Desarrollo de un simulador de conducción para la formación de pilotos de carreras

Trabajo de Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA

VNiVERSiDAD D SALAMANCA

Septiembre de 2022

Autor
Javier Mateos del Amo

Tutor
Roberto Therón Sánchez

Dr. Roberto Therón Sánchez, profesor adscrito al Departamento de Informática y Automática de la Universidad de Salamanca.

Hace constar:

Que el trabajo titulado “Desarrollo de un simulador de conducción para la formación de pilotos de carreras” ha sido realizado por D. Javier Mateos del Amo, con DNI 70832988A y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado de la Titulación Grado en Ingeniería Informática de esta Universidad.

Y para que así conste a todos los efectos oportunos.

THERON SANCHEZ ROBERTO - 07976246F

Digitally signed by THERON SANCHEZ ROBERTO - 07976246F
Date: 2022.09.07 12:41:53 +02'00'

En Salamanca, a miércoles, 7 de septiembre de 2022

Fdo. Roberto Therón Sánchez

Resumen

El trabajo expuesto en la siguiente memoria describe el diseño e implementación de un simulador de conducción para el entrenamiento de ingenieros y pilotos de carreras.

Mediante el simulador, los pilotos de carreras pueden rodar sobre un trazado, haciendo uso de un vehículo en concreto. A su vez, uno o múltiples ingenieros pueden realizar distintos ajustes al reglaje del coche, ver los diversos datos de telemetría del vehículo, ajustar la meteorología y momento del día de la sesión, visualizar mediante video la sesión, y provocar eventos de carrera, como distintos tipos de banderas.

Haciendo uso del simulador, se pueden obtener múltiples resultados, algunos de los cuales se describen a continuación: Aprendizaje del circuito por parte del piloto, tanto en la secuencia de curvas y rectas que lo componen, como en conducir lo más rápido posible sobre él; pruebas de distintos ajustes al reglaje del vehículo, y la observación de los efectos de estos cambios mediante la telemetría; ensayo de situaciones de carrera en conjunción con el piloto; conducción bajo varios momentos del día y condiciones meteorológicas, etc.

El simulador está compuesto por dos partes principales: El simulador 3D propiamente dicho, en el cual se representa el circuito y el coche, mediante el cual el piloto puede rodar; y el panel de control web, mediante el cual el o los ingenieros pueden realizar los distintos ajustes y visualizaciones descritas anteriormente. Ambas partes se encuentran orientadas a dispositivos de escritorio.

El simulador 3D, implementado mediante el uso del motor gráfico *Unreal Engine 5*, hace uso de una serie de técnicas y tecnologías que lo diferencian de gran parte de las soluciones comerciales ya existentes. Estas son: *Realidad Virtual* (*VR*, por sus siglas en inglés*)* para la visualización de la simulación por parte del piloto, y datos *DTM (Digital Terrain Model),* sumado a imágenes satelitales, para la creación del circuito. Las diferencias, ventajas e inconvenientes respecto a técnicas y tecnologías tradiciones se discuten a lo largo de la memoria. Sumado a esto, el piloto controla el vehículo de la simulación mediante un dispositivo hardware: un volante con levas de cambio y un juego de 3 pedales (acelerador, freno y embrague).

El panel de control web, implementado mediante el framework *React*, permite el trabajo simultaneo y sincronizado de varios ingenieros. Para la interacción con el simulador 3D, se ha hecho uso de comunicación en tiempo real mediante la tecnología de *WebSockets*, a través de la biblioteca *Sockets.io*. Para la transmisión de video se ha hecho uso del protocolo *WebRTC*, el cual también asienta sus bases sobre *WebSockets.*

Palabras clave: simulador, deporte de motor, carreras, piloto, ingeniero, telemetría, tiempo real, visualización, Unreal Engine

Abstract

The work presented in the following report describes the design and implementation of a driving simulator for the training of engineers and racing drivers.

By means of the simulator, racing drivers can drive on a track, using a specific vehicle. In turn, one or multiple engineers can make various adjustments to the car set-up, view various vehicle telemetry data, adjust the weather and time of day of the session, view video of the session, and trigger race events, such as different types of flags.

By using the simulator, multiple results can be obtained, some of which are described below: learning the circuit by the driver, both in the sequence of curves and straights that compose it, and in driving as fast as possible on it; testing different adjustments to the vehicle setting, and observing the effects of these changes through the telemetry; testing race situations in conjunction with the driver; driving under various times of the day and weather conditions, etc.

The simulator is composed of two main parts: The 3D simulator itself, in which the circuit and the car are represented, through which the driver can drive; and the web control panel, through which the engineer(s) can perform the various adjustments and visualizations described above. Both parts are oriented to desktop devices.

The 3D simulator, implemented using Unreal Engine 5 graphics engine, makes use of a series of techniques and technologies that differentiate it from most of the existing commercial solutions. These are: Virtual Reality (VR) for the visualization of the simulation by the driver, and DTM (Digital Terrain Model) data, added to satellite images, for the creation of the circuit. The differences, advantages and disadvantages with respect to traditional techniques and technologies are discussed throughout the report. In addition, the driver controls the simulation vehicle by means of a hardware device: a steering wheel with gearshift paddles and a set of 3 pedals (accelerator, brake and clutch).

The web control panel, implemented using the React framework, allows the simultaneous and synchronized work of several engineers. For the interaction with the 3D simulator, we have made use of real-time communication using WebSockets technology, through the Sockets.io library. For video transmission, use has been made of the WebRTC protocol, which is also based on WebSockets.

Keywords: simulator, motorsport, racing, driver, engineer, telemetry, real time, visualization, Unreal Engine

Tabla de contenido

Índice de ilustraciones

Índice de tablas

1. Introducción y antecedentes

1.1. Introducción al proyecto

El ámbito de las simulaciones ha cobrado una gran importancia en las últimas décadas. Se espera que el crecimiento del mercado de aquí a 2030 sea de un 13,5% por año, pasando este de su valor de 13.025 millones de dólares a fecha de 2021, a 39.074 millones para 2030 (Grand View Research, 2021)

Una simulación permite, mediante el uso de modelos numéricos que representan las características del sistema a simular, representar un sistema real en un entorno virtual a lo largo del tiempo. Las simulaciones se usan en múltiples ámbitos, como pueden ser el científico, médico, ciencias naturales, procesos de ingeniería, educación, militar, etc.

Estas se han aprovechado del avance tecnológico en los últimos años, de manera que son capaces de simular sistemas más complejos y de manera más precisa, lo que mediante medios tradicionales resultaría inviable.

El bajo coste humano y económico, sumado a la posibilidad de practicar bajo un entorno controlado, parametrizable y sin riesgos, ha posibilitado remplazar todo tipo de procedimientos comúnmente realizados en entornos reales por simulaciones especificas del ámbito del cual se dispone a obtener una respuesta.

Dentro del mundo de las simulaciones, podemos encontrarnos con diversos tipos. Existen las simulaciones de eventos discretos, estocásticas, deterministas, híbridas, etc. El proyecto descrito en este documento entra dentro de la categoría de simulaciones físicas.

Las simulaciones físicas son aquellas en las que se representan objetos físicos reales. Dentro de estas, podemos encontrar una subcategoría más, las simulaciones interactivas o HIL *(Human-in-the-loop)*, en las cuales intervienen seres humanos, categoría en la cual se ve enmarcado este proyecto. Este tipo de simulaciones interactivas son aplicadas en múltiples ámbitos: simuladores aéreos, marítimos… y en este caso, en el ámbito automovilístico, en concreto en el deportivo.

El deporte de motor, o por su nombre en inglés, *motorsport*, es un deporte en el que se hace una gran cantidad de pruebas. Bien de ingeniería (pruebas sobre materiales o componentes diseñados), estadísticas (elección de mejor estrategia) y físicas (conducción del vehículo). Estas pruebas se han ido trasformando con el paso de los años de un formato completamente real a uno digital, mediante el uso de las simulaciones. Esto es debido a varios factores: Menor coste, mayor rapidez de interacción, mejora en computación, menor riesgo, y uno especifico de este ámbito, limitación de pruebas reales mediante el reglamento deportivo de las competiciones, en concreto, en las simulaciones físicas.

Tiempo atrás la norma dentro del mundo era realizar las pruebas de conducción del vehículo en un circuito real haciendo uso del coche real. Debido a la falta de regulación en este aspecto, los equipos, algunos de ellos hasta con circuito propio, rodaban varias veces al mes durante varios días con múltiples vehículos para conseguir todo tipo de datos, realizar pruebas de nuevos componentes, probar reglajes del coche, etc. Debido a los elevados costes y la incapacidad de los equipos de menor presupuesto de realizar prácticas como esta, surgió la necesidad de regular estas pruebas, de manera que la diferencia entre equipos con diferentes presupuestos se redujese.

Es aquí donde los equipos, ante la imposibilidad de poder realizar pruebas en el mundo real, decidieron buscar otras alternativas, como la simulación. Las pruebas ahora se realizarían mediante un simulador, donde el coche y el circuito son virtuales, reduciendo bruscamente así los costes humanos y capitales de operación, desplazamiento, etc. Ayudado por los avances en computación, estas simulaciones pueden ofrecer resultados equiparables a los reales para pruebas de nuevos componentes, reglajes al vehículo, etc. Todo esto cumpliendo las nuevas reglas que se habían establecido.

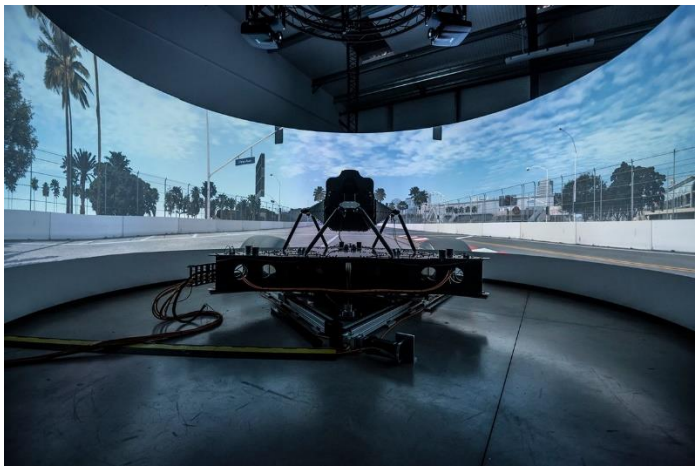


Ilustración 1. Simulador profesional de conducción

Actualmente, todos los equipos de las altas categorías del motorsport poseen este tipo de simuladores. La mayoría de ellos son soluciones propietarias, muy complejas, y con un alto coste y tiempo de desarrollo, como la visible en la Ilustración 1. Estas soluciones utilizan de una gran cantidad de datos, poder computacional, y hacen uso técnicas de alto coste para poder obtener los resultados buscados. Son estas características, la que hacen que, para los equipos de más bajas categorías, los cuales disponen de menor presupuesto, les sea inviable poder desarrollar o hacer uso de estas simulaciones. Es aquí donde se enmarca el enfoque de este proyecto.

La idea del proyecto desarrollado se basa en desarrollar un simulador 3D más asequible que las soluciones de mayor coste, haciendo uso de técnicas y tecnologías diferentes, las cuales puedan aportar resultados satisfactorios. De esta manera, se puede cubrir esa necesidad de los equipos más pequeños de tener acceso a herramientas de esta índole. Dado que la Realidad Virtual (*Virtual Reality*, VR) es una de las tecnologías que tiene un papel más relevante en el proyecto (de la cual se hablará más adelante), y al ámbito donde se va a usar el producto, se decide dar al proyecto el siguiente nombre comercial: **VRacing,** que se usará de aquí en adelante**.**

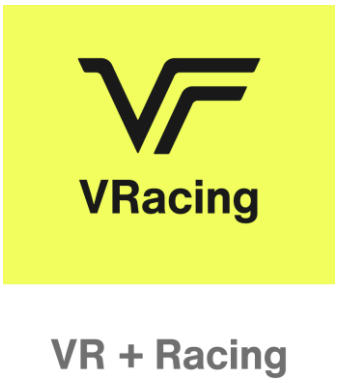


Ilustración 2. Logotipo del proyecto

VRacing es un simulador 3D de conducción de competición cuyo propósito principal es servir como entorno de pruebas y entrenamiento a pilotos e ingenieros de equipos de motorsport, el cual hace uso de técnicas y tecnologías distintas e innovadoras a las normalmente aplicadas en la industria.

Para ello, se ha planteado la arquitectura del simulador dividida en dos partes:

1) El simulador 3D propiamente dicho, mediante el cual interactúa el piloto, haciendo uso de un dispositivo hardware (volante y pedales) y unas gafas de realidad virtual;
2) Un panel de control web, mediante el cual los ingenieros pueden realizar cambios al reglaje del vehículo, visualizar la telemetría, visualizar señal de video, entre otros, sobre la sesión en curso.

