También comprobará que se haya elegido un objeto del Road Element y en caso de ser así, instanciará un nuevo preview en la escena con las características del elemento elegido anteriomente.

- **Set RoadManager:** Es un botón. Este botón tiene una simple función se encarga de que sólo pueda haber una instancia del RoadManager (añadida por medio de la ventana ERS). Por lo tanto en caso de que no haya ningún elemento RoadManager, el usuario puede pulsarlo, haciendo que el objeto aparezca en la escena. En caso de que exista el RoadManager, el botón se desactivará inhibiendo la posibilidad de que el usuario añada otro elemento igual, pues sólo es necesario uno.

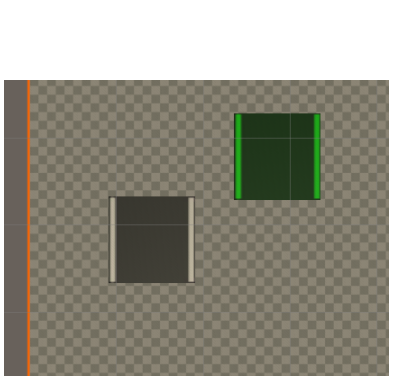


(a) Situación correcta (b) Situación incorrecta

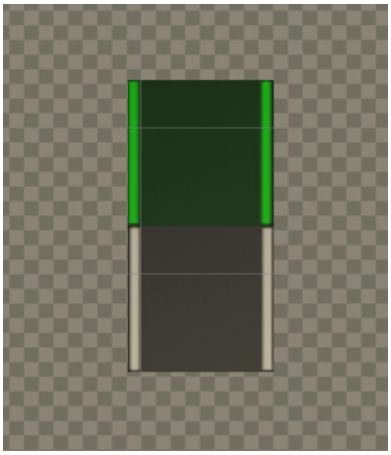
Figura 5.11: Distintos estados del preview según su colocación

En la figura siguiente, se observan dos imágenes, la 'a' y la 'b'.
En la imagen 'a', el snapping está desactivado, por lo tanto la colocación del elemento en el terreno es totalmente libre. Por lo tanto el preview puede ser colocado en cualquiera de las

posiciones en las que el usuario quiera, de este modo el preview adquiere un color verde que indica al usuario la correcta situación de la carretera.
En la imagen 'b', el snapping está activado, así pues el usuario se ve forzado a colocar la carretera en una de las posiciones mencionadas anteriormente, de ahí que al no estar cerca de una carretera, el preview se ve con un tono rojo.



(a) Snapping OFF



(b) Snapping ON

Figura 5.12: Funcionamiento del snapping entre carreteras

En la figura anterior 5.12a, se ven dos situaciones que explican el funcionamiento del snapping.
Si el snapping está desactivado, imagen 'a', pese a situar el cursor cerca de otra carretera, el preview no se adherirá a la carretera próxima a ella. En cambio en la imagen 'b', el snapping se encuentra activado, así pues, al situar el preview en la parte superior de la carretera ya colocada, se adhiere sin ningún tipo de problema.

```
1
2  [...]
3  Event e = Event.current;
4  Vector3 mousePos = e.mousePosition;
5  float ppp = EditorGUIUtility.pixelsPerPoint;
6  mousePos.y = scene.camera.pixelHeight - mousePos.y * ppp;
7  mousePos.x *= ppp;
8
9  Ray ray = scene.camera.ScreenPointToRay(mousePos);
10 RaycastHit hit;
11
12 [...]
13
```

```
14 if (isSnapping) {
15   var roadGO = hit.transform.gameObject;
16            if (roadGO.GetComponent<Road>()) {
17              float xPos = hit.transform.position.x;
18                     float zPos = hit.transform.position.z;
19                     float distance = 3f;
20                     if (hit.point.x - hit.transform.position.x >
    distance) {
21                                xPos = hit.transform.position.x +
    roadGO.GetComponent<Renderer>().bounds.size.x;
22                                zPos = hit.transform.position.z;
23       hasSnapped = true;
24                   }
25     else if (hit.point.x - hit.transform.position.x < -distance){
26                                  xPos = hit.transform.position.x -
    roadGO.GetComponent<Renderer>().bounds.size.x;
27                                  zPos = hit.transform.position.z;
28                                  hasSnapped = true;
29                       }
30                       else if (hit.point.z -
    hit.transform.position.z > distance){
31                                  xPos = hit.transform.position.x;
32                                  zPos = hit.transform.position.z +
    roadGO.GetComponent<Renderer>().bounds.size.z;
33                                  hasSnapped = true;
34                       }
35                       else if (hit.point.z -
    hit.transform.position.z < -distance){
36                                  xPos = hit.transform.position.x;
37                                  zPos = hit.transform.position.z -
    roadGO.GetComponent<Renderer>().bounds.size.z;
38                                  hasSnapped = true;
39                       } else {
40                                  hasSnapped = false;
41                       }
42              previewPrefab.GetComponent<Renderer>().material.color
    = hasSnapped ? Color.green : Color.red;
43              previewPrefab.transform.position = new Vector3(xPos,
    hit.point.y, zPos);
44       }
45 }
```

Listing 5.6: Desarrollo del método encargado de realizar el snapping entre carreteras

La función de snapping es la que se encarga de situar el elemento preview sobre una carretera ya colocada. Para ello la función sigue una serie de pasos para identificar la posición

del ratón y la carretera que el ratón está seleccionando (la más proxima).
Para ello, mediante el uso de la clase Event de Unity, el código accede a la posición del ratón. Una vez obtenida, se lanza un rayo desde la cámara del editor contra el primer objeto que golpeé. Así pues, se comprueba que el objeto golpeado, contenga a la clase Road, es decir, que sea una carretera. Una vez comprobado esto, se obtiene la posición X y Z del objeto carretera, y se resta con la posición exacta del punto en el que golpeó el rayo.
Se establece un umbral de distancia en el cual el snapping hará efecto. Así pues si esta resta, supera el umbral de cada una de las coordenadas, se sabe si la posición del ratón está cualquiera de los cuatro lados de la carretera ya situada, colocando el preview en una de estas posiciones.

Por otra parte, también se establecen una serie de atajos o shortcuts para permitir y en otros casos facilitar al usuario la interacción con la herramienta, así pues, vienen definidos del siguiente modo:

- **'B':** Se encarga de colocar en la misma posición y rotación del preview, un elemento con características idénticas a éste si la situación es la correcta.
- **'R':** Rota el preview 90º grados hacia la derecha.
- **'P':** Coloca un nuevo preview en la escena, eliminando el anterior en caso de ser necesario.
- **'S':** Activa o desactiva el snapping.
- **'ESC':** Elimina el preview de la escena.

La imagen 5.13 muestra un ejemplo de colocación perfecta de todas las carreteras al haber hecho uso de todas las herramientas anteriores.

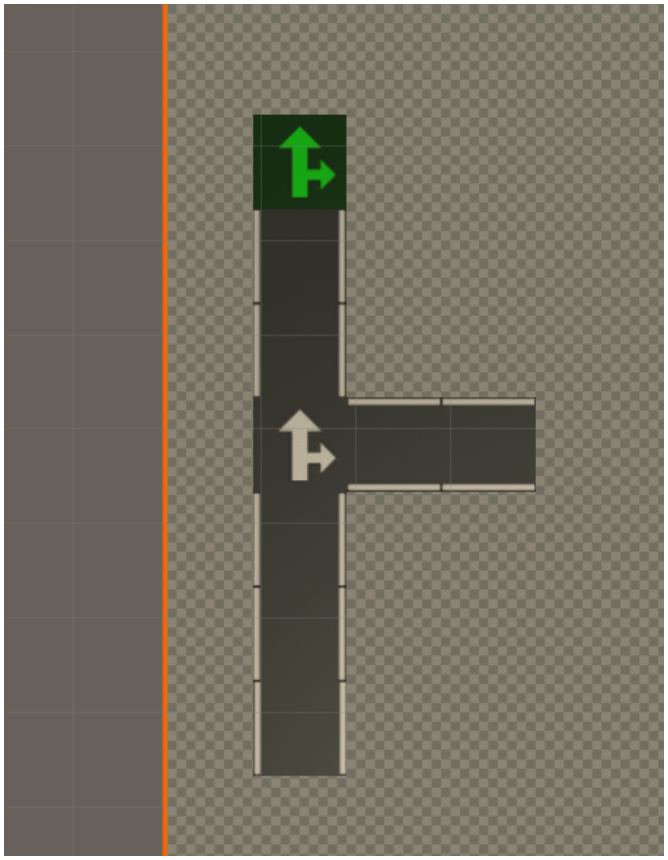


Figura 5.13: Ejemplo de carretera creada con la herramienta de ayuda

5.4.5.2 Implementación del reconocimiento de señales por parte de los agentes

Tal y cómo se menciona en el capítulo anterior, uno de los objetivos marcados por el proyecto es la capacidad que han de tener los agentes para reconocer ciertas señales de tráfico, al igual que hacen los humanos en la vida real. De este modo se dota de realismo e inmersión al usuario dentro del entorno. Así pues la primera señal de tráfico que se toma como referencia es un semáforo.

Un semáforo es un dispositivo de señalización luminosa que se encarga de controlar el tráfico de una carretera. Por lo general, consta de tres luces, una verde, una amarilla y otra de color rojo. La Dirección General de Tráfico (DGT) ha consensuado darle un significado a cada uno de estos colores para permitir o denegar el paso de los vehículos.[8]

- La luz **roja** estática prohíbe el paso del vehículo bajo cualquier tipo de circunstancias.
- La luz **ambar** estática indica la detención del vehículo si esta se ve con la suficiente antelación.
- La luz **verde** permite el paso del vehículo.

Pese a haber distintos tipos de semáforos con distintas combinaciones de luces, como ambar o rojas intermitentes... para la definición de este semáforo se utiliza como base el estándar, es decir aquel que viene en la propia definición de la RAE. Para ello se define un nuevo tipo de elemento, la clase **TrafficLight**.

Un semáforo actúa sobre uno o más vehículos que se encuentran sobre una carretera. Teniendo esto en cuenta se establecen una serie de requisitos que la clase ha de satisfacer para comportarse como debe.

- Definición de colores: verde, ambar y rojo.
- Definición de tiempo transcurrido entre colores.
- Temporizador que permita cambiar el color automáticamente.
- Sincronización con otros semáforos.

- Sincronización inversa con otros semáforos.
- Funcionamiento de vehículos siguiendo el color del semáforo.

Una vez establecidos los criterios para el correcto funcionamiento, del semáforo se procede al desarrollo del mismo:

En primer lugar, la clase TrafficLight proviene de la clase MonoBehaviour de Unity, la cual otorga el método **Update()**, que es llamado una vez por frame. Debido a esto es sencillo la creación de un temporizador. Con una variable que almacene el valor actual de un contador y otra que indique el valor máximo que alcanza. Así pues, si cada frame que pasa se retira la parte proporcional del segundo transcurrido, *Time.deltaTime*, se logra que cada segundo se reduzca en uno el valor del contador.

```
if (countdownTimer > 0) {
            countdownTimer -= Time.deltaTime;
}
```

Listing 5.7: Fragmento de código encargado de realizar una cuenta atrás.

Así pues se solventa el problema de medir los tiempos entre los colores. Del mismo modo mediante el uso de un enumerado se definen los distintos estados que el semáforo puede adquirir.

```
public enum Color {undefined, green, yellow, red };
public Color color;
```

Listing 5.8: Enumeración de colores de los semáforos.

Como nota, se ha añade también un estado extra 'undefined' [5.8] a efectos de inicialización o incluso para simular semáforos estropeados. Al haberse definido distintos estados también se puede sincronizar semáforos entre sí, tanto en su orden natural, como en inverso. Para trabajar con los semáforos sincronizados, se deben anular los temporizadores.

```
public TrafficLight invertedTrafficLight;

private void nextColor() {
        TrafficLight.Color invertedColor = TrafficLight.Color.green;
        switch (color) {
            case Color.red:
                setColor(Color.green);
                invertedColor = Color.red;
                break;
```

```
            case Color.yellow:
                setColor(Color.red);
                invertedColor = Color.yellow;
                break;
            case Color.green:
                setColor(Color.yellow);
                invertedColor = Color.green;
                break;
            default:
                randomColor();
                nextColor();
                break;
        }
        if (invertedTrafficLight) {
            invertedTrafficLight.setColor(invertedColor);
        }
    }
```

Listing 5.9: Método encargado de cambiar el semáforo de un color a otro.

La solución tomada para que un semáforo pertenezca a una carretera, se añadirá una nueva variable en la clase Road, **TrafficLight** , la cual actuará como asociación entre ambas. Si se tiene en cuenta esto, es necesario encontrar una solución para detectar que el vehículo ha detectado su semáforo. Aquí entra en juego la clase **BoxCollider** de Unity. Esta clase tiene una propiedad llamada isTrigger, que elimina la parte física del Collider y lo transforma en un detector, permitiendo el uso de nuevas funciones, como el método **OnTriggerStay()** que se ejecuta cada frame que detecta que algún objeto se encuentra dentro del BoxCollider.

```
 if (trafficLight != null) {
  trafficLightZone = gameObject.AddComponent<BoxCollider>();
            trafficLightZone.isTrigger = true;
            trafficLightZone.size = new Vector3(10,20,10);
}
```

Listing 5.10: Trozo de código encargado de colocar la posición en la cual el vehículo detectará al semáforo.

Así pues, en el código [5.11] se ve reflejado lo siguiente: en primer lugar, la clase *Road*, varia levemente su funcionamiento. En caso de que la clase, tenga un semáforo asociado, se encarga de crear un componente BoxCollider sobre ella. Éste será el encargado de notificar al vehículo, que se encuentra sobre una zona en la que actúa el semáforo.

```
void OnTriggerStay(Collider col) {
```

```
        var color = trafficLight.getColor();
        Vehicle veh = col.GetComponent<Vehicle>();
        veh.onTrafficLight(color);
}
```

Listing 5.11: Método OnTriggerStay de la clase Road

```
    public void onTrafficLight(TrafficLight.Color color) {
        switch(color) {
            case TrafficLight.Color.yellow:
                state = State.slowing;
                break;
            case TrafficLight.Color.red:
                state = State.stoppedOnTrafficLight;
                break;
            case TrafficLight.Color.green:
                state = State.moving;
                break;
            default:
                break;
        }
    }
```

Listing 5.12: Método onTrafficLight de la clase Vehicle

Como se observa en el código descrito anteriormente [5.12] se ha añadido un nuevo estado a la clase **Vehicle** (stoppedOnTrafficLight) que se encarga de detener al agente en caso de que el semáforo se encuentre mostrando el color rojo.
En este mismo método, se gestiona el estado del vehículo según el estado en el que se halle el semáforo. Este método no tiene porqué ser sobreescrito en otras subclases, pero en caso de que se quisiese variar su comportamiento según el semáforo, no supone problema.

Gracias a esto, queda configurado el funcionamiento de cualquier tipo de vehículo ante un semáforo. Es importante determinar, que sólo el vehículo que se encuentre dentro del BoxCollider, llamará a su método interno *onTrafficLight*. El resto de vehículos que se encuentren situados detrás del vehículo detenido, siguen las normas definidas anteriormente por el propio vehículo, deteniéndose entonces cuando la distancia con el vehículo de delante sea lo suficientemente corta. Vease figura 5.15

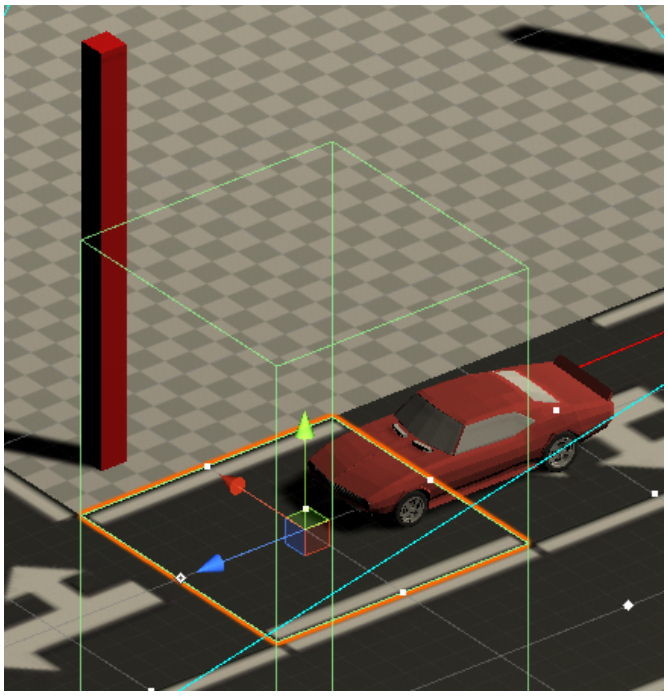


Figura 5.14: Semáforo mostrando su BoxCollider

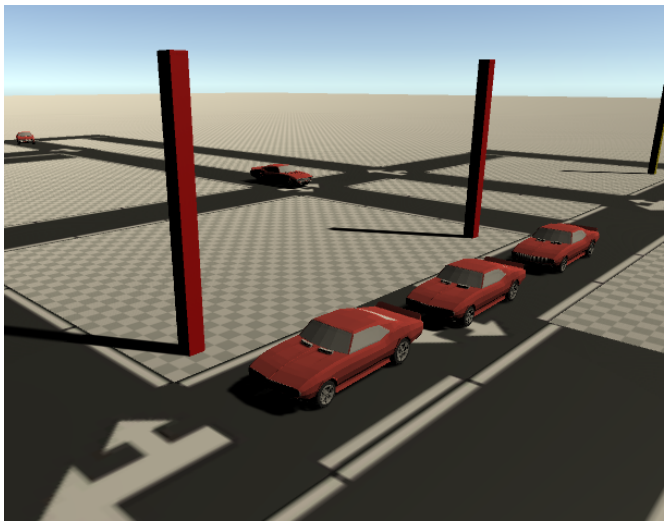


Figura 5.15: Agentes esperando detrás de un vehículo detenido en el semáforo

5.4.5.3 Mejora del movimiento de los agentes

En esta iteración se ha marcado como uno de los objetivos, la mejora del movimiento de los agentes, ya que hasta ahora el movimiento era demasiado simple. Pese a seguir un algoritmo de búsqueda del camino más corto, una vez que el agente alcanza su *waypoint* objetivo, mira al siguiente, resultando un movimiento totalmente imposible, algo contrario al realismo buscado.

```
1 transform.LookAt(waypoint[0]);
```

Listing 5.13: Fragmento de código encargado de hacer que el vehículo apunte a un waypoint.

Al observar el código, se ve claramente el funcionamiento actual del agente. Si se toma en cuenta el funcionamiento de un coche por ejemplo, la posición y rotación del mismo, depende totalmente de la velocidad y rotación de las ruedas, algo que no se toma para nada en cuenta en este proyecto (figura 5.16).

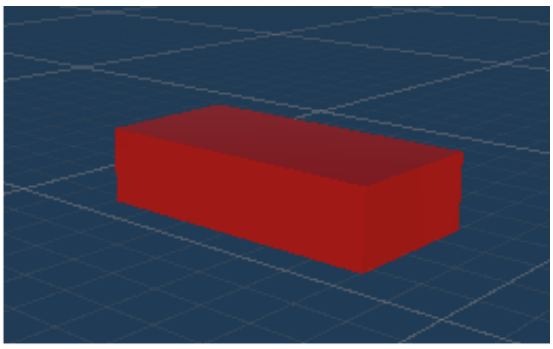


Figura 5.16: Vehículo

Por ello, se vuelve necesario el uso de otra componente auxiliar de Unity, que se simula perfectamente el comportamiento de una rueda. Este componente se llama **WheelCollider**. Gracias a éste, se pueden situar el número de ruedas necesario para la simulación correcta del agente. En este caso es de cuatro ruedas.

Por lo tanto se vuelve necesario añadir cuatro componentes *WheelCollider* (figura 5.17), uno por cada una de las ruedas a añadir. Este componente tiene muchos componentes modificables para obtener los atributos que buscamos en el agente. Como la suspensión, el rozamiento con el suelo, la manera de reaccionar cuando la rueda está de lado... Así pues el usuario puede modificar todos atributos según busque unas caracterísitcas en sus agentes u otros.