1.2. Organización del documento

El presente documento detalla el proceso completo de investigación, diseño, desarrollo, resultados y conclusiones del producto descrito.

El proceso empieza realizando planteamiento de los objetivos que se desean conseguir, tanto a nivel de proyecto como a nivel personal. Esto es seguido por una breve introducción a los conceptos teóricos necesarios para la total comprensión del aspecto técnico del proyecto, mayoritariamente del área del *motorsport*. A continuación, se detallan las técnicas y herramientas que se han usado a lo largo del desarrollo de este.

Una vez asentadas las bases sobre las que se desarrolla el proyecto, se procede a explicar los aspectos más relevantes del desarrollo de este. A esto le siguen los resultados obtenidos a lo largo del desarrollo y las conclusiones a las que se ha llegado, sumado a las líneas futuras de mejora y ampliación del mismo. Para finalizar, se compilan todos los recursos consultados a lo largo del desarrollo del proyecto y de esta memoria, los cuales han sido de vital importancia para la obtención de este.

Adicionalmente, se incluyen una serie de anexos, por si el lector desea profundizar más en alguno de los temas tratados. Estos son los siguientes:

- **Anexo I: Plan del proyecto software**
  En este Anexo se incluye toda la información respecto a la estimación del esfuerzo y la planificación temporal del proyecto, representada en un diagrama de *Gantt*.

- **Anexo II: Especificación de requisitos**
  En este Anexo, se analizan los objetivos que se desean alcanzar en el desarrollo del proyecto, y partiendo de ellos, se identifican los actores que participan en el uso del sistema, sus requisitos no funcionales y de información y, por último, los funcionales, los cuales dan pie al artefacto de diagrama de casos de uso

- **Anexo III: Análisis y Diseño del sistema**
  En este Anexo, tomando como punto inicial lo alcanzado en el apartado anterior, se procede al análisis y diseño del sistema, obteniendo en el proceso los múltiples artefactos que generan

- **Anexo IV: Manual del programador**
  En este Anexo se incluye toda información de relevancia para el programador, como la estructura de directorios y ficheros, los pasos necesarios para la compilación del proyecto en su totalidad y una descripción de las clases del código fuente.

- **Anexo V: Manual de usuario**
  En este Anexo se incluye toda la información de relevancia de cara al uso del producto por parte del usuario final. Trata desde como ejecutar el proyecto a una descripción de cada pantalla de este, entre otros.

2. Objetivos

El principal objetivo del proyecto planteado es la capacidad de entrenar y servir como escenario de pruebas tanto a pilotos como a ingenieros. Para conseguir este resultado, se desglosa el proyecto en los siguientes objetivos a conseguir. Estos se ven clasificados en los siguientes tipos: Técnicos y personales

2.1. Objetivos técnicos

Los objetivos a nivel técnico son los siguientes:

- **Telemetría del vehículo:** El panel de control permitirá al ingeniero la visualización de la telemetría que el simulador 3D está generando en tiempo real. Esta estará compuesta por diversos canales, como, por ejemplo: velocidad, posición del acelerador, posición del freno, revoluciones del motor, etc.

- **Modificaciones al reglaje del vehículo:** El ingeniero, desde el panel de control web, podrá hacer una serie de cambios al reglaje del vehículo, como pueden ser modificaciones a la suspensión, motor, aerodinámica, etc. Estos tendrán un efecto sobre el comportamiento o rendimiento del vehículo en la simulación 3D.

- **Múltiples condiciones atmosféricas:** El simulador contará con distintas condiciones atmosféricas. Se hará hincapié en la lluvia y el viento, debido al impacto que ambos parámetros tienen en el mundo de la competición. La condición atmosférica que se representa en la sesión será seleccionada por el ingeniero mediante el panel de control

- **Ciclo día-noche:** El simulador contará con un ciclo completo de día-noche. El panel de control permitirá al ingeniero seleccionar distintos momentos del día y fechas de calendario, con una simulación de la posición solar realista basada en la latitud y longitud del circuito modelado.

- **Medición de los tiempos de la sesión:** El ingeniero podrá visualizar desde el panel de control los tiempos marcados por el piloto en el entorno 3D, bien en vueltas, sectores y mini sectores de cada vuelta.

- **Visualización de señal de video**: El ingeniero podrá visualizar desde el panel de control la sesión actual del simulador desde varias perspectivas en formato de video.

- **Gestión de eventos de carrera:** El ingeniero, desde el panel de control, podrá activar distintos eventos relacionados con situaciones de carrera, como banderas amarillas, banderas rojas, Safety Car (SC), etc. También este podrá ver si el piloto incumple ciertas normas en el simulador 3D, como superar los límites legales de pista.

- **Contenido del simulador modularizable:** Se busca conseguir que el simulador esté basado en una estructura modularizada, de manera que se puedan importar nuevos circuitos y vehículos en un futuro, sin necesidad de tener que modificar este de raíz.

- **Realidad Virtual:** Implementación del simulador en la tecnología de visualización VR (Realidad Virtual)

- **Implementación de vehículo y circuitos reales:** Se hará una implementación de vehículos y circuitos existentes en la vida real en el simulador. Para ello se harán uso de: en el caso del vehículo, documentación y otros recursos disponibles para la obtención de los datos necesarios para su correcta simulación; y, en el caso del circuito, modelos de altura DTM (*Digital Terrain Model*) de la máxima resolución disponible sumado a imágenes satelitales de la zona para una correcta modelización del perfil y características de la pista

2.2. Objetivos personales

Los objetivos personales que persigue el proyecto y su desarrollo son los siguientes:

- **Adquisición de nuevos conocimientos:** Se busca la adquisición de nuevos conocimientos, tanto sobre técnicas, herramientas y lenguajes de programación, desde un aspecto más orientado a la profesión de ingeniero informático, como la adquisición de nuevos conocimientos sobre dinámica de vehículos y simulación, desde la perspectiva del *motorsport*.

- **Aplicación de conocimientos existentes:** Se busca aplicar y combinar todos los conocimientos aprendidos a lo largo del grado en un proyecto de mayor envergadura, tanto en tareas de desarrollo como en tareas de planificación de proyecto.

- **Orientación al mercado:** Como añadido al desarrollo académico de este proyecto, se plantea este proyecto como una posible solución comercial viable. Como aliciente a la consecución de esta idea, este proyecto ha sido seleccionado en la convocatoria de "Prototipos Orientados al Mercado", ayuda al emprendimiento, tanto en formación como económica, ofrecida por el plan TCUE 2021-2023, organismo de la Fundación General de la Universidad de Salamanca.

3. Conceptos teóricos

En el siguiente apartado se procede a explicar los conceptos teóricos, principalmente del aspecto del *motorsport*, de manera que se asiente una sólida base en los aspectos teóricos específicos de este campo de manera que luego se disponga de toda la información disponible para comprender la implementación del proyecto.

El orden de explicación de éstos parte de una vista a alto nivel, empezando por tópicos como qué es el *motorsport*, hasta los detalles más a bajo nivel, como características de este que se ven reflejadas en el simulador desarrollado.

3.1. Motorsport

El deporte de motor, o por su nombre en inglés, *motorsport*, es una rama del deporte competitivo en la cual se hace uso de vehículos propulsados por un motor como principal característica.

El *motorsport* engloba varias disciplinas, entre las que destacan el automovilismo y motociclismo, en las que, comúnmente, se usan vehículos de cuatro en la primera y de dos ruedas en la última, respectivamente. El proyecto desarrollado se encuentra enfocado en la disciplina del automovilismo.

El principal evento de este deporte son las carreras, cuyo ganador es el que es capaz de completar el número de vueltas establecido en el mínimo tiempo posible, a lo largo de un trazado designado, haciendo uso de su vehículo. Por ello, los participantes, que, dado la alta carga de trabajo, suelen ser equipos de múltiples integrantes, centran todos sus esfuerzos en conseguir este objetivo.

Alcanzar el objetivo de ganar, conlleva una mezcla humana y técnica. Tanto es importante la conducción del piloto el día de la prueba, como la preparación y trabajo previo, presente y futuro sobre el vehículo del equipo técnico que lo acompaña. Desde el aspecto puramente técnico, conseguir el máximo rendimiento del vehículo sin normativa alguna que restrinja las características de este, sería una tarea “sencilla”.

Por ello, todos estos campeonatos se ven regulados bajo cierta normativa, comúnmente técnica y deportiva, sancionada por organismos internacionales reconocidos: en el caso del automovilismo, este trabajo recae sobre la FIA (Federación Internacional de Automovilismo). De esta manera, los equipos técnicos de cada equipo deben diseñar y optimizar su vehículo dentro de un reglamento estricto.

Por este motivo, los equipos buscan la manera de optimizar al máximo su vehículo y el rendimiento de su piloto con todas las herramientas que estén a su disposición, para obtener el máximo rendimiento dentro de la reglamentación vigente. Es aquí donde entra la herramienta sobre la que trata este trabajo, los simuladores.

3.2. Simulaciones fuera y dentro del motorsport

Una simulación, según la definición formal, consiste en la imitación de un sistema real mediante el uso de modelos numéricos (TWI Global, 2022). Como resultado, se obtiene un producto el cual puede ser usado en múltiples ámbitos: Toma de decisiones, mejora de la seguridad, entrenamiento de personal, etc. Estas además son aplicadas en multitud de áreas: ingeniería, medicina, militar, sector del automóvil, entre otras.

En el mundo del *motorsport*, las simulaciones son una herramienta clave. Estas son usadas para todo tipo de propósitos con fin de optimizar el vehículo, el piloto, e incluso, el evento donde se compite, para así lograr el objetivo de la victoria. Desde la simulación de estrés en los componentes, buscando la mejor combinación de cantidad y tipo de material que ofrezca el resultado buscado, a simulaciones de estrategia durante la carrera en curso para encontrar la estrategia optima a seguir, simulaciones aerodinámicas, simulaciones de tiempo por vuelta, etc.

Sin embargo, la simulación más frecuente en la industria, y a su vez la que más ayuda a la mejora de equipo y piloto, son las simulaciones de conducción, o simuladores *DIL* (*Driver-in-the-loop*). Este tipo de simulaciones presentan como objetivo principal el trabajo conjunto de pilotos e ingenieros (nombre que se les da a los miembros técnicos del equipo encargados de la búsqueda del rendimiento óptimo del vehículo) de manera que ambos en conjunto saquen el mayor rendimiento al coche, desde un punto de vista de pilotaje y de ajustes.

En este tipo de simuladores, el piloto, puede conducir el vehículo designado bajo un trazado especifico, de manera que pueda, tanto aprenderse el mismo, como probar los distintos ajustes en el vehículo y ver con cuál es más rápido o se siente más cómodo. Mientras, el ingeniero, puede realizar ajustes al vehículo, de manera que puedan buscar la configuración optima que les aporte mayor velocidad. Además, el ingeniero cuenta con la información necesaria para poder medir esta diferencia de velocidad, y de los datos de telemetría que arroja el vehículo para poder buscar zonas del reglaje del vehículo donde sea necesaria una optimización.

3.3. Telemetría

La telemetría, por pura definición, consiste en la medición de magnitudes físicas de forma remota y la transmisión de estas mediciones para su visualización.

En el mundo del *motorsport*, estas mediciones se realizan sobre infinidad de magnitudes del vehículo haciendo uso de múltiples sensores, y varias veces por segundo (Reese, 2022). Algunas de estas magnitudes son: La posición del acelerador, posición del freno, velocidad, revoluciones del motor, etc. A veces es tal la cantidad de información, que no toda puede ser transmitida en tiempo real, y se debe almacenar en el vehículo para su posterior extracción y tratamiento. La información capturada, puede ser visualizada por los ingenieros del equipo y almacenada para posterior análisis.

Los datos capturados pueden servir al equipo de múltiples formas: Observar el estado actual del vehículo, de manera que se puedan detectar algún fallo, análisis posterior del rendimiento y comportamiento del vehículo, y donde se encuentran las posibles zonas de mejora, etc.

3.4. Reglaje

El reglaje del vehículo, o *setup* por su nombre en inglés, consiste en la configuración de varios parámetros de los subsistemas que lo conforman. Estos subsistemas pueden ser la dirección, suspensión, motor, transmisión, etc. En función de las limitaciones técnicas de cada componente y la normativa impuesta por cada competición, estos ajustes se pueden realizar en mayor o menor cantidad de áreas.

3.5. Medición de tiempos

A lo largo de un trazada de un circuito, se hace una serie de medición de tiempos para determinar el rendimiento de cada vehículo. Estas mediciones, no se realizan en un solo punto, si no que se realizan en varios. En función de la categoría, los puntos o momentos donde se hacen estas mediciones varían, si bien, en la mayoría de las competiciones, hacen uso del siguiente estándar de facto:

Se realizan tres tipos de mediciones por tiempo: Vuelta, sectores y mini sectores. La medición de tiempo de la vuelta se hace en un único punto del trazado, comúnmente llamado “línea de meta”. Con esta medición, se obtiene el tiempo, en el orden de minutos, segundos y milésimas de segundo, que el vehículo ha tardado en completar una vuelta al trazado.

Partiendo de la longitud total del trazado, los sectores surgen de la división de este en tres partes de igual longitud, normalmente tres partes. A lo largo de la pista, por tanto, se toma la medición del tiempo en estos tres puntos equidistantes. La suma de estos tres arroja como resultado el tiempo por vuelta descrito anteriormente.

Por último, contamos con los mini sectores, los cuales surgen de la división de cada sector en cuatro zonas de medición más pequeñas. Con lo que a lo largo de un trazado nos podemos encontrar con doce mini sectores.

Como resultado de la división del trazado expuesta, se obtienen diferentes niveles de granularidad en la medición de tiempos, de manera que sea posible analizar donde se está ganando o perdiendo más tiempo a lo largo de una vuelta.

3.6. Eventos de carrera

A lo largo de una carrera en *motorsport*, existen una serie de eventos que se pueden producir, los cuales alteran el normal funcionamiento de esta. Estos eventos, si bien pueden producirse por diferentes motivos, el más común es el de accidentes en los que se ven uno o múltiples coches implicados.

Dada la situación de peligro que causa un accidente en el trazado al resto de vehículos, existe una serie de medidas las cuales ayudan a neutralizar parcial o totalmente el evento, de manera que se pueda proceder al restablecimiento de la pista a su estado normal. Durante estos periodos de neutralización, los pilotos en pista deben de seguir una serie de normas, en función del tipo de la neutralización impuesta, de manera que se facilite la acción de los oficiales de pista, colectivo que se encarga de restaurar la pista a su estado normal.

Las neutralizaciones más comunes son las siguientes (Circuit de Barcelona-Catalunya, 2022):

- Bandera amarilla: Ocurre cuando hay un accidente sobre la pista, el cual, si bien requiere de intervención, se puede realizar mientras el resto de los vehículos pueden seguir rodando sobre ella. A lo largo del trazado en distintos puntos se muestra una bandera de color amarillo, y se les comunica a los pilotos, los cuales deberán de aminorar su velocidad a lo largo de la zona accidentada.

- Bandera roja: Esta se muestra cuando no es posible el rodaje del resto de vehículos durante las acciones de restauración de la pista. Esto puede ser debido a la cantidad de elementos del accidente que la invaden, a la necesidad de reparar protecciones del circuito, etc. Los pilotos, durante este periodo, deberán aminorar su velocidad y retornar inmediatamente a sus garajes hasta nuevo aviso.

- Safety Car: El coche de seguridad, o *Safety Car* (SC) por su nombre en inglés, es un vehículo el cual entra en la pista, por orden de dirección de carrera, y su labor es reagrupar a todos los participantes detrás de él, sin que estos puedan sobrepasarle ni a el mismo ni entre ellos, de manera que se consigue la ventana máxima de tiempo entre pasada y pasada del grupo de vehículos por un mismo punto del circuito, permitiendo así trabajos durante un tiempo más prolongado por parte de los oficiales de pista. Se podría considerar una situación intermedia entre la bandera amarilla y la roja.

- Bandera verde: Esta es mostrada una vez el peligro ha sido eliminado del trazado, e indica el restablecimiento de la carrera a su estado normal.

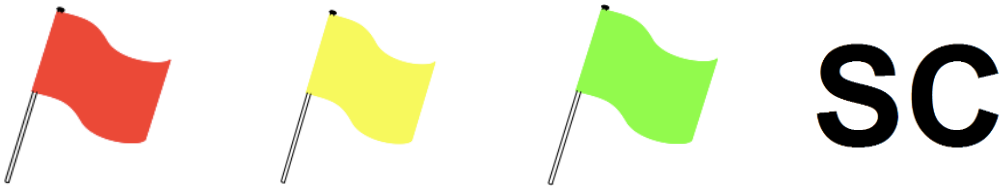


Ilustración 3. Simbología de los eventos más comunes

3.7. Digital Terrain Model (DTM)

Los *Digital Terrain Model* (DTM, por sus siglas) son una colección de datos de elevación del terreno dada una zona geográfica. Estos, son una subcategoría de los *Digital Elevation Models* (DEM), que a su vez también contienen a los *Digital Surface Model* (DSM), los cuales tienen en cuenta la altura de edificios y otras estructuras, cosa que los DTM no tienen en cuenta, solo muestran el terreno.

Este tipo de tecnologías obtienen sus datos mediante escaneos de múltiples formas haciendo uso de multitud de tecnologías, si bien la más frecuente es *Light Detection and Ranging*, o por sus siglas, LiDAR. Esta consiste en el lanzamiento del haz de un láser hacia un objeto, en este caso, desde una aeronave, y midiendo el tiempo que tarde en retornar este, calcular a qué distancia se encuentra. Esto combinado con datos de posicionamiento GPS, permite escanear una zona geográfica.

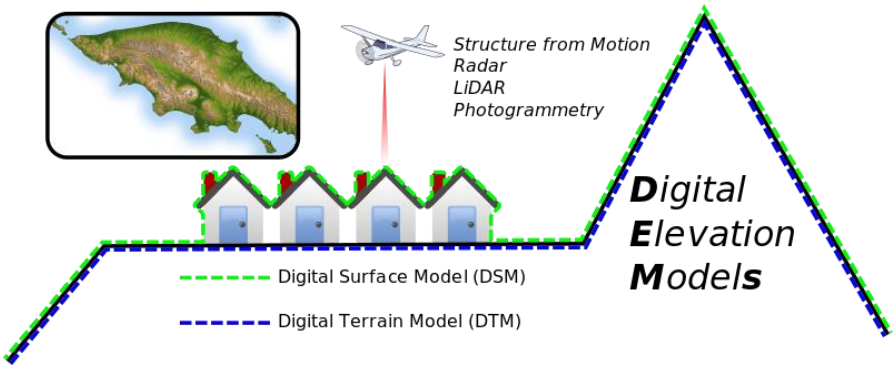


Ilustración 4. Tipos de DEM (Digital Elevation Models) comparados

Si bien existen proveedores globales de datos DTM, suele ser tarea de cada nación, o incluso a veces, de cada región, escanear y poner a disposición esta serie de datos a los usuarios, la mayoría de las veces de forma gratuita y abierta.

4. Técnicas y herramientas

A lo largo del siguiente apartado se dan a conocer las distintas técnicas, herramientas y lenguajes de programación que se han usado a lo largo del desarrollo y gestión del proyecto.

4.1. Técnicas

Las técnicas usadas a lo largo del proyecto son las siguientes:

4.1.1. Proceso Unificado

El Proceso Unificado es un marco de trabajo, comúnmente aplicado a proyectos *software*, el cual se ve caracterizado por el uso de casos de uso, están centrado en la arquitectura y ser iterativo e incremental

Debido a la naturaleza del proyecto, el ciclo de vida del Proceso Unificado puede adaptarse de manera satisfactoria a este. A continuación, se dan a conocer los argumentos que han hecho valida la elección de este marco de trabajo:

- Proyecto unipersonal: Una de las ventajas que aportan otros marcos de trabajo como Scrum, es la colaboración con otros miembros del equipo de una forma eficiente y que evite los bloqueos. Para ello, existen varias características de este que ayudan a perseguir este objetivo: Scrum Manager, Daily Meeting, etc. Este proyecto, al ser desarrollado de forma unipersonal, no aprovecharía correctamente todas las oportunidades que metodologías de este tipo ofrecen.

- Estructura modular del proyecto: El proyecto posee una estructura modular, formada por dos partes principales: Simulador 3D y Panel de control. Estos módulos hacen uso de tecnologías muy diversas, de áreas de la ingeniería informática diferentes. Por ello, la separación por iteraciones que proporciona el Proceso Unificado permite separar y focalizar todos los esfuerzos de desarrollo en un *stack* de tecnologías a la vez

El ciclo de vida iterativo e incremental del marco de trabajo se encuentra formado por una serie de etapas (Modelado de negocio, elicitación de requisitos, análisis, diseño, implementación, pruebas y despliegue), en las cuales se van obteniendo productos entregables, que, a la culminación de todas, se obtiene como resultado un producto software completo.

Como resultado, se divide el proceso en las siguientes iteraciones, de las cuales se hace una breve descripción a continuación. Estas marcaran la temporización del proyecto.

- **Iteración inicial:** En esta iteración se realiza una investigación inicial acerca del estado del arte del proyecto, soluciones existentes y sus características, etc. Seguidamente, se elabora una primera versión de los artefactos de elicitación de requisitos, análisis y diseño del sistema.

- **Iteración Simulador 3D:** Esta interacción se centra en la plataforma 3D del simulador. Se realiza una investigación sobre cómo implementar las características requeridas haciendo uso del motor gráfico escogido y las limitaciones de este. Continuadamente, se refinan los artefactos elaborados en la primera iteración del proyecto. Por último, se hace la debida implementación y pruebas de las características señaladas.

- **Iteración Panel de Control:** Esta interacción se centra en el panel de control del simulador. Se realiza una investigación sobre la información a mostrar, de qué forma mostrarla y las tecnologías disponibles para hacerlo. Seguidamente, se refinan de nuevo los artefactos de la primera interacción. Por último, se realiza la debida implementación y pruebas.

Estas interacciones a su vez fueron desglosadas en distintos hitos principales y secundarios, a los cuales, mediante el uso de un diagrama de *Gantt* y la herramienta *Microsoft Project*, se les asigna un tiempo de elaboración, dependencias y recursos.

Si se desea consultar más información acerca de la planificación realizada, se invita al lector a la revisión del Anexo I, donde se explica este proceso con mayor detalle.

4.1.2. Metodología de Durán y Bernárdez

La metodología de elicitación de requisitos de Durán y Bernárdez (Durán Toro & Bernárdez Jiménez, 2000) tiene como objetivos la definición de tareas, productos a obtener y tareas a realizar de un proyecto *software* durante la actividad de la elicitación de requisitos. Desde su nacimiento en el año 1998 se ha convertido en un estándar de facto en la industria.

La metodología propone las siguientes tareas. Estas se listan a modo de resumen:

- Obtención información del dominio del problema y sistema actual
- Preparación y realización de las reuniones de elicitación
- Identificación de los objetivos del sistema
- Identificación de los requisitos de almacenamiento de información
- Identificación de los requisitos funcionales
- Identificación de los requisitos no funcionales

Como resultado de la aplicación de esta metodología, se obtiene un documento el cual recoge todas las conclusiones y artefactos obtenidos en la elaboración de las fases anteriores. Este documento es el “Documento de Requisitos del Sistema”, o por sus siglas, *DRS*.

4.1.3. UML

UML, que por sus siglas hace referencia a *Unified Modeling Language*, es un lenguaje gráfico de modelado de productos *software* cuyo objetivo es la visualización, documentación y especificación de sistemas.

Este cuenta con varios tipos de diagramas estándar, los cuales están enfocados a distintas fases o partes del producto software desarrollado.

4.2. Herramientas

A lo largo del desarrollo del proyecto se ha hecho uso de las siguientes herramientas. Estas se clasifican en varias categorías en función de a que ámbito estén enfocado

4.2.1. Herramientas CASE

Las herramientas *Computer Aided Software Engineering*, por sus siglas en inglés CASE son aquellas herramientas enfocadas a las tareas de ingeniería *software* de un proyecto: Planificación, gestión, cálculo de costos, etc. de un proyecto *software.* En el desarrollo del proyecto se han usado las siguientes:

REM

Aplicación de escritorio desarrollada por Durán y Bernárdez la cual permite describir todos los artefactos que se pueden encontrar en el DRS, explicado anteriormente.

En el desarrollo de este proyecto, esta herramienta ha sido de utilidad para la correcta elicitación de requisitos, plasmando toda la información necesaria para estos, y guardando la trazabilidad con el resto de los componentes del DRS, como actores u objetivos.

EZ Estimate

Esta aplicación de escritorio ha sido usada para el cálculo del esfuerzo requerido, en horas por persona, de los casos de uso establecidos con anterioridad. El resultado de la aplicación de esta herramienta ha servido para establecer una planificación temporal del proyecto.

Microsoft Project

Esta aplicación de escritorio, desarrollada por Microsoft y como parte de su *suite* Office, permite la planificación temporal y gestión de proyectos. Además, esta presenta funciones que permiten la exportación de la planificación como diagrama de Gantt.

Ha sido de utilidad en este proyecto, ya que ha permitido la planificación temporal de este, con los resultados obtenidos de cálculo de esfuerzo.

Draw.io

Esta aplicación web permite el dibujo de gráficos y diagramas de múltiples tipos. Se ha usado tanto en diagramas de carácter general como en diagramas de UML. En el último caso, destaca frente a otras alternativas como *Visual Paradign* por su sencillez y facilidad de uso.