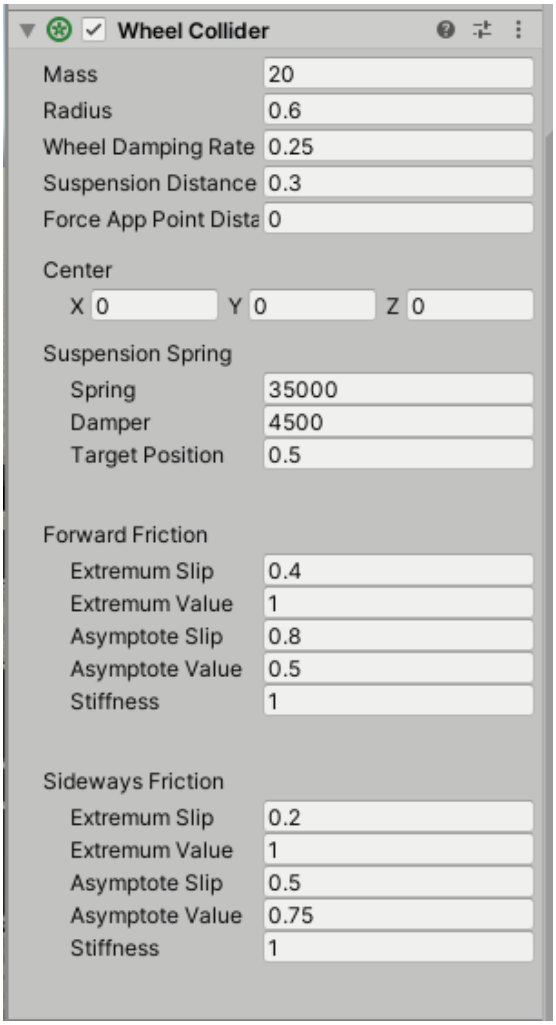


Figura 5.17: Componente WheelCollider

Para facilitar la adición de estos componentes, se va a añadir un modelo de vehículo en tres dimensiones, para sustituir a los ”cubos móviles”. El modelo utilizado se trata de un Chevrolet Camaro (figura 5.18), como se dijo anteriormente, ésto es solo una mejora visual y se encuentra disponible completamente gratis en internet [9].

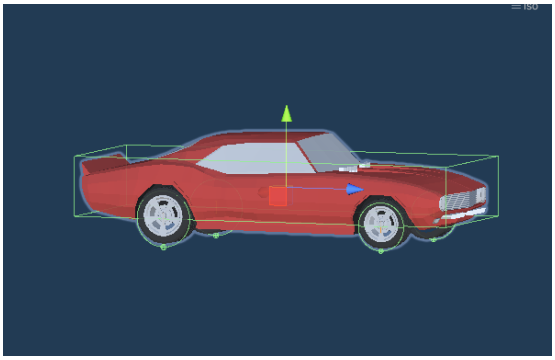


Figura 5.18: Nuevo modelo de la clase vehículo

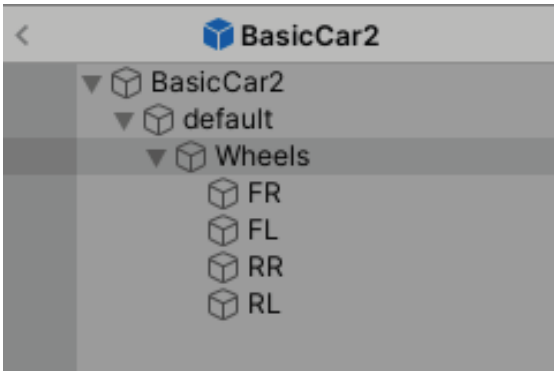


Figura 5.19: Estructura del agente

Como es necesario hacer una distinción de cada una de las ruedas, es necesario también reestructurar el propio prefab del vehículo. Añadiendo un componente WheelCollider en cada una de las transformaciones correspondientes a las ruedas, véase la (figura 5.19. Se ha de tener en cuenta que las ruedas de los vehículos no suelen ser iguales en cada eje, si no que existe una pequeña variación entre ellas. Por este motivo, es necesario modificar el tamaño de la rueda (radio) de los componentes para adaptarlos a los del nuevo modelo.
Después de haber mencionado esto, cabe destacar que el coste de añadir un nuevo vehículo en un futuro, sería el posicionamiento de las ruedas y la adaptación del script, ya que se ha preparado para agentes de cuatro ruedas por el momento. La inclusión de estos componentes, hace necesario una actualización de la clase *Vehicle*. Así pues, ya no es necesario que el Rigidbody sea el encargado de mover el propio agente. Si no que las ruedas deben generar su torque correspondiente (fuerza aplicada sobre un objeto y que lo hace girar) para poder mover el Rigidbody (la carrocería) que ellas sostienen. La clase *WheelCollider* contiene los métodos necesarios para emular correctamente todos los movimientos de la rueda. Así pues es necesario establecer también una serie de constantes y variables que limiten el correcto funcionamiento del agente.

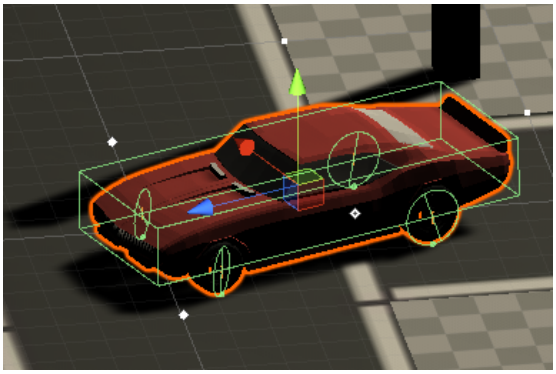


Figura 5.20: Vehículo con las ruedas indicando la dirección que sigue

La posibilidad de añadir torque y movimiento a cada una de las ruedas individualmente, ofrece la posibilidad de crear vehículos con distintos tipos de **tracción**. Se denomina tracción al mecanismo encargado de transmitir la fuerza y el movimiento del movimiento del motor a las ruedas [10]. La tracción de un vehículo puede ser de tres tipos generalmente, tracción delantera (TD), tracción trasera (TT) o distintos tipos de tracción a las cuatro ruedas (4x4, AWD o 4WD)

- **4x4**: tracción a las cuatro ruedas regulable en cuanto a la potencia
- **AWD**: tracción a las cuatro ruedas permanente.
- **4WD**: tracción a las cuatro ruedas al gusto del conductor.

Dependiendo del tipo de tracción de un vehículo, el agente requerirá un tipo de conducción u otro. Cabe mencionar, que la mayoría de los vehículos que se encuentran en la calle son de tracción delantera, dicho esto, el agente desarrollado como base, contendrá también el mismo tipo de tracción. De este modo, el motor del vehículo dirigirá la potencia de las ruedas al eje delantero, que serán las motrices. Estas mismas ruedas son las que llevarán la dirección del agente.
Al estar hablando del movimiento del agente y de las tracciones utilizadas, también es importante mencionar el método utilizado para detener al agente. En cuanto a esto es necesario conocer el funcionamiento del freno de pie de un vehículo, que es el que se utiliza como normal general cuando está en tránsito. Cuando un conductor en la vida real, utiliza el freno de pie, éste actua sobre las cuatro ruedas del vehículo al mismo tiempo. Si el agente realiza esta misma función, dotará de mayor realismo e inmersión al usuario.

```
public abstract class Vehicle : MonoBehaviour
{

    [...]
    public WheelCollider fr_wheel;
    public WheelCollider fl_wheel;
    public WheelCollider rr_wheel;
    public WheelCollider rl_wheel;

    public float speedForce = 600f;
    public float brakeForce = 600f;
    public float slowForce = 50f;
    float maxSteering = 55;
    [...]

}
```

Listing 5.14: Descripción de las variables de la clase vehicle

El código descrito anteriormente [5.14] establece el torque máximo aplicable a cada una de las ruedas en los instantes de movimiento (*speedTorque*), es decir, cuando el vehículo se encuentra sin limitaciones de movimiento. A mayor valor de esta variable, el agente alcanzará antes la velocidad objetivo.
También indica el torque máximo aplicable en los momentos que el vehículo no ejerce tanta fuerza para moverse (*slowTorque*), que se produce en momentos que por ejemplo, se encuentra próximo a otro vehículo. Cuanto más alta sea esta variable, la ruedas tendrán mayor aceleración cuando el vehículo se encuentra en el estado de cautela.
Por último también se determina la fuerza de frenado que la rueda debe de ejercer en caso de frenada. Cuanto mayor sea este valor el agente necesitará menos tiempo para su completa detención.
La variable *maxSteering*, se encarga de limitar el ángulo máximo que la rueda puede girar. Es decir, desde un eje perpendicular al suelo, la rotación máxima alcanzable. En resumen, a mayor valor de esta variable, el agente virará con mayor facilidad.

En la vida real, a la hora de conducir un vehículo, lo prioritario es la seguridad del ocupante. Para ello se ha de mantener una distancia de seguridad en todo momento con aquel que se encuentra delante de él o en sus laterales. Por lo tanto en EasyRoadSystem, el agente ha de mantener en todo momento una distancia prudente con el resto.
Al tener claro que se pueden aplicar distintas fuerzas a las ruedas para generar un tipo de movimiento u otro, el vehículo aplicará una cantidad u otra de torque según la proximidad a la que se encuentre de otro agente.

```
void FixedUpdate()
{
    [...]
    switch (state) {
        case State.moving:
            if (waypoints.Count > 0) {
                selectNextWayPoint();
                if (waypoints.Count > 0 && destination != Vector3.zero) {
                    steer();
                    setTorque(speedForce, 0);
                }
            } else {
                state = State.calculating;
            }
            break;
        case State.calculating:
            selectRandomDestination();
            break;
        case State.slowing:
            if (waypoints.Count > 0) {
                selectNextWayPoint();
                if (waypoints.Count > 0 && destination != Vector3.zero) {
                    steer();
                    setTorque(slowForce, 0);
                }
            } else {
                state = State.calculating;
            }
            break;
        case State.stop:
            setTorque(0f, breakForce);
                if (waypoints.Count > 0 && destination != Vector3.zero) {
                    state = State.moving;
                }
                break;
            case State.parking:
                break;
            default:
                break;
    }
}
    [...]
```

```
43
44 private void setTorque(float motorTorque, float brakeTorque) {
45     fl_wheel.motorTorque = motorTorque;
46     fr_wheel.motorTorque = motorTorque;
47     rl_wheel.brakeTorque = brakeTorque;
48     rr_wheel.brakeTorque = brakeTorque;
49     fl_wheel.brakeTorque = brakeTorque;
50     fr_wheel.brakeTorque = brakeTorque;
51 }
52
53 private void steer() {
54     Vector3 relativeV =
       transform.InverseTransformPoint(waypoints[0]);
55     float newSteer = (relativeV.x / relativeV.magnitude *
       maxSteering);
56     fl_wheel.steerAngle = newSteer;
57     fr_wheel.steerAngle = newSteer;
58 }
59
60    [...]
61 }
```

Listing 5.15: Fragmento de código de de las modificaciones clase Vehicle.

Así pues, en el código anterior [5.15] se observan los dos métodos que se encargan de gestionar el movimiento de las ruedas.
El método **setTorque(float motorTorque, float brakeTorque)** se encarga de simular la tracción que le otorga el motor a cada uno de los ejes del vehículo. Como se mencionó anteriormente, el agente es de tracción delantera, la fuerza del motor se dirige al eje delantero, las ruedas delanteras reciben el valor de la variable *motorTorque*. Así pues, en párrafos anteriores se comentó que el freno de pie actúa sobre las cuatro ruedas a la vez, en el código se refleja esto, colocando el valor de la variable *brakeTorque* sobre el torque de frenado de las cuatro ruedas.
El segundo, es el método **steer()**. Lo que éste hace, es transformar las coordenadas globales a locales obteniendo un vector entre la posición del agente y su próximo waypoint. Una vez hecho ésto, se multiplica por la variable *maxSteering*, para conseguir que la rueda apunte hacia el waypoint requerido.

Gracias a esta nueva implementación del movimiento de los agentes, se ha dejado de lado el movimiento antiguo de los agentes, que consistía en apuntar al siguiente waypoint nada más hubiese alcanzado el anterior. Ahora los vehículos se encuentran dotados de la capacidad de realizar giros infinitamente más realistas, aunque a su vez limitados por su capacidad de

giro, al igual que ocurre en la vida real.

5.4.6 Pruebas

Tests de la iteración 4		
Nombre del test	Resultado esperado	Resultado recibido
Se ha facilitado la creación de carreteras	S	S
Se ha creado una nueva ventana (editor) en el cual se pueden llevar a cabo todas las variaciones de carreteras implementadas	S	S
Las nuevas carreteras se colocan perfectamente alineadas con el resto	S	S
No se puede colocar una carretera en el lugar en el que ya existe otra	S	N
Las carreteras que no pueden ser colocadas se ven de color rojo	S	N
Las carreteras que pueden ser colocadas se ven de color verde	S	S
Las carreteras nuevas se pueden rotar mediante la tecla 'R'	S	S
El objeto a visualizar en el editor, se actualiza correctamente al elegir uno distinto	S	N
En el editor, se puede alternar entre los dos modos de colocación posible, mediante la tecla 'S' o el checkbox disponible	S	S
Al cerrar el editor o pulsar la tecla 'ESC' el preview desaparece	S	S
Se colocan las nuevas carreteras en la escena mediante la tecla 'B'	S	S
Se puede elegir la carretera a colocar haciendo click en el selector de objetos del editor	S	S
Tras cambiar entre un objeto y otro, los anteriores tests mantienen su funcionamiento normal	S	S
El nuevo elemento colocado, mantiene la apariencia del preview que se mostraba anteriormente. Sin haber perdido los colores originales tras cambiarlos	S	S

Tabla 5.3: Tabla que contiene la descripción de los test realizados durante la iteración 4 (primera parte).

Tests de la iteración 4		
Nombre del test	Resultado esperado	Resultado recibido
Se puede colocar una carretera haciendo snap cuando no hay otra a la que pegarse	N	N
Se ha creado un semáforo que es capaz de ser reconocido por cualquiera de los agentes desde un punto concreto.	S	S
El semáforo alterna entre los tres colores habituales	S	S
Los vehículos se detienen al detectar el semáforo en rojo	S	S
Los vehículos regulan su velocidad al detectar el semáforo en ámbar	S	S
Los vehículos prosiguen su paso al detectar el semáforo en color verde.	S	S
El tiempo de espera para alternar los colores del semáforo es personalizable	S	S
Los semáforos funcionan pese a que no haya ningún vehículo que pueda detectarlos	S	S
Dos o más semáforos pueden funcionar en sincronía o asincronía para permitir la creación de pasos más complejos	S	S
Los vehículos disponen de ruedas funcionales	S	S
Los vehículos seguirán la dirección marcada por las ruedas	S	S
Los coches disponen de un nuevo modelo en 3D	S	S
Los vehículos funcionan transmitiendo sus fuerzas a las ruedas, y no mediante el empuje de la carrocería (como antiguamente)	S	S
Los vehículos alternarán entre distintos estados según el mismo considere necesario	S	S
Se puede controlar un vehículo	N	N

Tabla 5.4: Tabla que contiene la descripción de los test realizados durante la iteración 4 (segunda parte).

Las pruebas realizadas en esta iteración incluyen también aquellas realizadas en la anterior, realizando también las de la segunda iteración.

5.4.7 Planificación

Tras evaluar el correcto funcionamiento de la aplicación durante el transcurso de esta iteración, se da paso a la útima iteración de este proyecto.
La interfaz de usuario de la librería podría contener algún elemento que facilitase su utilización. Por ejemplo, todas las carreteras actuales tienen las mismas texturas y pese a que sea decisión del usuario cambiarlas o no, la librería quiere ofrecer unas por defecto.
Como también se puede observar en el apartado de pruebas de esta iteración, existen algunos problemas a la hora de colocar varias carreteras en el mismo lugar, algo que no debería de ocurrir.

5.5 Iteración 5

Ésta es la última iteración del proyecto ERS, la cual simplemente se centrará en unas pocas mejoras visuales y en el arreglo de algunos bugs.

5.5.1 Objetivos

- Mejora de GUI.
- Pulimiento de bugs

5.5.2 Análisis de riesgos

Actualmente, el proyecto coloca correctamente las carreteras y demás elementos que se usan en la herramienta, pero visualmente, aún podría llamar más la atención. En cualquier carretera convencional, las carreteras tienen señales pintadas en el suelo, que indican aquellos movimientos que se pueden realizar dentro de ella. De esta manera, las intersecciones se indican de una manera más visual, otorgando realismo, inmersión y un entorno más atractivo.

5.5.3 Análisis de requisitos

Debido a esto, el análisis de requisitos para esta iteración constará solamente de lo siguiente:

- El usuario debe tener mejor capacidad visual para visualizar los elementos que está colocando sobre la escena.
- El usuario debe ser capaz de observar como cambia el objeto a previsualizar a la vez que él lo selecciona.
- Hacer uso de **texturas**. Bastaría con colocar una imagen sobre el plano, que indique las posibles direcciones que el vehículo puede tomar.
- **Dibujar** en tiempo de ejecución cada uno de los símbolos que se quieran mostrar en el entorno.

En la anterior iteración se encontraron algunos bugs que deben ser pulidos sencillamente en ésta. Por ejemplo, cuando se utiliza la herramienta del proyecto y se activa el snapping, existen diversos puntos en los que se colocará el preview en el punto exacto en dónde ya hay otro objeto colocado. También existe la posibilidad de que el EditorWindow no cambie la

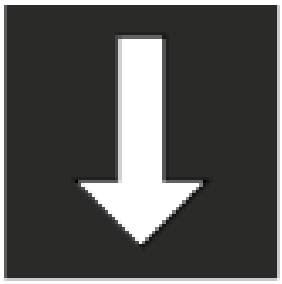
ventana del objeto seleccionado. Estos bugs se abordarán durante el transcurso de la iteración.

5.5.4 Análisis

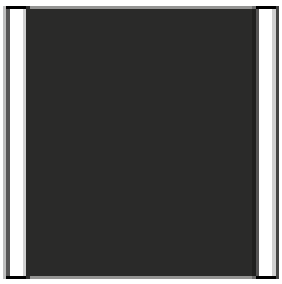
La etapa final de este documento buscará cumplir todos los requisitos restantes, funcionales y no funcionales. Para el caso de los funcionales, la herramienta ya cumple con todos ellos. Por ello el único labor en cuanto a ellos será comprobar que se sigan cumpliendo.
Para el caso de los requisitos no funcionales [4.1.2], faltan por acometer los siguientes: el RNF1 y el RNF4. Estos dos se cumplirán al finalizar esta iteración, mediante la creación de un manual de usuario y la adición a éste de métodos de contacto con el autor.w

5.5.5 Desarrollo

Cómo se ha mencionado durante el análisis de riesgos, las mejoras visuales de esta iteración se centran en colocar sobre los planos de la carretera distintas señales que indiquen las posibles direcciones a tomar.
Para ello se ha realizado un diseño personalizado para cada una de las distintas carreteras, siempre basados en las señales que se encuentran dibujadas en la propia carretera.
Con los diseños preparados y en el formato adecuado (.png), basta con crear el material en Unity y vincularlo desde la ventana Project a cada uno de los *prefabs*. Desde ese momento, al instanciar cada una de las carreteras, se mostrará la nueva textura creada.