4.2.2. Herramientas de control de código

A lo largo del desarrollo del proyecto se ha hecho uso de una serie de protocolos y herramientas, las cuales han permitido tener un histórico de cambios del código, tanto en el simulador 3D como en el panel de control.

Git

El protocolo Git es un sistema distribuido de control de código. Fue creado por Linus Torvalds y su principal función es guardar un registro de los cambios que han sufrido los archivos de un proyecto, así como la creación y eliminación de estos. También permite la colaboración de múltiples personas bajo una misma base de código, detectando los posibles conflictos de edición y ofreciendo al usuario procedimientos para resolverlos.

GitHub

GitHub es una plataforma online la cual trabaja haciendo uso del protocolo Git y permite alojar repositorios de código de forma remota. Esta ha sido usada para el alojamiento, en un repositorio privado, del código del panel de control web.

GitKraken

La aplicación de escritorio GitKraken permite la interacción con el repositorio de código, bien aportando nuevos cambios o visualizando los ya realizados, de manera visual. La alternativa a las aplicaciones de tipo visual de esta índole seria realizar todo mediante línea de comandos, lo cual, bajo ciertas circunstancias, puede ser menos eficiente o más incómodo.

En este proyecto ha sido usada para la interacción con el repositorio de GitHub del panel de control web.

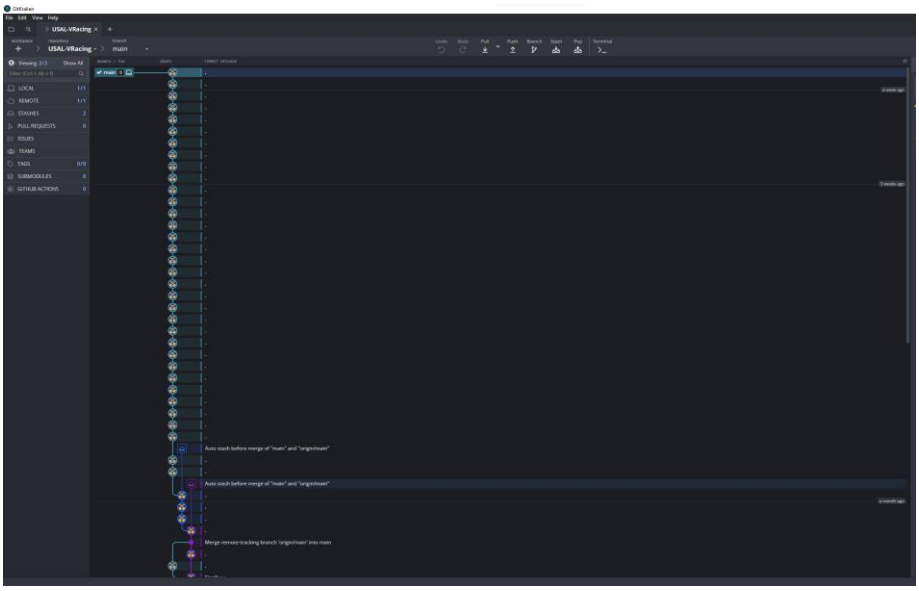


Ilustración 5. Lista de Commits subidos a GitHub vistos desde GitKraken

Helix Core

Helix Core, o comúnmente conocido en la industria de videojuegos como *Perforce*, nombre de la compañía, es un sistema de control de código, similar a *Git* pero que presenta una serie de características especiales que lo hacen más apto para trabajar con proyectos que hacen uso de motores gráficos y los archivos que estos usan. Esto es debido principalmente al control de código que realiza este sistema sobre los archivos de tipo binario.

Estos archivos, dado su formato humanamente ininteligible, no pueden ser juntados entre si cuando surge un conflicto entre dos versiones editadas de un mismo archivo, como si realizan protocolos como *Git*. Por tanto, *Perforce*, integra un sistema de bloqueo de archivos binarios (hacer *Checkout* a un archivo), de manera que estos solo puedan ser editados por una misma persona a la vez. De esta manera, se evitan los conflictos en archivos de este tipo.

También, el uso de esta tecnología se debe a que, los repositorios de plataformas como GitHub o similares, están limitados en la cantidad de datos que se pueden a almacenar, en torno a 1GB a 5GB, lo cual, para un proyecto que hace uso de un motor gráfico 3D, es demasiado poco.

En este proyecto *Perforce* ha sido usado en el desarrollo del simulador 3D, de manera que se tuviese un control de todos los cambios realizados, y se pudiese mantener la integridad de los archivos binarios, ya que el proyecto ha sido editado desde múltiples equipos del principal desarrollador.

Helix Core Server (P4D)

Helix Core Server es la aplicación, compatible con múltiples sistemas operativos, que crea y gestiona un servidor del protocolo P4 (Perforce). Este ha sido usado en el proyecto, siendo alojado en un servidor Linux (Ubuntu Server), del cual se disponía localmente en el hogar del desarrollador.

Helix Visual Client (P4V)

Helix Visual Client es el cliente visual el cual permite interaccionar con el servidor de *Perforce*, desde un punto de vista del cliente. Este se podría equiparar al *GitKraken*, pero para el entorno de *Perforce*. Ha sido usado en el proyecto para tal misión.

Este a su vez, contiene dos aplicaciones integradas: la aplicación P4Merge, la cual se encarga de mostrar de forma visual los conflictos entre archivos y ofrece al usuario varias opciones para su corrección; y la aplicación P4Admin, la cual permite administrar el servidor de P4D de forma remota y haciendo uso de una interfaz.

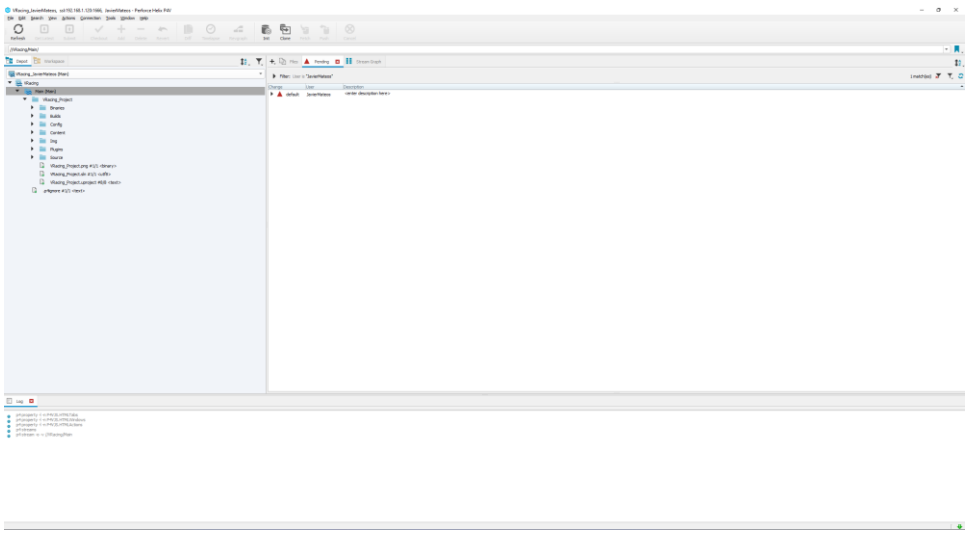


Ilustración 6. Vista del cliente de P4V

4.2.3. Herramientas de desarrollo

En la fase desarrollo del proyecto, se han usado las siguientes herramientas de depuración y programación:

Google Chrome y Dev Tools

Se ha usado el navegador Google Chrome debido a la compatibilidad que este ofrece respecto a los distintos módulos JavaScript que usan los lenguajes de programación del proyecto. Sumado a estos, sus potentes herramientas de desarrollo (*Dev Tools*) proporcionan una gran ventaja a la hora de depurar errores en el código.

React Developer Tools

Sumado a las herramientas que ofrece Google Chrome, se ha añadido en forma de extensión las *React Developer Tools*, las cuales permiten un mayor grado de depuración de aplicaciones escritas usando el *framework* ReactJS.

Visual Studio Code

Visual Studio Code es un editor de código el cual permite la edición de archivos de múltiples lenguajes de programación, aportando estas varias ayudas hacia la sintaxis, visualización y validez del código.

Este ha sido escogido frente a otras alternativas debido a la posibilidad de expandir su funcionalidad mediante el uso de extensiones. Para este proyecto se ha hecho uso de varias de las mismas, de las cuales se listan a continuación las más relevantes:

- *ES7+ React/Redux/React-Native Snippets*: Esta extensión brinda la posibilidad de inserción de código frecuentemente usado en la creación de aplicaciones que hacen uso de ReactJS.

- *JavaScript (ES6) Code Snippets*: Mismo concepto, pero para el lenguaje de programación JavaScript.

- *Prettier - Code Formatter*: Extensión la cual se encarga de dar un formato visualmente adecuado al código del proyecto.

Visual Studio

Visual Studio es un IDE (Entorno de desarrollo integrado) que permite la edición, depuración, compilación y ejecución de varios lenguajes de programación. Además, el funcionamiento de este puede ser ampliado mediante el uso de extensiones de terceros.

Se ha usado en este proyecto debido a que uno de los lenguajes soportado es C++, lenguaje que a su vez usa Unreal Engine para la compilación del propio motor como de código añadido al proyecto. Además, Unreal Engine soporta de forma nativa este IDE.

Postman

Postman es una aplicación de escritorio que permite la depuraciones y pruebas de *APIs* (*Application Programming Interfaces*, por sus siglas en inglés) y derivados. Estas pruebas se realizan en forma de peticiones de distintos tipos a los *endpoint* de la API, de manera que se pueda verificar el correcto funcionamiento de esta.

En el desarrollo del proyecto, se ha hecho uso de la funcionalidad de pruebas a *APIs* que hacen uso de la biblioteca *Socket.io*, la cual es una implementación del protocolo de *WebSockets.*

Unreal Engine 5

Unreal Engine es un motor gráfico 3D desarrollado por la compañía Epic Games. Actualmente se encuentra en su versión número 5, la cual fue publicada recientemente e incluye una serie de cambios substanciales. El motor gráfico se encuentra formado por dos módulos de alto nivel: El *runtime* del motor, o el motor gráfico propiamente dicho; y el editor, el cual es una aplicación que permite la creación de proyectos 3D que hacen uso del *runtime*.

El *runtime* es la base sobre la que se ejecuta el proyecto 3D. Este está formado por distintos módulos los cuales cumplen distintas funciones: Renderizado de gráficos, físicas, sonido, entradas y salidas, comunicación con hardware, etc. La aplicación desarrollada hace uso de múltiples módulos para su funcionamiento, asentándose como una capa superior a este *runtime.*

El editor de *Unreal Engine* proporciona una serie de herramientas visuales que permiten la utilización de todos los subsistemas del *runtime*. Estas pueden ser, por ejemplo, el editor de *blueprints*, el editor de *skeletal meshes*, de materiales e instancias de materiales, etc. Además, las funcionalidades de este pueden ser extendidas mediante el uso de extensiones o *plugins*.

Este ha sido elegido frente a otras alternativas debido a la información disponible sobre el mismo y disponer de las necesidades requeridas parala implementación del proyecto: Soporte para Realidad Virtual (VR), integración de un sistema de físicas con vehículos, extensiones para la visualización de datos geográficos 3D reales, captura de dispositivos hardware de tipo volante, etc.

4.2.4. Herramientas 3D

A continuación, se listan las herramientas del ámbito de 3D (modelaje y texturización) usadas a lo largo del desarrollo del proyecto

Blender

Blender es un software de modelado, renderizado, animación, texturizado, etc. de modelos 3D. La gran cantidad de información y guías disponibles, sumado a la gratuidad y que posee todas las funciones de las cuales se requerían, han hecho de este una solución ventajosa respeto a las otras disponibles en el mercado.

Este ha sido usado para el tratamiento y creación de los diversos modelos 3D usados en el simulador, como el coche o la pista.