(a) Señal de dirección de frente

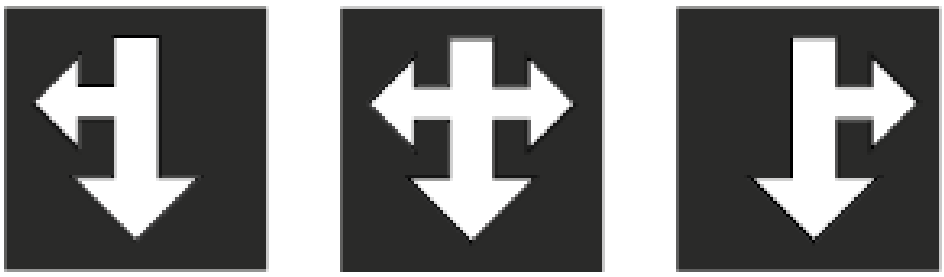


(b) Carretera estándar

Figura 5.21: Carreteras con la misma dirección

El otro objetivo a abordar, es buscar la solución de un bug mediante el cual se permitía la instanciación de varias carreteras en un mismo punto.
La solución pasa por establecer un control estricto que impida dicha casuística.

```
1 Collider[] colliders =
      Physics.OverlapSphere(previewPrefab.transform.position, 1f);
```



(a) Dirección derecha (b) Cualquier dirección (c) Dirección izquierda

Figura 5.22: Carreteras con distintas direcciones

```
2  if (colliders.Length > 1) {
3   foreach(Collider collider in colliders) {
4             if (collider.gameObject ==
      previewPrefab.transform.gameObject) continue;
5                     if (collider.gameObject.GetComponent<Road>()
      == null) continue;
6
      previewPrefab.GetComponent<Renderer>().material.color =
      Color.red;
7                     hasSnapped = false;
8                     break;
9           }
10 }
```

Listing 5.16: Fragmento de código encargado de comprobar si existe un objeto previamente colocado en la misma posición

El comportamiento del código descrito [5.16] es el siguiente:

- Se detectan los objetos que utilicen físicas en una esfera de radio pequeño justo en el punto dónde se encuentra el preview.
- Si el objeto con el que choca, es el propio preview se ignora.
- Si el objeto con el que choca, no es una carretera, también se ignora.
- Si se continua ejecutando el bucle, el color del preview se cambia a rojo y la variable hasSnapped se pone a false. Esto impide que se pueda construir exactamente en ese punto.

El otro error que se ha encontrado en la anterior iteración consiste en que el visor de la herramienta de ayuda de ERS, no se actualiza correctamente
Este bug se producía porque una vez que se creaba el editor con el método *gameObjectEditor = Editor.CreateEditor(selectedPrefab);* no se volvía a actualizar pese a cambiar el objeto seleccionado. Este comportamiento es el adecuado, así que la solución para este error consiste en volver a crear el editor si se detecta que el objeto que muestra, no es el mismo que el seleccionado.
Aprovechando el arreglo del fallo, también se añadió la eliminación del Editor una vez que se detecte que no existe ningún objeto seleccionado.

```
1  if (selectedPrefab != null){
2              prefabPath =
       PrefabUtility.GetPrefabAssetPathOfNearestInstanceRoot(selectedPrefab);
3              if (gameObjectEditor == null) {
4                  gameObjectEditor =
       Editor.CreateEditor(selectedPrefab);
5              } else {
6                  if (gameObjectEditor.target != selectedPrefab) {
7                  DestroyImmediate (gameObjectEditor);
8                  gameObjectEditor =
       Editor.CreateEditor(selectedPrefab);
9                  }
10             }
11
12             GUIStyle bgColor = new GUIStyle();
13             bgColor.normal.background =
       EditorGUIUtility.whiteTexture;
14
       gameObjectEditor.OnInteractivePreviewGUI(GUILayoutUtility.GetRect(256,
       256), bgColor);
15         } else {
16             if (gameObjectEditor != null)
17             {
18                 DestroyImmediate (gameObjectEditor);
19             }
20         }
```

Listing 5.17: Fragmento de código encargado de cambiar la previsualización del objeto a colocar en la escena.

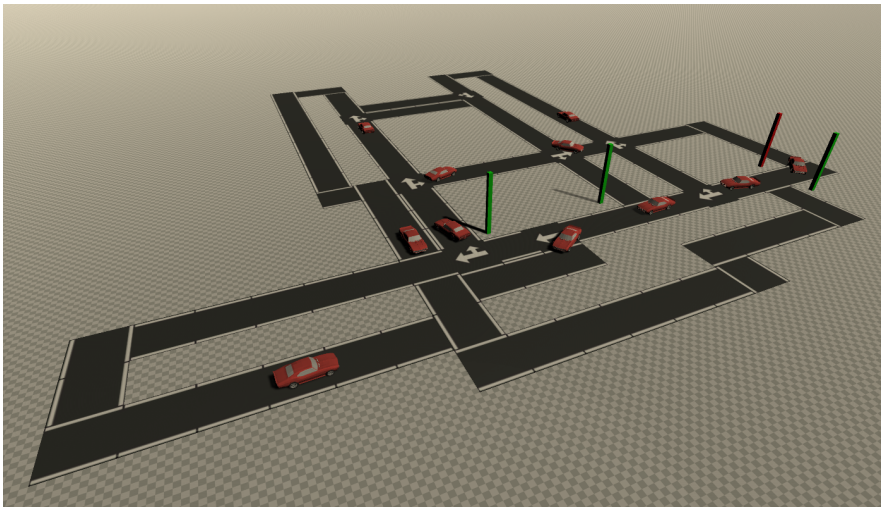


Figura 5.23: Ciudad realizada con ERS

5.5.6 Pruebas

Tests de la iteración 5		
Nombre del test	Resultado esperado	Resultado recibido
Se han añadido nuevas texturas al proyecto	S	S
Existe una textura para cada tipo de carreteras o intersección.	S	S
Las texturas se ven correctamente en cada uno de los tipos de carretera disponibles	S	S
El editor actualiza su objeto a colocar correctamente	S	S
El editor es capaz de detectar si ya existe una carretera colocada en el punto en el cual se quiere colocar una nueva	S	S

Tabla 5.5: Tabla que contiene la descripción de los test realizados durante la iteración 5.

Las pruebas realizadas en esta iteración incluyen también aquellas realizadas en todas las anteriores, desde la segunda hasta la última etapa de desarrollo.

Capítulo 6

Conclusiones

6.1 Concordancia de objetivos cumplidos

En primer lugar, cabe resaltar que todos los objetivos que Easy Road System buscaba, se han cumplido satisfactoriamente (ver sección 1.3).
A continuación se explicará de forma breve, como cada uno de ellos se ha realizado:

- **Generación de entornos viales realistas**: la herramienta es capaz de generar un mundo virtual en el cual aparezcan los elementos indispensables para la circulación vial. Esto puede ser generado desde un entorno anteriormente creado y al que se le quieran añadir carreteras o desde uno que previamente no existiera, iniciándose desde el principio.

- **Identificación del entorno (señales) que los agentes tienen que respetar**: Easy Road System dota a los vehículos de una forma de identificar una señal que se encuentre en su trayectoria. Para ello se comprueba la posición de un Collider con la posición del vehículo. Si esta posición coincide, el vehículo tomará las medidas oportunas para respetar la señal que acaba de observar.

- **Colocación de carreteras, señales y agentes**: la librería provee los elementos necesarios para que el usuario sea capaz de situar cada uno de los objetos dentro del entorno. se permite la colocación de los elementos citados anteriormente de la manera típica de Unity, arrastrándolos directamente a la escena. Cabe añadir que también se ha diseñado una ventana con la cual esta colocación, se realizará de manera infinitamente más precisa y segura para su buen funcionamiento.

- **Facilidad de integración de la librería en cualquier proyecto**: la manera que ofrece Unity para la utilización de librerías no podría ser más sencilla, bastará con importar Easy Road System al proyecto y todas la clases que han sido desarrolladas durante el transcurso de este trabajo se establecerán en Unity.

- **Simplicidad de utilización mediante una interfaz**: como ya se mencionó anteriormente, para la colocación de elementos (sobre todo las carreteras), se ha creado una ventana funcional con la cual se podrán añadir todos los elementos deseados de manera rápida y precisa.

- **Posibilidad de añadir vehículos determinados por el usuario**: una vez importada la librería, el usuario puede modificar los vehículos que se ofrecen por defecto en la librería e incluso crear nuevos. Para ello solamente tendría que copiar la clase Vehicle y modificarla a su gusto. Si lo que busca es una modificación estética, tendría que cambiar el modelo asociado al vehículo que desee.

- **Garantizar el funcionamiento autómata de cada agente, sin la implicación del usuario**: debido a cómo está desarollada la librería, el usuario sólo deberá colocar las carreteras, un vehículo y la clase RoadManager para que la librería comience a funcionar. Todo esto se puede colocar de manera rápida y sencilla mediante los botones que provee la interfaz de la herramienta. De este modo, la clase RoadManager se encargará de que cada agente funcione de manera independiente de los otros y sin la interacción del usuario.

6.2 Lecciones aprendidas

Después de la realización del proyecto, considero que he adquirido **experiencia** en Unity, herramienta por la que siempre he sentido una gran atracción.
Sinceramente, me he sentido muy afortunado de poder realizar mi TFG con una herramienta que me encanta y aún encima de un tema el cual adoro. Nunca había hecho un uso tan profundo de algoritmos de búsqueda por ejemplo ni nunca se me habría ocurrido diseñar mi propia herramienta dentro de Unity para conseguir una personalización más cómoda de cualquier aspecto relacionado con alguno de mis scripts. Sinceramente, era bastante tedioso tener que ir colocando cada una de las carreteras paso por paso y dejándolas lo suficientemente cerca como para que funcionasen correctamente. La herramienta de creación ha sido la mejor ayuda que pude implementar.
He aprendido a desarrollar una variante del **algoritmo A*** ya que he tenido que aplicarlo utilizando carreteras. Personalmente, nunca había utilizado ningún algoritmo de búsqueda para ningún tema no escolar y sinceramente, ver como poco a poco vas logrando el objetivo que te propones es una experiencia muy reconfortante.
He aprendido a **valorar** los costes en tiempo que puede significar una mala decisión. Por ejemplo el mismo algoritmo A* o hacer llamadas en funciones que se van a estar repitiendo durante la ejecución de la aplicación cuando no son necesarias.

6.3 Competencias con la titulación

En cuanto al proyecto, se han tenido en cuenta diversos factores implicados con diversas ramas de la ingeniería informática.
EasyRoadSystem es una herramienta que se ha desarrollado siguiendo un estricto protocolo de desarrollo, con un ciclo de vida adecuado al proyecto y una metodología de trabajo basada en iteraciones. En el proyecto se han tenido en cuenta las diversas etapas por las que pasaría el proyecto, siguiendo un análisis, planificación, pruebas, documentación. Todo esto implica que se ha tenido una *preocupación por la calidad del software*, buscando garantizar que todo lo que está entregado cumple con los requisitos tomados previamente y que sea correcto.

También se ha buscado un uso correcto de *principios de diseño*:

- El código utilizado en ERS, no se repite, cada clase utiliza sus propios métodos y otros que pueden ser utilizados de manera más común, vienen encapsulados dentro de una misma clase. Por ejemplo, los métodos matemáticos en la clase *Mathf*.
- Principio de responsabilidad única: se ha buscado que las clases de ERS, tengan una función propia y que la misma función no esté en clases distintas.
- Se pueden extender las clases de manera sencilla: por ejemplo bastaría con extender la clase Vehícle para generar un vehículo que pudiese comportarse de manera distinta ante los semáforos.
- Principio de sustitución de Liskov: cualquiera de las carreteras, extienden el funcionamiento de la clase Road. De ella sobreescriben el método loadConnectionPoints() haciendo una llamada al funcionamiento de ese mismo método de la clase padre, añadiendo también sus casuísticas. Gracias a esto, la clase hija podrá sustituir a la clase que ha extendido sin ningún tipo de problema.
- Se ha buscado que el código sea limpio y no enrevesado, manteniéndolo simple.

Del mismo modo, al haber utilizado Unity3D como motor gráfico, se ha facilitado enormemente el uso de patrones de diseño:

- **Singleton**: sólo existe una clase RoadManager y es única, habiendo una unica forma de acceso hacia ella.
- **Constructor**: se puede modificar los valores de cada uno de los vehículos por separado y de manera sencilla.

- **Factoria**: el método de creación de los elementos de la escena es siempre el mismo.
- **Fachada**: la clase RoadManager se encarga de tratar con la parte del código más compleja, simplificando el uso para las demás clases.
- **Observador**: se utilizan diversos métodos para detectar si un objeto sigue en contacto con otro.
- **Estado**: tanto los vehículos como los semáforos, varían su comportamiento en función del estado en el que se encuentran.
- **Plantilla**: se utilizan clases como base (abstractas) que no pueden ser instanciadas por si mismas. Han de ser extendidas, utilizando las mismas como base para sus hijas.

Mencionar también el uso de un motor gráfico (Unity), como medio para visualizar los conceptos implementados en un lenguaje de programación.
Implementación de diversos algoritmos así como valorado la utilización de procedimientos ya conocidos para alcanzar unos objetivos. Así pues, se ha desarrollado una IA capaz de gestionar los caminos que deben seguir los agentes para alcanzar su destino mediante un proceso lógico, habiendo valorado incluso el uso de redes neuronales para lograr este funcionamiento.
También se ha implementado una función de *snapping* la cual permite, mediante una serie de cálculos matemáticos, la colocación precisa de cada uno de los elementos de ERS.
También se ha planteado la herramienta, como un proyecto a largo plazo, en el cual se debía de gestionar los recursos disponibles mediante una toma de decisiones en cada una de las iteraciones, haciendo un análisis de riesgos previo al desarrollo de cada una de las funcionalidades.

Capítulo 7

Futuros Desarrollos

Easy Road System tiene muchísimo camino que recorrer todavía, muchísimas ideas que por falta de tiempo no han podido ser implementadas.
De este modo, a continuación se enumerarán una serie de ideas junto al motivo por el cual sería interesante desarrollarlas:

- **Adición de un agente controlado por el usuario**: en una gran cantidad de entornos en los que están presentes vehículos móviles, existen muchísimos casos en los que añadir que un vehículo pueda ser controlado por una persona es ampliamente beneficioso. Por ejemplo en el caso de una autoescuela o de juegos de simulación de conducción.

- **Adición de nuevas entidades como pueden ser autobuses o motos**: a pesar de que todos los vehículos pueden ser personalizados por los usuarios, Easy Road System quiere establecer una serie de vehículos base para que el desarrollador que utilice la librería, tenga que complicarse lo mínimo posible, como pueden ser vehículos de varios ejes o motos.

- **Nuevas clases de carreteras, que permitan por ejemplo la circulación de dos vehículos al mismo tiempo podría ser interesante o rotondas**: actualmente la herramienta, cubre las carreteras más básicas existentes, que son aquellas de un sólo carril. En la vida real, estas carreteras no son las únicas, ya que hay algunas que permiten la circulación de varios vehículos en distintas direcciones. Por otro lado, las rotondas son un elemento clave a la hora de la circulación, pues pueden controlar intersecciones de varios carriles sencillamente.

- **Generación de carreteras basadas formatos de intercambio, como XML, JSON o texto plano, se podrían extraer los caracteres de éstos y generar escenas basadas en ellos**: Esta característica podría estar enfocada a personas que por el motivo que sea, quieran exportar un documento que contenga información de su escena desde

otro formato. Por ejemplo aquellas que se descarguen un mapa de carreteras y quieran rápidamente trabajar con esta herramienta.

- **Adición de peatones y pasos de cebra**: Son dos elementos clave en la circulación, es necesario que en una versión futura se contemple su aparición. Gracias a ella aumentaremos todavía más el realismo que esta librería puede llegar a ofrecer.
- **Mejoras en la creación de carreteras**: En este punto del desarrollo, la librería puede colocar carreteras de manera perfecta en ángulos de 90 grados sobre la anterior carretera. Se podría estudiar una forma mediante curvas de Bézier por ejemplo y modificación de la forma de la carretera, para crear conseguir generar curvas que se vean mucho más naturales.

Apéndices

Apéndice A

Material adicional

A.1 Glosario de términos utilizados en Unity3D

Algunos de los componentes usuales que aparecen en este proyecto sobre Unity3d, son:

- *BoxCollider*: Da a los objetos la posibilidad de detectar colisiones.
- *Componente*: Cada uno de los elementos que se puede añadir a un objeto.
- *EditorWindow*: Es otra clase sobre la que derivar un script. Permite la visualización de variables en la pantalla. Es usado para crear ventanas.
- *Escena*: Lugar virtual en donde se colocan todos los objetos para su uso.
- *GameObject*: Es el objeto en sí. Puede contener otros GameObject como hijos.
- *GetComponent<T>()*: Método que permite acceder a cada uno de los componentes propios del motor.
- *MonoBehaviour*: Es una clases base sobre la que derivar los scripts. Otorga métodos que permiten actualizar el objeto a cada frame.
- *Prefab*: Se denomina así a un gameobject que se guarda para un uso futuro dentro de una carpeta.
- *Raycast*: Rayo virtual que golpea contra cualquier gameobject que contenga un collider.
- *Renderer*: Permite que el objeto se visualice en la escena.
- *Rigidbody*: Este componente hará que al objeto le afecten las fuerzas, como son la gravedad.

- *Script*: Componente el cual contiene código.
- *Textura*: Imagen 2D que otorga a los objetos otro aspecto.
- *Transform*: Contiene la posición, rotación y escala del objeto.
- *Update()*: Método que se llama a cada frame transcurrido de la ejecución de Unity.
- *WheelCollider*: Se encarga de emular las físicas que tienen una rueda, suspensión, torque, rozamiento…

Apéndice B

Comparativa de Unity3D frente a otros motores gráficos

Comparativa de motores gráficos					
Nombre del test	**Soporte 3D**	**Tienda interna**	**Multiplataforma**	**Comunidad**	**Dificultad**
Unity3D	S	S	S	Muy grande	Fácil
Unreal Engine 4	S	S	S	Muy grande	Media
Godot	S	S	S	Media	Fácil
jMonkey	S	N	S	Pequeña	Media

Tabla B.1: Tabla comparativa de Unity3D frente a otros motores gráficos.

Unity3D y Unreal engine son los dos motores gráficos más importantes por excelencia. Ambos disponen de una comunidad increíblemente grande y todas las clases que se utilizan en ellos están muy bien documentadas. Godot es un motor gráfico más nuevo pero que está cogiendo bastante fuerza últimamente, debido a esto, su comunidad es bastante reducida comparada con las dos anteriores. JMonkey es un motor con una comunidad mucho más pequeña, ya que es bastante limitado comparado con cualquiera de las otras tres opciones.

Unity, Unreal y Godot tienen una tienda interna que facilita en grandes medidas, la capacidad de compartir o adquirir distintas librerías o herramientas que otros desarrolladores hayan querido compartir.

Pese a que los cuatro sean multiplataforma, no todos tienen la misma facilidad para trabajar con una u otra. En esta característica destaca sobre todo unity que soporta hasta 25 plataformas diferentes.

Por último, algo muy importante a la hora de seleccionar un motor gráfico, es la curva de aprendizaje necesaria para dominar el entorno. Como se ha mencionado anteriormente, Unity3D y Unreal se encuentran muy igualados entre sí y si la sencillez de uno u otro es un factor clave para tomar nuestra decisión, el resultado sería Unity3D, ya que Unreal posee un montón de elementos que pueden crear una complejidad inimaginable solamente para el tema de iluminación o materiales por ejemplo.

Apéndice C

Diagramas UML

En el siguiente apéndice se muestran los diagramas UML utilizados para representar las clases utilizadas por Easy Road System.