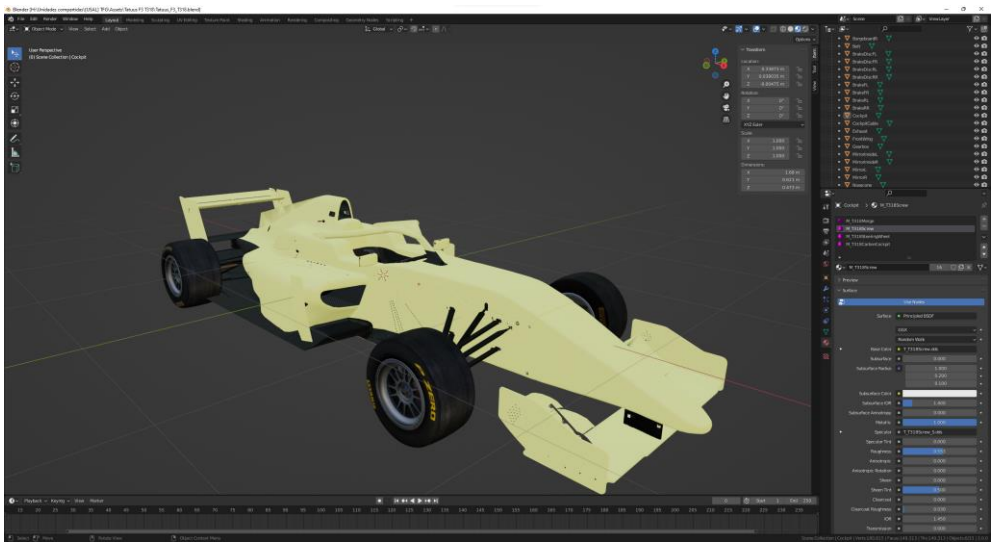


Ilustración 7. Modelo 3D del vehículo en Blender

Quixel Mixer

Quixel Mixer es una aplicación de escritorio la cual permite la texturización de modelos 3D. Esta combina diferentes texturas y propiedades físicas de estos para crear los denominados materiales, que asemejan a su contraparte de la vida real. Esta ofrece una gran biblioteca de materiales listos para su uso, lo cual ofrece una ventaja respecto a las alternativas. Además, si se hace uso de esta en conjunción con el motor gráfico Unreal Engine, tanto la herramienta como la biblioteca de materiales se obtienen a coste cero.

Esta ha sido usada en la texturización de distintos modelos 3D, como el de la pista.

4.2.5. Herramientas gráficas

En este apartado, se tratan las herramientas de naturaleza gráfica que se han usado a lo largo del desarrollo del proyecto:

Figma

Figma es una aplicación de escritorio para el prototipado, diseño y pruebas de interacción de interfaces de usuario (UI), todo enmarcado dentro del proceso de experiencia de usuario (UX). Su funcionamiento *online* y la gratuidad de este han hecho que sea una elección favorable respecto a otras soluciones.

Esta aplicación ha sido usada para el desarrollo de las diversas interfaces de usuario del proyecto, y sus pruebas con usuarios, para una vez validadas, ser implementadas. Se ha trabajado con la biblioteca de componentes disponible comercialmente de *Material UI,* biblioteca de componentes a su vez disponible para su uso con *ReactJS*.

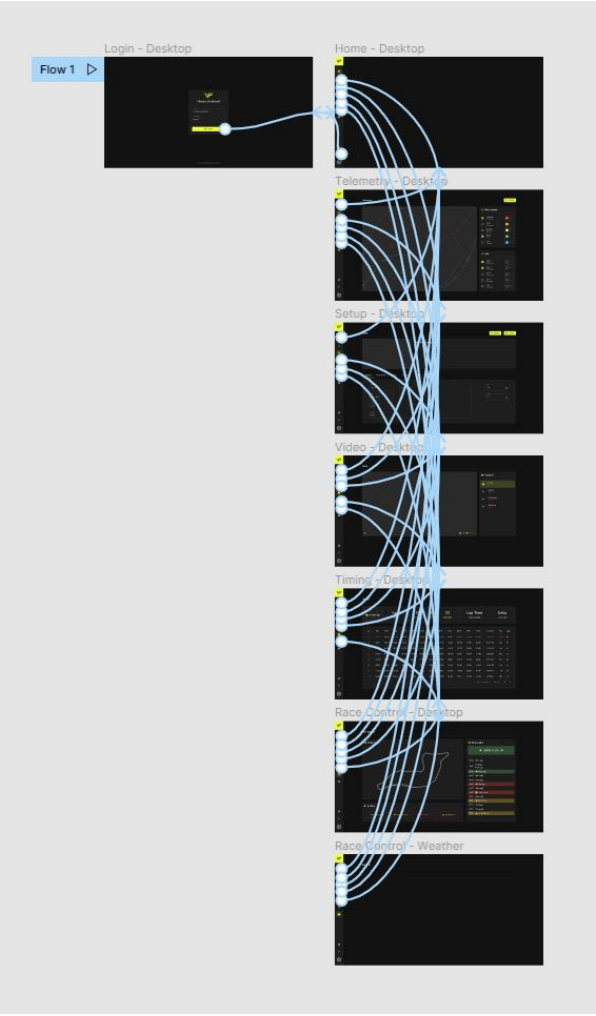


Ilustración 8. Interfaces diseñadas con flujos de interacciones

Adobe Illustrator

Adobe Illustrator es un software de edición de gráficos vectoriales enmarcado dentro del paquete de Creative Cloud de la compañía Adobe. Este permite trabajar con gráficos los cuales no están formados por valores de pixeles absolutos, si no que se calcula su representación gráfica a través de fórmulas matemáticas. Esto permite la escalación de estos gráficos a cualquier tamaño sin que ello conlleve una pérdida de calidad, entre otros.

Se ha elegido respecto a otras soluciones ya que se había usado múltiples veces anteriormente, y cumplía todos los requisitos necesarios para el trabajo a realizar. En este proyecto esta aplicación ha sido usada para la creación del logo y todo el *media kit*.

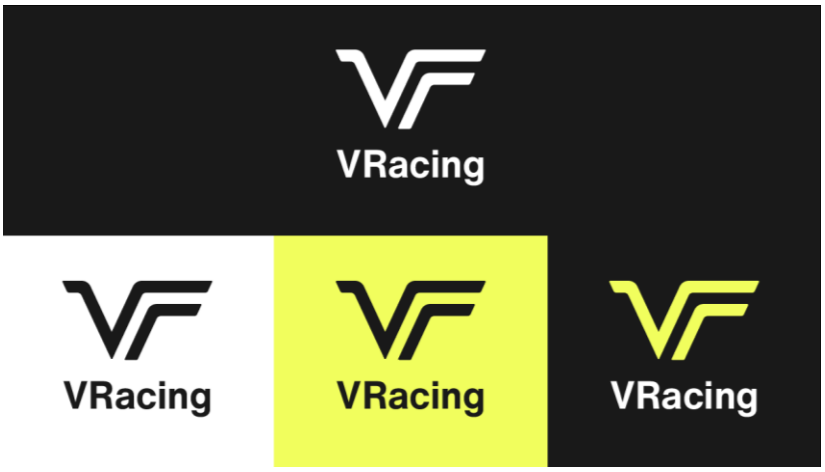


Ilustración 9. Media kit diseñado en Adobe Illustrator

Adobe Photoshop

Adobe Photoshop es un programa de edición de imágenes. Este, a diferencia de Illustrator, si hace uso de imágenes en formato de pixeles, de manera que está enfocado a propósitos y dispone de herramientas distintas.

Este ha sido usado para el retoque de imágenes, como, por ejemplo, algunas de las mostradas en este documento y las visibles en el panel de control.

4.2.6. Herramientas de tratamiento del terreno

En este apartado, se mencionan todas aquellas herramientas usadas para la modificación y transformación del terreno sobre el que se asienta el circuito

ArcGIS Pro

Este *software* GIS (*Geographical Information System*) permite el tratamiento, edición y exportación, entre otras funciones, de archivos de carácter geográfico y cartográfico. Este destaca frente a otras soluciones por su simplicidad, pero a la vez ser un *software* muy completo y con todo tipo de herramientas

Provisto bajo licencia de estudiante de la Universidad de Salamanca, esta ha sido usado para aplicar distintas transformaciones o los archivos de carácter geográfico (DTM) usados en la creación del trazado.

TerreSculptor 2.0

Este *software* permite la creación, edición y transformación de terrenos 3D enfocados a su uso en motores gráficos. Este ha sido elegido frente a otras soluciones, mayoritariamente de pago, debido a que, presenta las características que se necesitaban, todo a coste cero

Este ha sido usado para realizar las transformaciones necesarias de manera que se pueda hacer uso de los archivos resultantes de la aplicación de *ArcGIS Pro* dentro del motor gráfico *Unreal Engine*.

4.3. Lenguajes de programación y bibliotecas de código

En el proyecto se han hecho uso de una serie de lenguajes de programación y biblioteca para la obtención de la aplicación final. Estas, agrupadas en función de su lugar de aplicación, son las siguientes:

4.3.1. Simulador 3D

En el desarrollo del simulador 3D propiamente dicho, se han hecho uso de los siguientes lenguajes y bibliotecas de código, bien incluidas en la instalación del motor gráfico o externas, elementos los cuales trabajan todos en conjunción con el motor gráfico *Unreal Engine 5*

Blueprints (Unreal Engine)

Blueprints es un sistema de programación visual del motor gráfico *Unreal Engine*. Este permite la programación de sistemas y funcionalidades de una forma rápida, y con la mayoría de las características que su alternativa *C++* ofrece sin descuidar el rendimiento del código resultante.

Si bien los sistemas creados mediante el uso de esta tecnología son ejecutados en una máquina virtual durante su uso en la aplicación 3D, se puede activar la *nativización* del código de Blueprints a *C++*, haciendo que *Unreal Engine* traduzca el código de *Blueprints* a *C++* en tiempo de compilación, de manera que la perdida de rendimiento sea mínima.

Este lenguaje ha sido usado en la implementación de todos aquellos sistemas no críticos en rendimiento del simulador.

C++

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup como extensión del lenguaje C. Este se podría clasificar como un lenguaje mixto, ya que posee la capacidad de uno orientado a objetos y uno estructurado.

El lenguaje de programación C++ es, junto a *Blueprints*, los lenguajes soportados de forma oficial por el motor gráfico *Unreal Engine*. Este permite la creación de sistemas y funcionalidades, de igual manera que lo permite *Blueprints*, aunque permite llegar a aquellas clases y funciones para las que *Blueprints* no tiene acceso.

Este guarda una alta integración con el motor gráfico de *Unreal*, ya que posee una serie de *macros,* las cuales permiten integrarlo con el motor gráfico y sus otros componentes de una forma sencilla. Además, al ser un lenguaje compilado de forma nativa, su grado de optimización es mayor que el que puede aportar el sistema anteriormente descrito.

Cesium for Unreal Engine

Cesium es un proveedor de datos geográficos 3D con cobertura mundial. Este, provee varias formas de visualización de estos datos: mapas, a través de sus bibliotecas para *JavaScript*; y motores gráficos 3D mediante sus propias extensiones, compatible actualmente con los motores *Unreal Engine* y *O3DE*. Además, se le pueden insertar datos geográficos propietarios, y mostrarlos en las visualizaciones.

Mediante el uso de *Cesium for Unreal Engine* ha sido posible la importación de datos geográficos, referenciados por un sistema de coordenadas real (*WSG84*) e imágenes satelitales, los cuales han servido para el posicionamiento y referenciación del terreno de mayor resolución usado, y la pista y sus objetos colindantes.

Chaos Vehicles

Este *plugin*, incluido por defecto con el motor, aunque desactivado, proporciona una implementación de dinámica de vehículos la cual hace uso del sistema de físicas de *Unreal Engine*, *Chaos Physics*.

Por tanto, para la simulación de las físicas del vehículo se ha hecho uso de este *plugin*. Este requiere de la adaptación y configuración de los parámetros específicos de cada vehículo para su correcto funcionamiento.

OpenXR

OpenXR es un estándar para Realidad Virtual (VR) y Realidad Aumentada (AR), que en conjunto de denominan XR, que permite la interoperabilidad de distintos fabricantes y modelos de estos visores bajo una misma base de código. Unreal Engine hace uso de esta tecnología, en formato de *plugin*, para que sea posible ejecutar sus aplicaciones 3D con múltiples visores de XR sin necesidad de tener que adaptar el código de manera específica a cada uno de estos.

En el proyecto se ha hecho uso de esta librería para poder visualizar el proyecto a través de las gafas de realidad virtual *HP Reverb G2.*

Ultra Dynamic Sky

Este *plugin*, disponible comercialmente en el *Marketplace* de Unreal Engine, permite la integración de un sistema de climatología y ciclo día-noche en el motor gráfico. Posee varios *presets* de climatología con múltiples efectos visuales y físicos sobre el simulador y la posibilidad de recrear un ciclo día-noche realista en función de la latitud, longitud y zona horaria de la localización deseada.

SocketIOClient

Extensión que implementa la librería de *Socket.IO* para su uso en *Unreal Engine*. Esta permite la conexión, envío y recepción de datos a través de *WebSockets.*

En este proyecto ha sido usada para el envío y recepción de datos con el servidor, que a su vez eran comandados o mostrados en el panel de control web.

Raw Input

Extensión incluida con el motor gráfico la cual permite la integración de controles hardware de tipo *joystick* o volante.

Esta ha sido usada en el proyecto para la lectura de los distintos botones y ejes que el volante usado, el *Thrustmaster T300 Alcantara* posee.