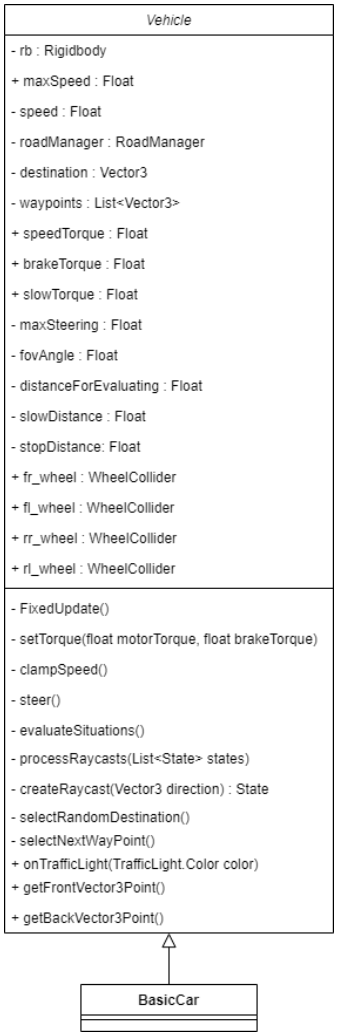


Figura C.1: Vehicle y BasicCar UML

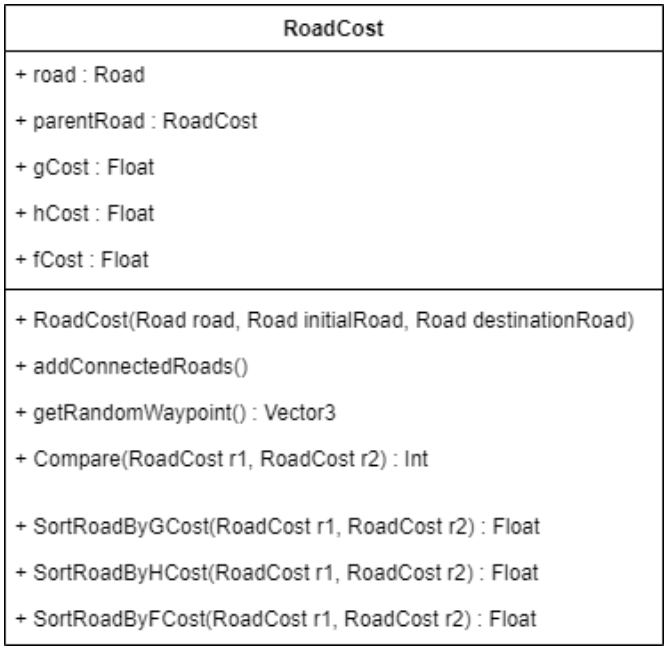


Figura C.2: RoadCost UML

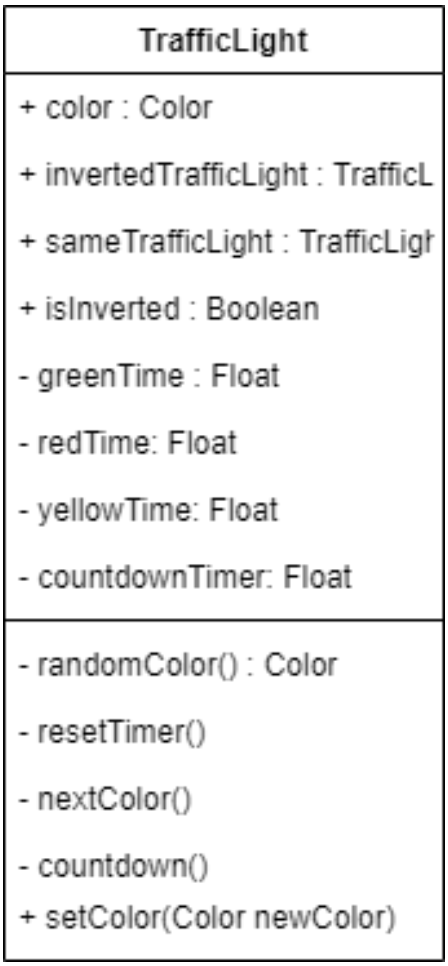


Figura C.3: TrafficLight UML

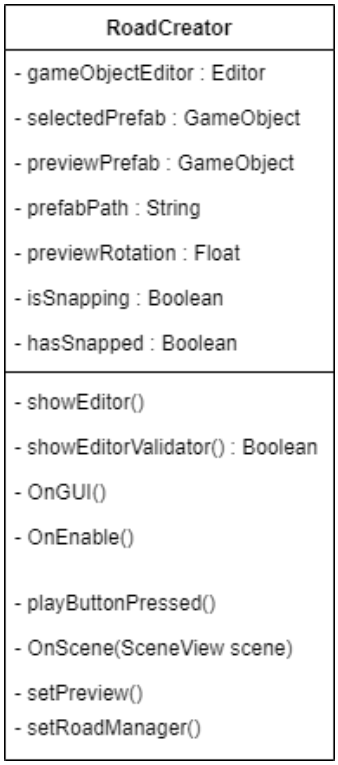


Figura C.4: RoadCreator UML

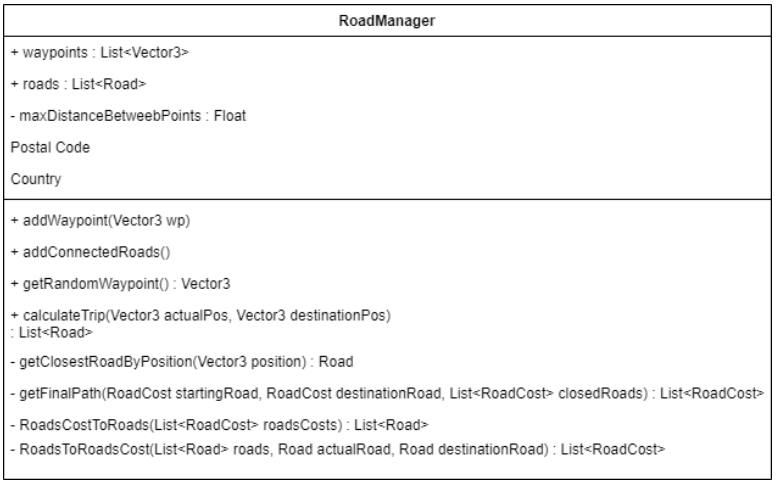


Figura C.5: RoadManager UML

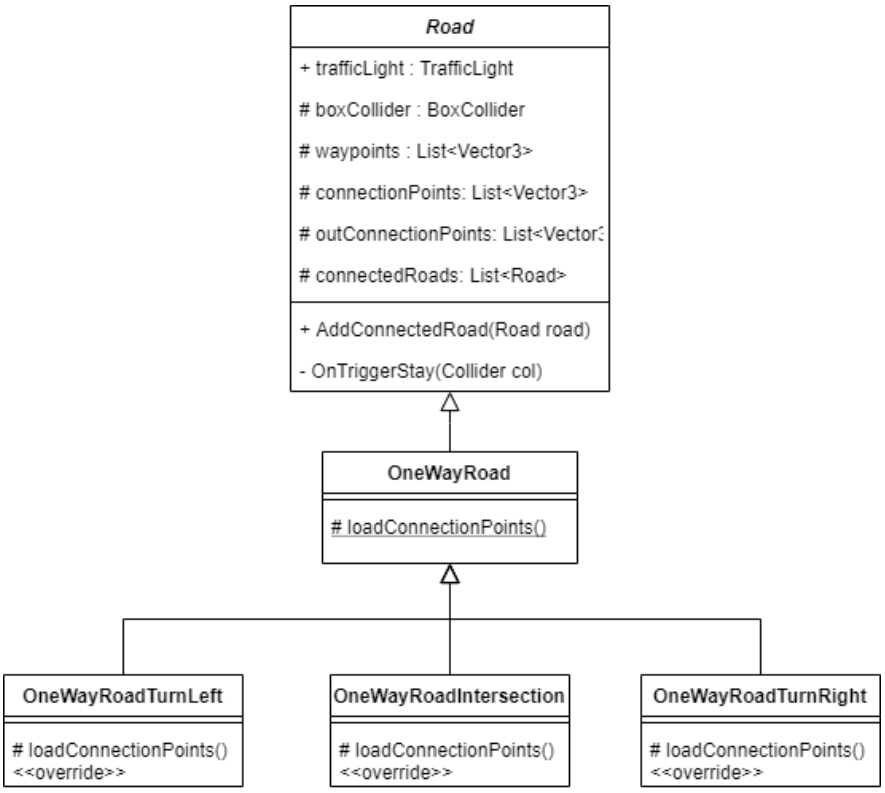


Figura C.6: Road UML

Apéndice D

Manual de usuario de Easy Road System

Este es el manual de usuario de la librería o paquete Easy Road System v1.0.
Aquí se describen los distintos componentes de la herramienta y cómo utilizarla.

D.1 Elementos de la ventana ERS

En esta sección se muestra la ventana ERS junto con una descripción de cada uno de los elementos que contiene

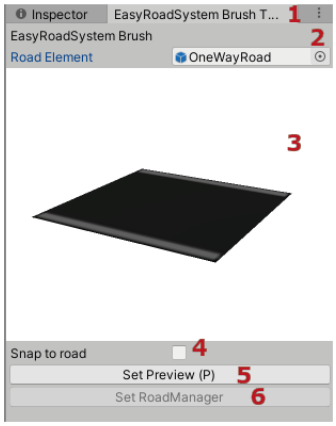


Figura D.1: Ventana ERS

1. **Pestaña ERS**: pestaña para intercambiar entre el inspector o la ventana de ERS.
2. **Road Element Picker**: Permite seleccionar un prefab de entre todos los disponibles.
3. **Vista previa del elemento seleccionado**: muestra el elemento seleccionado en el Road Element.

4. **Activación del snapping**: Activa o desactiva la función del snapping
5. **Botón 'Set Preview'**: Coloca en la escena el road element a modo de previsualización
6. **Botón 'Set RoadManager'**: Coloca en la escena, si no está ya colocado, el objeto RoadManager.

D.2 Importación de ERS

1. Abrir o crear un proyecto en Unity3D.
2. En la barra de herramientas: Assets -> Import package -> Custom Package...
3. Buscar y seleccionar el paquete 'ERS.unitypackage'. El menú de ERS se añadirá a la barra de herramientas.
4. En la barra de herramientas: EasyRoadSystem -> ShowEditor

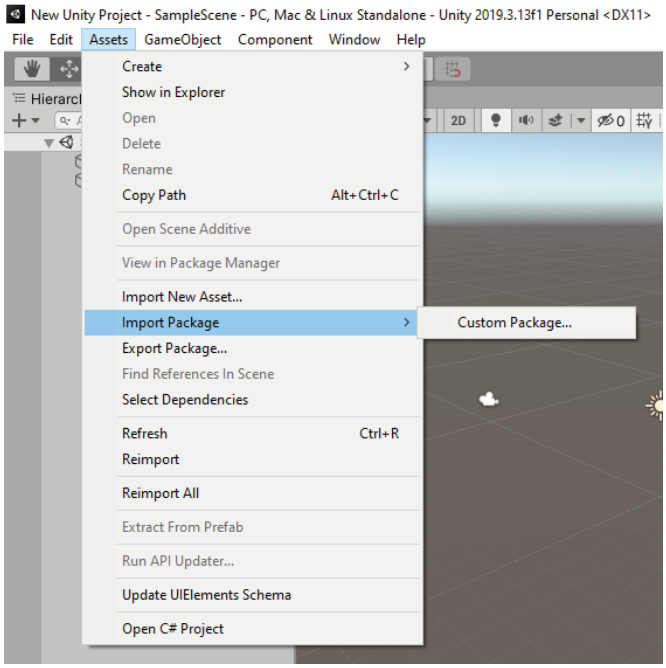


Figura D.2: Importar ERS

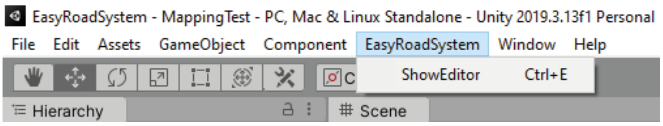


Figura D.3: Barra de herramientas ERS

D.3 Uso de ERS

Antes de hacer aplicar los pasos de esta sección, comprobar que la importación de ERS es correcta (se puede hacer observando si en la barra de herramientas de Unity3D, aparece el menú

'EasyRoadSystem'.

Colocación de la primera carretera o posicionamiento libre de elementos:

1. Desactivar el Snap to road.
2. Seleccionar un objeto en el picker 'Road Element' dentro de la ventana ERS.
3. Pulsar el botón Set RoadManager si está disponible.
4. Pulsar el botón Set Preview para visualizar el elemento a colocar.
5. Hacer click dentro de la ventana Scene de Unity.
6. Colocar la carretera con la tecla B.

Continuación de carreteras:

1. Activar el Snap to road.
2. Seleccionar un objeto en el picker 'Road Element' dentro de la ventana ERS.
3. Pulsar el botón Set RoadManager si está disponible.
4. Pulsar el botón Set Preview para visualizar el elemento a colocar.
5. Hacer click dentro de la ventana Scene de Unity.
6. Situar el cursor en uno de los bordes de una carretera ya colocada. El preview se colocará automaticamente pegado a la carretera seleccionada.
7. Colocar la carretera con la tecla B.

D.4 Snapping

El snapping es la opción que activa o desactiva el perfecto colocamiento en línea de una carretera con otra. Se activa o desactiva con la tecla 'S'.
Cuando el snapping está activado, el preview deberá de situarse en uno de los bordes de la carretera a la que se quiere adjuntar la nueva.
Si está desactivado, la colocación de la carretera será totalmente libre.

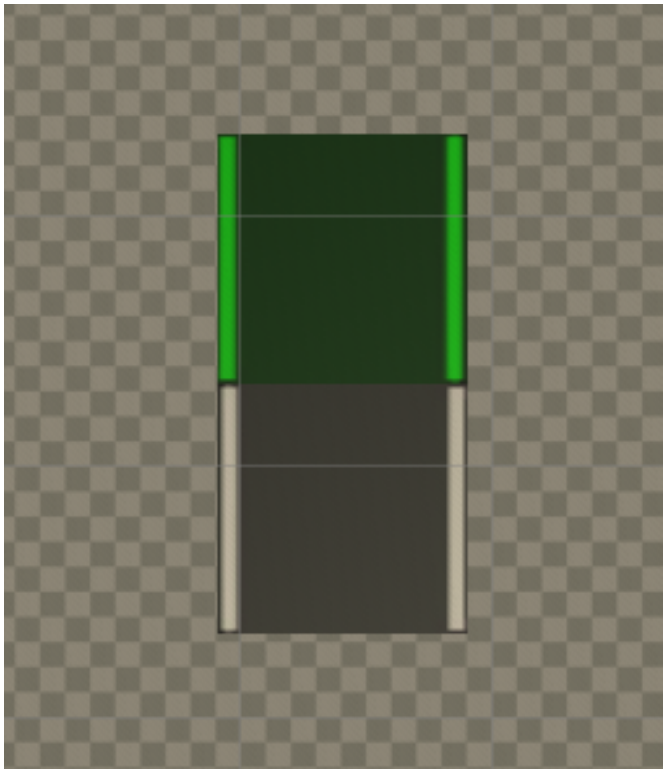


Figura D.4: Ejemplo de snapping activo

D.5 Teclas rápidas

- '**Control + E**': abre la ventana de ERS.
- '**ESC**': elimina el preview de la escena.
- '**P**': muestra el preview en la escena.
- '**R**': rota el preview 90 grados.
- '**B**': coloca el objeto seleccionado en la escena.
- '**S**': activa / desactiva el *snapping*.

D.6 Requisitos mínimos

Los requisitos mínimos para la utilización de esta librería, son los mismos que para la utilización de Unity3D:

- *SO*: Windows 7 / 8 / 10

- *Procesador*: Core 2 Duo ó superior.
- *RAM*: 1 GB de ram
- *HDD*: 50 MB de espacio disponible
- *Gráficos*: DirectX11 Compatible GPU con 512 MB Video RAM.

D.7 Requisitos recomendados

Los requisitos recomendados para la utilización de esta librería, son los siguientes:

- *SO*: Windows 10 N
- *Procesador*: Intel(R) Core(TM) i7-8750H @2.20GHz 2.21 GHz
- *RAM*: 16 GB de ram
- *SSD*: 50 MB de espacio disponible
- *Gráficos*: Nvidia GTX 1060 (6GB)

D.8 Preguntas frecuentes

- ¿Cómo roto una carretera?
+ Para rotar una carretera, bastará con tener un preview sobre la escena, y pulsar la tecla 'R'. Si el problema persiste, haz click sobre cualquier elemento de la escena y reinténtalo.
- ¿Puedo modificar una funcionalidad de ERS?
+ Sí, eres libre de modificar cualquier script para ponerlo a tu gusto, aunque recomiendo que trabajes sobre copias por si algo sale mal.
- El botón 'Set RoadManager' está desactivado, ¿qué hago?
+ Estar tranquilo, el botón RoadManager se desactivará si ya se detecta un elemento RoadManager en la jerarquía de objetos de Unity.
- No puedo colocar una carretera, ¿por qué?
+ Las carreteras solamente se pueden colocar sobre otras superficies, coloca primero un terreno o coloca una carretera arrastrándola directamente a mano desde la sección recursos.
Si el snapping está desactivado, podrás colocar una carretera sobre cualquier superficie situada debajo del ratón.
- ¿Puedo colocar mis propias texturas en la carretera?
+ Sí, solamente tienes que conseguir una nueva y cambiarla en el prefab deseado, recomiendo que trabajes sobre una copia por si algo sale mal.

- ¿Cómo añado un nuevo vehículo?
+ Para añadir un nuevo vehículo, debes añadir cuatro componentes 'WheelCollider' y configurarlo sobre cada una de las ruedas.
Acuérdate también de añadir un componente BasicCar para hacerlo funcionar y linkear cada una de las ruedas con sus respectivas en el script (ERS sólo soporta vehículos de cuatro ruedas por el momento).

D.9 Contacto

Para cualquier problema, no dudes en contactar conmigo a través de uno de los siguientes canales:

- **e-mail**: sergio.fernandez.barreiro@udc.es
- **discord**: serfeba#2123

Lista de acrónimos

4WD *Four-Wheel Drive*

4x4 *Cuatro por cuatro*

A* *A estrella.*

AWD *All-Wheel Drive*

BFS *Breadth-First Search*

DGT *Dirección General Tráfico.*

DFS *Deep-First Search*

ERS *Easy Road System.*

GUI *Intefaz de Usuario Gráfica*

IA *Inteligencia Artificial.*

OOP *Programación Orientada a Objetos*

TD *Tracción delantera*

TT *Tracción trasera*

UML *Lenguaje Unificado Modelado*

Glosario

Escena Lugar virtual donde se desarrollan los hechos.

Torque Fuerza aplicada sobre un objeto la cual lo hace girar.

Semáforo Dispositivo lumínico encargado de regular el tráfico.

Snapping Acción de colocar un elemento junto a otro sin dejar espacio entre ambos.

Unity3D Motor gráfico gratuito utilizado para este proyecto.

Vial Relativo a las vías, carreteras, vehículos...

Bibliografía

[1] S. Software. (2012) Euro truck simulator 2. [Online]. Available: $https://store.steampowered.com/app/227300/Euro_Truck_Simulator_2/$

[2] Unity. (2005) Unity (motor gráfico). [En línea]. Disponible en: $https://www.unity.com/es)$

[3] E. Abellán. (2020) Metodología scrum: qué es y cómo funciona). [En línea]. Disponible en: $https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html)$

[4] Ryte, "Modelo en espiral." [En línea]. Disponible en: $https://es.ryte.com/wiki/Modelo_en_Espirall$

[5] K.-L. Du and M. N. S. Swamy, *Search and Optimization by Metaheuristics*. Basilea, Suiza: BirkHäuser, 2016.

[6] Google, "$algoritmosdebusqueda$." [En línea]. Disponible en: $https://code.google.com/archive/p/algoritmosbusqueda/downloads$

[7] BOE, "$codigodecirculacionvial$." [En línea]. Disponible en: $https://www.boe.es/biblioteca_juridica/codigos/abrir_pdf.php?fich=020_Codigo_de_Trafico_.y_Seguridad_Vial.pdf$

[8] DGT, "Informacion semaforos." [En línea]. Disponible en: $http://www.dgt.es/Galerias/la-dgt/empleo-publico/oposiciones/doc/2013/TEMA_24_GESTION_TECNICA_TRAFICO.docl$

[9] Camaro ss coupe. [En línea]. Disponible en: $https://free3d.com/es/modelo-3d/chevrolet-camaro-ss-coupe-373476.html$

[10] J. Muñoz, "Tracción del vehículo: que es y qué tipos de sistema de traccion existen." [En línea]. Disponible en: $https://www.autofact.cl/blog/mi-auto/conduccion/traccion-auto$