4.3.2. Panel de control (*Frontend*) y servidor (*Backend*)

A lo largo del desarrollo del panel de control web del proyecto se han hecho uso de una serie de lenguajes de programación, *frameworks* y bibliotecas de código, de los cuales se listan los más relevantes a continuación:

JavaScript

JavaScript, por sus siglas *JS*, es un lenguaje de programación interpretado orientado a objetos y sin tipado. En una estructura clásica de despliegue de página web, este puede ser usado de múltiples formas: bien ejecutándose desde el lado del cliente (navegador web), permitiendo así la implementación de páginas webs dinámicas, base sobre las que se asientan bibliotecas como *React*; o del lado del servidor, siendo este, en conjunto con bibliotecas suplementarias, el encargado de la implementación de *APIs* y derivados.

A lo largo de este proyecto ha sido usado, tanto en el lado del cliente como en el del servidor, en conjunto con otras bibliotecas, las cuales se exponen a continuación.

ReactJS

ReactJS es una biblioteca de JavaScript diseñada para la creación de interfaces de usuarios dinámicas mediante el uso de componentes como pilar principal. Un componente permite separar partes de la interfaz de forma independiente, aislada y haciéndolas reutilizables.

La biblioteca funciona mediante el uso de JavaScript en el lado del cliente. Cada componente posee un estado, de cuya gestión se hace cargo *ReactJS*. Si este cambia, los cambios se verán reflejados de manera automática en la interfaz construida.

Ya que *ReactJS*, por defecto, solo se encarga de gestionar el estado de los componentes, es necesario el uso de bibliotecas de terceros para ampliar la funcionalidad que este provee. Estas pueden ser bibliotecas de enrutado de distintas vistas, de componentes, etc.

Esta ha sido escogida respecto a técnicas clásicas de diseño web debido a la naturalidad cambiante en tiempo real de los datos mostrados en el panel de control. Por ejemplo, los datos como la telemetría están en constante cambio, y el manejo del estado que proporciona *ReactJS* simplifica la actualización de estos datos sobre la interfaz de usuario.

También hay que destacar la elección de esta debido a otras alternativas con la misma filosofía, como *VueJS* o *AngularJS*, debido a la comunidad e información que esta biblioteca posee, sumada a la variedad de extensiones disponibles, y la simplicidad de aprendizaje y desarrollo que ofrece esta frente a las otras alternativas.

React Router

React Router es una biblioteca de código abierto que permite el enrutamiento de páginas creadas usando *ReactJS*. De esta manera, se puede mostrar al usuario diferentes páginas o secciones de la interfaz de la aplicación en función de sus acciones, como por ejemplo la navegación a través de un menú.

MUI

MUI (Anteriormente conocida como *Material UI*) es una biblioteca de componentes y utilidades para *ReactJS* la cual proporciona al desarrollador una rápida creación de componentes e interfaces. Esta implementa componentes recurrentes en el desarrollo de interfaces de múltiples tipos, como son botones, formularios, elementos de tipografía, etc.

Añadido a esto, todos los componentes de la biblioteca están diseñados e implementados entorno a la guía de *Material Design* creada por *Google*, la cual establece unas normas y recomendaciones para la fácil comprensión e interacción con la interfaz desde el punto de vista del usuario. Sin embargo, la biblioteca permite la total personalización de los componentes y utilidades que esta implementa, de manera que los desarrolladores pueden dar un aspecto único al producto desarrollado.

Chart.js

Chart.js es una biblioteca de código para el dibujado de gráficas de múltiples tipos, como de líneas o de barras. Los datos pueden ser cargados de forma dinámica y estilizados de diferentes formas. Además, la capacidad de la biblioteca puede ser ampliada mediante el uso de extensiones.

En el proyecto se ha hecho uso de la implementación compatible con *React* de esta biblioteca (*react-chartjs-2)*, además de la inclusión de una serie de extensiones para ampliar la funcionalidad de las gráficas implementadas: *Annotation,* para el *renderizado* de gráficos adicionales sobre el lienzo y *Zoom,* para dotar de ampliación y movimiento a la gráfica renderizada.

Leaflet

Esta biblioteca de código permite la inclusión de mapas cartográficos. Estos mapas son brindados por un proveedor, como *OpenStreetMap*. Además, es posible ampliar el funcionamiento de esta biblioteca mediante el uso de extensiones.

En el proyecto ha sido usada en su versión compatible con *React* (*react-leaflet*) para la visualización del mapa de la pista en el panel de control.

NodeJS

NodeJS es un entorno de ejecución usado en la capa del *Backend* o servidor el cual hace uso del lenguaje *JavaScript* y del motor V8 de Google. Comúnmente es usado para la creación y alojamiento de *APIs*. Este proporciona métodos de interactuar con peticiones del protocolo *HTTP*, bases de datos, etc.

Añadido a esto, *NodeJS* también brinda al desarrollador de un instalador de paquetes y bibliotecas, *NPM (Node Package Manager)*, usado a lo largo de este proyecto de forma intensiva.

Express

Express es un *framework* para servidores *backend,* creado para su uso en conjunto con *NodeJS,* el cual simplifica la creación de *APIs*. Este se ha convertido en un estándar de facto en la industria.

Socket.IO

Socket.IO es una biblioteca de código abierto la cual permite el intercambio de información en tiempo real entre múltiples clientes manteniendo una baja latencia y alto rendimiento. Esta hace uso del estándar de *WebSockets* y ofrece al programador una serie de métodos que simplifican la interacción con este protocolo.

Esta biblioteca es usada tanto en el *frontend* de este proyecto, a modo de cliente que envía o recibe mensajes, como en el *backend*, que se encarga de distribuir, almacenar o transformar los mensajes recibidos de los clientes.

Node-WRTC

Node-WRTC es una biblioteca de código abierto la cual implementa el código necesario para la integración de sistemas que hagan uso del protocolo *WebRTC*. Esta ha sido desarrollada para su uso en conjunto con *NodeJS*.

4.3.3. Lanzador del simulador

Se ha desarrollado una pequeña aplicación de escritorio cuya función es de lanzador del simulador, sumado a la captura de pantalla para la señal de video mostrada en el panel de control. Se listan a continuación las tecnologías relevantes usadas.

ElectronJS

ElectronJS es un *framework* el cual permite la creación de aplicaciones gráficas para escritorio haciendo uso de tecnologías de cliente y servidor originalmente orientadas a la *web*. Este *framework* ejecuta un *backend* de *NodeJS* de forma local sumado a una instancia de *Chromium*, navegador básico desarrollado por Google y que sirve de base para otros, como *Firefox* o *Chrome*, instancia la cual permite la ejecución de aplicaciones web *frontend*, que, en el caso de este proyecto, ejecuta una aplicación web basada en *ReactJS*

En el proyecto este *framework* ha sido usado para la creación del pequeño lanzador del simulador 3D, visible en la Ilustración 10. Lanzador del simulador creado con ElectronJS

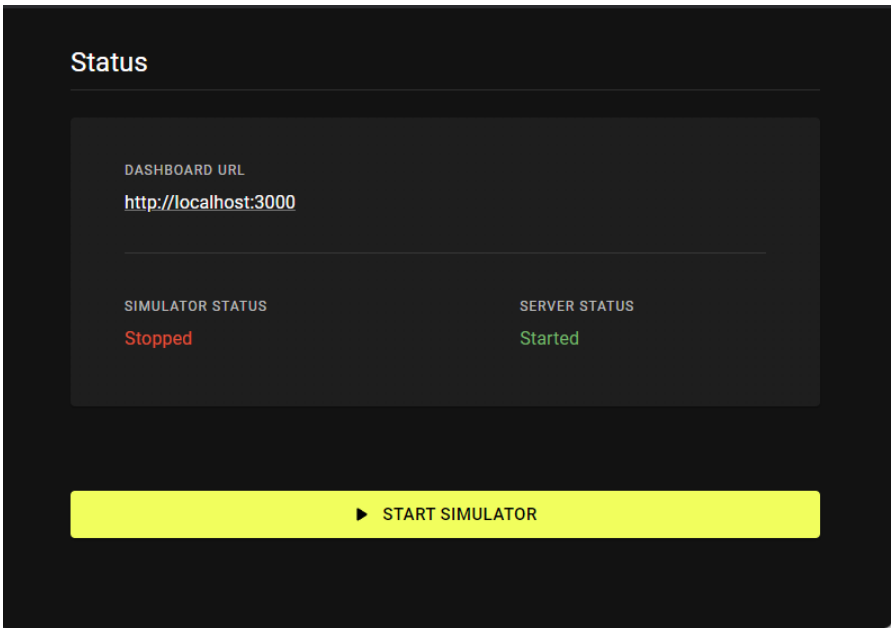


Ilustración 10. Lanzador del simulador creado con ElectronJS

5. Aspectos relevantes

En el siguiente apartado se detallan los aspectos más relevantes del desarrollo del proyecto y de su resultado final. Para ello, se estructura el siguiente punto el punto de vista de la ingeniería del software, tratando las distintas fases de las que lo componen, cuya explicación se puede encontrar en el apartado 4.1 de este documento.

5.1. Estimación del esfuerzo

Para la correcta planificación del proyecto se ha de realizar una estimación del esfuerzo que conlleva cada caso de uso de la aplicación. Para realizar esta estimación, se ha hecho uso de la métrica basada en funcionalidad "Análisis de puntos de caso de uso (UCP)". Esta métrica para productos *software* presenta una serie de cálculos los cuales arrojan como resultado final, entre otros, el número de horas necesarias para completar el proyecto.

Como ayuda a la hora de realizar estos cálculos se ha hecho uso de la herramienta *EZ Estimate*, la cual fue expuesta en el apartado anterior de herramientas utilizadas.

El cálculo de los puntos de caso de uso (UCP) viene dado por la siguiente formula:

$$UCP = UCCP \times TCF \times EF$$

Donde:

- **UCCP**: Puntos de caso de uso no ajustados
- **TCF**: Factores de complejidad técnica
- **EF**: Factores de complejidad del entorno

A su vez, el componente UCCP de esta fórmula, se puede descomponer en la siguiente formulación:

$$UCCP = UUCW + UAW$$

Donde:

- **UUCW**: Factor de peso de los casos de uso sin ajustar
- **UAW**: Factor de peso de los actores sin ajustar

El factor UAW, evalúa la complejidad de las interacciones de los actores con el sistema. Por tanto, este factor se puntúa de la siguiente forma en función de la complejidad de la interacción:

Tipo de actor	Descripción	Factor
Simple	Interacciona con el sistema mediante una *API*	1
Medio	Interacciona con el sistema mediante un protocolo (TCP/IP, por ejemplo) o de una interfaz de tipo texto	2
Complejo	Interacciona con el sistema mediante una interfaz gráfica de usuario (GUI, por sus siglas en inglés)	3

Tabla 1. Tipos de actores UAW

El cálculo final de UAW se realizaría multiplicando el factor del tipo por el número de actores de ese tipo y sumando los tres tipos.

El factor de peso de los casos de uso o UUCW se calcula de forma similar. Existen dos maneras de puntuar los factores de UUCW: Mediante número de transacciones o número de clases de análisis. En el desarrollo de este proyecto se ha hecho uso de la primera, mediante el uso de la siguiente tabla de referencia para la clasificación de los casos de uso en múltiples tipos:

Tipo de caso	Descripción	Factor
Simple	3 transacciones o menos	5
Medio	Entre 4 y 7 transacciones	10
Complejo	Más de 7 transacciones	15

Tabla 2. Tipos de casos de uso UUCW

Por tanto, como resultado de la aplicación de ambas métricas (UAW y UUCW) al proyecto, se obtienen los siguientes resultados:

	Actor	Categoría	Peso
UAW	Piloto	Complejo	3
	Ingeniero	Complejo	3
	Sistema	Medio	2
	TOTAL		**8**

Tabla 3. Cálculo de UAW

	Caso de uso	Categoría	Peso
UUCW	Ver telemetría	Simple	5
	Descargar imagen	Simple	5
	Enviar telemetría	Simple	5
	Ver reglaje	Simple	5
	Modificar reglaje	Medio	10
	Exportar reglaje	Simple	5
	Importar reglaje	Simple	5
	Ver condiciones actuales	Simple	5
	Cambiar fecha actual	Simple	5
	Cambiar hora actual	Simple	5
	Cambiar condiciones atmosféricas	Simple	5
	Iniciar simulador	Simple	5
	Conducir	Simple	5
	Ver video	Simple	5
	Enviar señal de video	Simple	5
	Ver eventos de carrera	Simple	5
	Cambiar bandera	Simple	5
	Enviar límite de pista	Simple	5
	Ver tiempos	Simple	5
	Marcar tiempo	Simple	5
	TOTAL		**105**

Tabla 4. Cálculo de UUCW

Por tanto, el resultado de la métrica UUCP obtenido es el siguiente:

$$UUCP = 105 + 8$$

$$\boldsymbol{UUCP = 113}$$

A continuación, se da los valores necesarios a los 8 distintos factores de complejidad de entorno, EF, en un rango de 0 a 5, los cuales están relacionados con la experiencia y habilidades del equipo que desarrolla el proyecto:

Factor	Nombre	Peso	Valor	Factor	Notas
E1	Familiaridad con UML	1,5	2	3	Si bien se conoce el lenguaje, se ha de mejorar los conocimientos en este
E2	Experiencia en la aplicación	0,5	3	1,5	Se requiere una serie de conocimientos de múltiples ámbitos
E3	Experiencia en orientación a objetos	1	4	4	El desarrollador lleva trabajando con lenguajes de esta índole desde hace 2 años
E4	Capacidad de los analistas	0,5	3	1,5	El único analista es el desarrollador. Capacidad media de análisis de problemas
E5	Motivación del equipo	1	5	5	Dado el ámbito del tema desarrollado, el desarrollador se encuentra altamente motivado
E6	Estabilidad de los requisitos	2	4	8	Son mayoritariamente estables
E7	Trabajadores a tiempo parcial	-1	0	0	El único desarrollador está a tiempo completo
E8	Dificultad del lenguaje de programación	-1	2	-2	Si bien es sencillo, requiere de cierto grado de aprendizaje
TOTAL				**21**	$EFactor$

Tabla 5. Cálculo de factores de complejidad de entorno

Por lo tanto, el valor EF resultante es:

$$EF = 1{,}4 + (-0{,}03 \times EFactor)$$

$$\boldsymbol{EF = 0,77}$$

Donde:

$$EFactor = \sum(Valor \times Peso)$$

Seguidamente, se hace lo propio, pero con los factores de complejidad técnica, TCF:

Factor	Nombre	Peso	Valor	Factor	Notas
F1	Sistema distribuido	2	4	8	Al ser una plataforma cliente-servidor esta es distribuida por naturaleza, más si sumamos el simulador 3D
F2	Rendimiento	1	4	3	Relevante desde un punto de vista de la Realidad Virtual
F3	Eficiencia del usuario final	1	3	3	El usuario final interacciona de una manera rápida con el producto, por tanto, la eficiencia se busca manteniendo el orden
F4	Procesamiento interno complejo	1	2	2	Maneja gran cantidad de datos y realiza una serie de procesos sobre ellos
F5	Reusabilidad	1	3	3	Es relevante la reusabilidad dada la estructura modular del proyecto
F6	Facilidad de instalación	0.5	4	2	El usuario final, si bien tiene conocimientos, no es experto en el ámbito informático
F7	Facilidad de uso	0.5	3	1,5	Requiere de cierta facilidad, aunque lo usa gente experimentada en la materia
F8	Portabilidad	2	2	4	Se usa siempre en dispositivos de escritorio.
F9	Facilidad de cambio	1	4	4	Debe de soportar modificaciones al código
F10	Concurrencia	1	4	4	Sincronización entre múltiples clientes alta
F11	Características especiales de seguridad	1	2	2	No requiere de alta seguridad
F12	Acceso directo a terceras partes	1	1	1	El sistema es bastante estanco
F13	Entrenamiento especial del usuario	1	4	4	Los usuarios necesitan un alto conocimiento del área sobre el que trabaja el sistema
TOTAL				**46**	$TFactor$

Tabla 6. Cálculo de factores de complejidad técnica

Por lo tanto, el valor TCF resultante es:

$$TCF = 0{,}6 + (0{,}01 \times TFactor)$$

$$\boldsymbol{TCF = 1{,}06}$$

Donde:

$$TFactor = \sum (Valor \times Peso)$$

Con todos los factores necesarios calculados, ahora podemos proceder al cálculo de los puntos de caso de uso, UCP:

$$UCP = 113 \times 1{,}06 \times 0{,}77$$

$$\boldsymbol{UCP = 92{,}2306}$$

Para la obtención del número de horas necesarias para completar el proyecto, TPH, debemos de multiplicar el UCP final por la constante de esfuerzo F, con valor de 20:

$$TPH = 20 \times UCP$$

$$\boldsymbol{TPH = 1844{,}612}$$

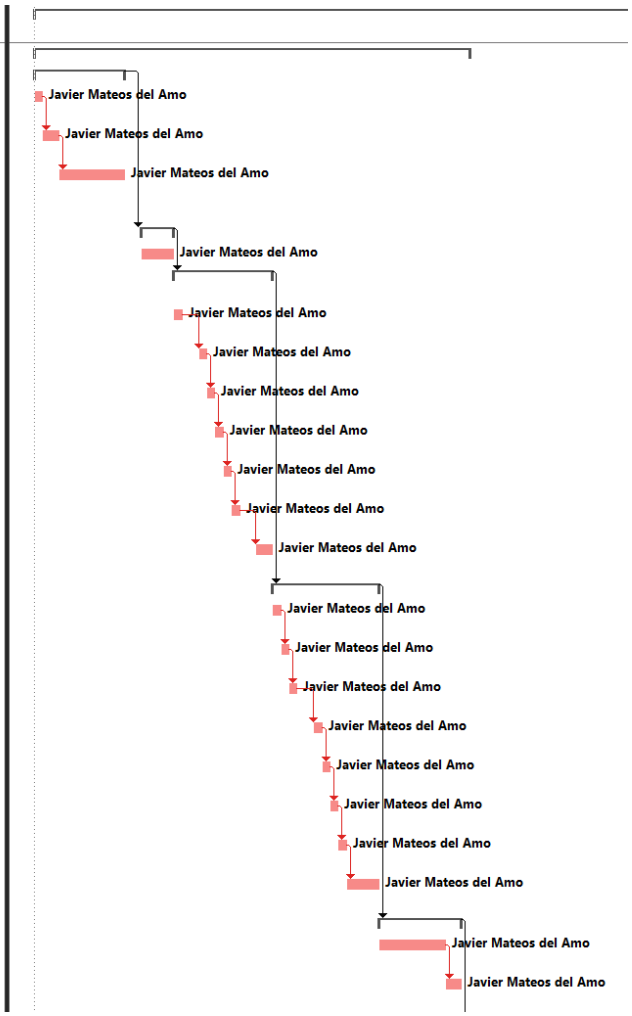
Por tanto, será necesarias **1844,612** horas o **230.6** días aproximadamente para completar el proyecto, estando formado el equipo de desarrollo por una persona a jornada completa de 8h, siguiendo el calendario de trabajo presente en el Anexo I, al cual se insta al lector a acudir si se desea más detalle.

5.2. Planificación temporal

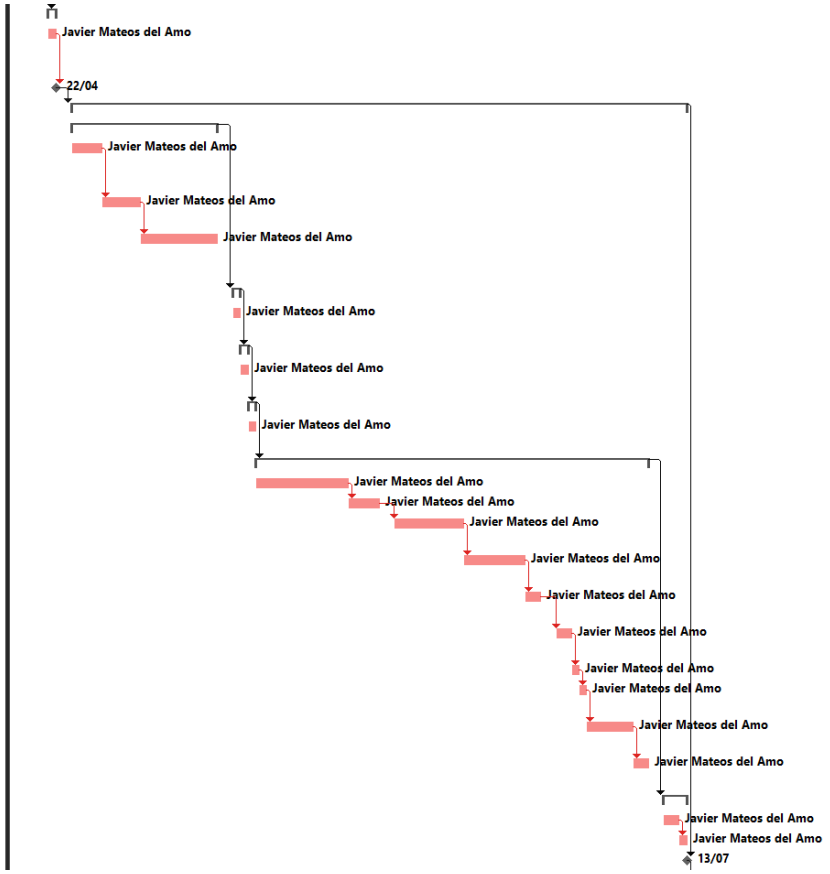
Una vez calculado el esfuerzo requerido para el desarrollo del proyecto, obtenido en el anterior apartado, se puede realizar una planificación temporal donde se indiquen las distintas fases de desarrollo del proyecto a lo largo del tiempo.

Para ello, se hace uso de un diagrama de *Gantt*, el cual permite de manera visual la representación de las tareas a realizar a lo largo del tiempo. Este se puede consultar en la Ilustración 11.

VRacing	**120 días**	**mar 01/03/22**	**lun 22/08/22**			
Iteración inicial	**37 días**	**mar 01/03/22**	**vie 22/04/22**			
Modelado de negocio	**9 días**	**mar 01/03/22**	**vie 11/03/22**			
Investigación sobre el estado del arte del problema	1 día	mar 01/03/22	mar 01/03/22		Javier Mateos del Amo	
Investigación sobre soluciones existentes	2 días	mié 02/03/22	jue 03/03/22	4	Javier Mateos del Amo	
Investigación de las tecnologias mas aptas para el desarrollo del proyecto	6 días	vie 04/03/22	vie 11/03/22	5	Javier Mateos del Amo	
Elicitación de requisitos	**4 días**	**lun 14/03/22**	**jue 17/03/22**	**3**		
Elicitación inicial	4 días	lun 14/03/22	jue 17/03/22		Javier Mateos del Amo	
Análisis	**8 días**	**vie 18/03/22**	**mar 29/03/22**	**7**		
Analisis del subsistema de telemetría del vehículo	1 día	vie 18/03/22	vie 18/03/22		Javier Mateos del Amo	
Analisis del subsistema de medición de tiempos	1 día	lun 21/03/22	lun 21/03/22	10	Javier Mateos del Amo	
Analisis del subsistema de reglajes del vehículo	1 día	mar 22/03/22	mar 22/03/22	11	Javier Mateos del Amo	
Analisis del subsistema de gestión del entorno	1 día	mié 23/03/22	mié 23/03/22	12	Javier Mateos del Amo	
Analisis del subsistema de señal de video	1 día	jue 24/03/22	jue 24/03/22	13	Javier Mateos del Amo	
Analisis del subsistema de eventos de carrera	1 día	vie 25/03/22	vie 25/03/22	14	Javier Mateos del Amo	
Analisis del subsistema de Simulador 3D	2 días	lun 28/03/22	mar 29/03/22	15	Javier Mateos del Amo	
Diseño	**9 días**	**mié 30/03/22**	**lun 11/04/22**	**9**		
Diseño de la arquitectura inicial	1 día	mié 30/03/22	mié 30/03/22		Javier Mateos del Amo	
Diseñodel subsistema de telemetría del vehículo	1 día	jue 31/03/22	jue 31/03/22	18	Javier Mateos del Amo	
Diseño del subsistema de medición de tiempos	1 día	vie 01/04/22	vie 01/04/22	19	Javier Mateos del Amo	
Diseño del subsistema de reglajes del vehículo	1 día	lun 04/04/22	lun 04/04/22	20	Javier Mateos del Amo	
Diseño del subsistema de gestión del entorno	1 día	mar 05/04/22	mar 05/04/22	21	Javier Mateos del Amo	
Diseño del subsistema de señal de video	1 día	mié 06/04/22	mié 06/04/22	22	Javier Mateos del Amo	
Diseño del subsistema de eventos de carrera	1 día	jue 07/04/22	jue 07/04/22	23	Javier Mateos del Amo	
Diseño del subsistema de Simulador 3D	2 días	vie 08/04/22	lun 11/04/22	24	Javier Mateos del Amo	
Implementación	**6 días**	**mar 12/04/22**	**jue 21/04/22**	**17**		
Preparación del entorno de desarrollo	4 días	mar 12/04/22	mar 19/04/22		Javier Mateos del Amo	
Implementación primitiva de la comunicación en tiempo real	2 días	mié 20/04/22	jue 21/04/22	27	Javier Mateos del Amo	



Pruebas	**1 día**	**vie 22/04/22**	**vie 22/04/22**	**26**	
Pruebas del sistema de comunicación primitivo de tiempo real	1 día	vie 22/04/22	vie 22/04/22		Javier Mateos del Amo
Hito Fin Iteración Inicial	0 días	vie 22/04/22	vie 22/04/22	30	
Iteración Simulador 3D	**56 días**	**lun 25/04/22**	**mié 13/07/22**	**31**	
Modelado de negocio	**14 días**	**lun 25/04/22**	**vie 13/05/22**		
Investigación de funcionamiento de Unreal Engine	4 días	lun 25/04/22	jue 28/04/22		Javier Mateos del Amo
Investigación de Chaos Vehicles de Unreal Engine	2 días	vie 29/04/22	mar 03/05/22	34	Javier Mateos del Amo
Investigación de metodos de uso de datos de terreno reales en Unreal Engine	8 días	mié 04/05/22	vie 13/05/22	35	Javier Mateos del Amo
Elicitación de requisitos	**1 día**	**lun 16/05/22**	**lun 16/05/22**	**33**	
Refinamiento de los requisitos elicitados	1 día	lun 16/05/22	lun 16/05/22		Javier Mateos del Amo
Análisis	**1 día**	**mar 17/05/22**	**mar 17/05/22**	**37**	
Refinamiento del analisis realizado	1 día	mar 17/05/22	mar 17/05/22		Javier Mateos del Amo
Diseño	**1 día**	**mié 18/05/22**	**mié 18/05/22**	**39**	
Refinamiento del diseño realizado	1 día	mié 18/05/22	mié 18/05/22		Javier Mateos del Amo
Implementación	**36 días**	**jue 19/05/22**	**vie 08/07/22**	**41**	
Implementación del circuito	8 días	jue 19/05/22	lun 30/05/22		Javier Mateos del Amo
Implementación del vehiculo	4 días	mar 31/05/22	vie 03/06/22	44	Javier Mateos del Amo
Implementación del sistema de telemetría	6 días	lun 06/06/22	mar 14/06/22	45	Javier Mateos del Amo
Implementación del sistema de reglajes	6 días	mié 15/06/22	mié 22/06/22	46	Javier Mateos del Amo
Implementación del sistema de medición de tiempos	2 días	jue 23/06/22	vie 24/06/22	47	Javier Mateos del Amo
Implementación de eventos de carrera	2 días	lun 27/06/22	mar 28/06/22	48	Javier Mateos del Amo
Implementación del ciclo día-no	1 día	mié 29/06/22	mié 29/06/22	49	Javier Mateos del Amo
Implementación de las condiciones atmosféricas	1 día	jue 30/06/22	jue 30/06/22	50	Javier Mateos del Amo
Implementación del sistema de captura de video	4 días	vie 01/07/22	mié 06/07/22	51	Javier Mateos del Amo
Implementación del lanzador del simualdor	2 días	jue 07/07/22	vie 08/07/22	52	Javier Mateos del Amo
Pruebas	**3 días**	**lun 11/07/22**	**mié 13/07/22**	**43**	
Pruebas unitarias	2 días	lun 11/07/22	mar 12/07/22		Javier Mateos del Amo
Pruebas del subsistema	1 día	mié 13/07/22	mié 13/07/22	55	Javier Mateos del Amo
Hito Fin Iteración Simulador 3D	0 días	mié 13/07/22	mié 13/07/22	32	



Iteración Panel de Control	27 días	jue 14/07/22	lun 22/08/22	57	
◢ **Modelado de negocio**	**3 días**	**jue 14/07/22**	**lun 18/07/22**		
Investigacion sobre React	1 día	jue 14/07/22	jue 14/07/22		Javier Mateos del Amo
Investigación de bibliotecas de renderizado de gráficas	1 día	vie 15/07/22	vie 15/07/22	60	Javier Mateos del Amo
Investigación sobre NodeJS y Socket.io	1 día	lun 18/07/22	lun 18/07/22	61	Javier Mateos del Amo
◢ **Elicitación de requisitos**	**1 día**	**mar 19/07/22**	**mar 19/07/22**	**59**	
Refinamiento de los requisitos elicitados	1 día	mar 19/07/22	mar 19/07/22		Javier Mateos del Amo
◢ **Análisis**	**1 día**	**mié 20/07/22**	**mié 20/07/22**	**63**	
Refinamiento del analisis realizado	1 día	mié 20/07/22	mié 20/07/22		Javier Mateos del Amo
◢ **Diseño**	**1 día**	**jue 21/07/22**	**jue 21/07/22**	**65**	
Refinamiento del diseño realizado	1 día	jue 21/07/22	jue 21/07/22		Javier Mateos del Amo
◢ **Implementación**	**18 días**	**vie 22/07/22**	**mié 17/08/22**	**67**	
Implementación del sistema de telemetría	6 días	vie 22/07/22	vie 29/07/22		Javier Mateos del Amo
Implementación del sistema de reglajes	4 días	lun 01/08/22	jue 04/08/22	70	Javier Mateos del Amo
Implementación del sistema de tiempos	2 días	vie 05/08/22	lun 08/08/22	71	Javier Mateos del Amo
Implementación de eventos de carrera	2 días	mar 09/08/22	mié 10/08/22	72	Javier Mateos del Amo
Implementación de la gestión del entorno	1 día	jue 11/08/22	jue 11/08/22	73	Javier Mateos del Amo
Implementación del sistema de reproducción de video	2 días	vie 12/08/22	mar 16/08/22	74	Javier Mateos del Amo
Otras implementaciones	1 día	mié 17/08/22	mié 17/08/22	75	Javier Mateos del Amo
◢ **Pruebas**	**3 días**	**jue 18/08/22**	**lun 22/08/22**	**69**	
Pruebas unitarias	2 días	jue 18/08/22	vie 19/08/22		Javier Mateos del Amo
Pruebas del subsistema	1 día	lun 22/08/22	lun 22/08/22	78	Javier Mateos del Amo
Hito Fin Iteración Panel de Control	0 días	lun 22/08/22	lun 22/08/22	58	

Ilustración 11. Diagrama de Gantt del proyecto (I, II y III)

5.3. Especificación de requisitos

Tomando como base la propuesta del trabajo de fin de grado y la planificación temporal realizada con anterioridad, se definieron los objetivos principales que tenía como objetivo el proyecto. Tras ello, en una primera iteración inicial, se puedo extraer de cada objetivo del sistema los múltiples requisitos que estos presentaban.

Estos requisitos han sido representados en forma de tablas haciendo uso de la metodología de Duran y Bernárdez (Durán Toro & Bernárdez Jiménez, 2000), expuesta en el apartado de "Técnicas y Herramientas". Esta metodología está compuesta de una serie de formatos de tablas para la especificación de requisitos funcionales, no funcionales, actores, etc.

A continuación, se incluyen algunos ejemplos de cada tipo artefactos que esta metodología aporta. Si se desea consultar la totalidad de estos, se invita al lector a acudir al Anexo II

OBJ-0001	**Gestión de la telemetría del vehículo**
Versión	1.0 (01/08/2022)
Autores	Javier Mateos del Amo
Fuentes	-
Descripción	El sistema deberá capturar los múltiples canales de la telemetría del vehículo de la sesión actual en tiempo real, almacenarla temporalmente para nuevos clientes, mostrarla y dar la posibilidad de guardar la misma
Subobjetivos	-
Importancia	Vital
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta
Comentarios	-

Tabla 7. Ejemplo de objetivo del sistema

ACT-0002	**Ingeniero**
Versión	1.0 (01/08/2022)
Autores	Javier Mateos del Amo
Fuentes	-
Descripción	Este actor representa al ingeniero de carreras el cual visualiza los datos, modifica los ajustes, etc. de la sesión en la cual se encuentra el piloto
Comentarios	-

Tabla 8. Ejemplo de actor del sistema

Ilustración 12. Diagrama de actores del sistema

RI-0001	Telemetría
Versión	1.0 (01/08/2022)
Autores	Javier Mateos del Amo
Fuentes	-
Objetivos asociados	• [OBJ-0001] Gestión de la telemetría del vehículo
Requisitos asociados	• [UC-0001] Ver telemetría • [UC-0002] Descargar imagen • [UC-0003] Enviar telemetría
Descripción	El sistema deberá guardar la información de la telemetría del vehículo de la sesión actual
Datos específicos	-
Intervalo temporal	Presente
Importancia	Vital
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta

Tabla 9. Ejemplo de requisito de información del sistema

UC-0006	Exportar reglaje	
Versión	1.0 (01/08/2022)	
Autores	Javier Mateos del Amo	
Fuentes	-	
Objetivos asociados	• [OBJ-0002] Gestión del reglaje del vehículo	
Requisitos asociados	• [RI-0002] Reglaje	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El actor **[ACT-0002] Ingeniero** debe haber ejecutado previamente el caso de uso **[UC-0004] Ver reglaje**
	2	El actor procede a presionar un botón de “Exportar reglaje”
	3	El actor recibe en la ruta seleccionada el reglaje en forma de archivo
Postcondición	-	
Excepciones	-	
Rendimiento	-	
Frecuencia esperada	-	
Importancia	Vital	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	-	

Tabla 10. Ejemplo de caso de uso del sistema

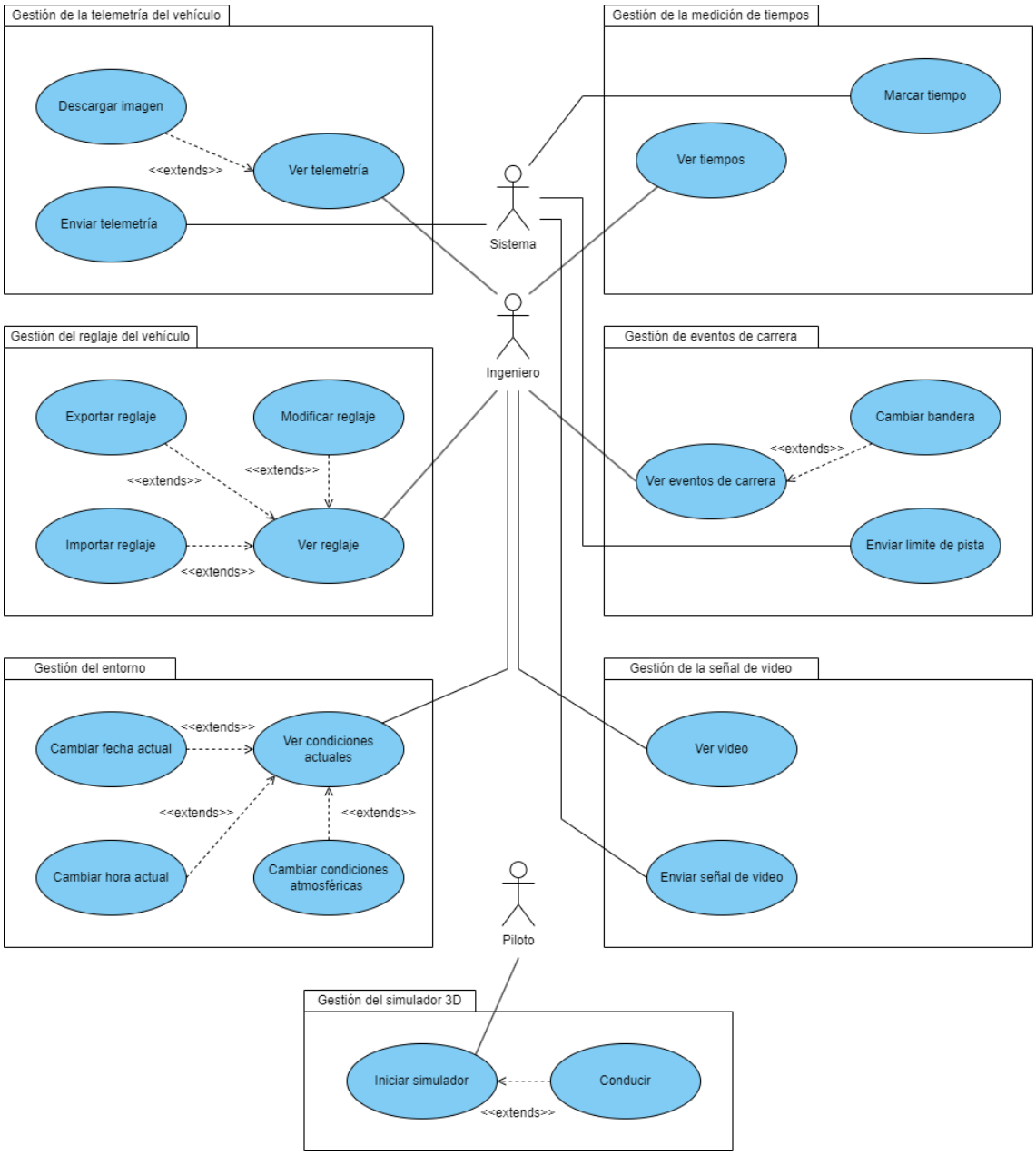


Ilustración 13. Diagrama de casos de uso

5.4. Análisis del sistema

En esta fase, una vez realizado la recogida de todos los requisitos en el anterior apartado, se realiza un análisis de estos y se obtienen unos artefactos propios del modelo de análisis, de los cuales algunas muestras son presentadas a continuación. Si el lector desea ampliar el conocimiento de este apartado, se le invita a leer el Anexo II

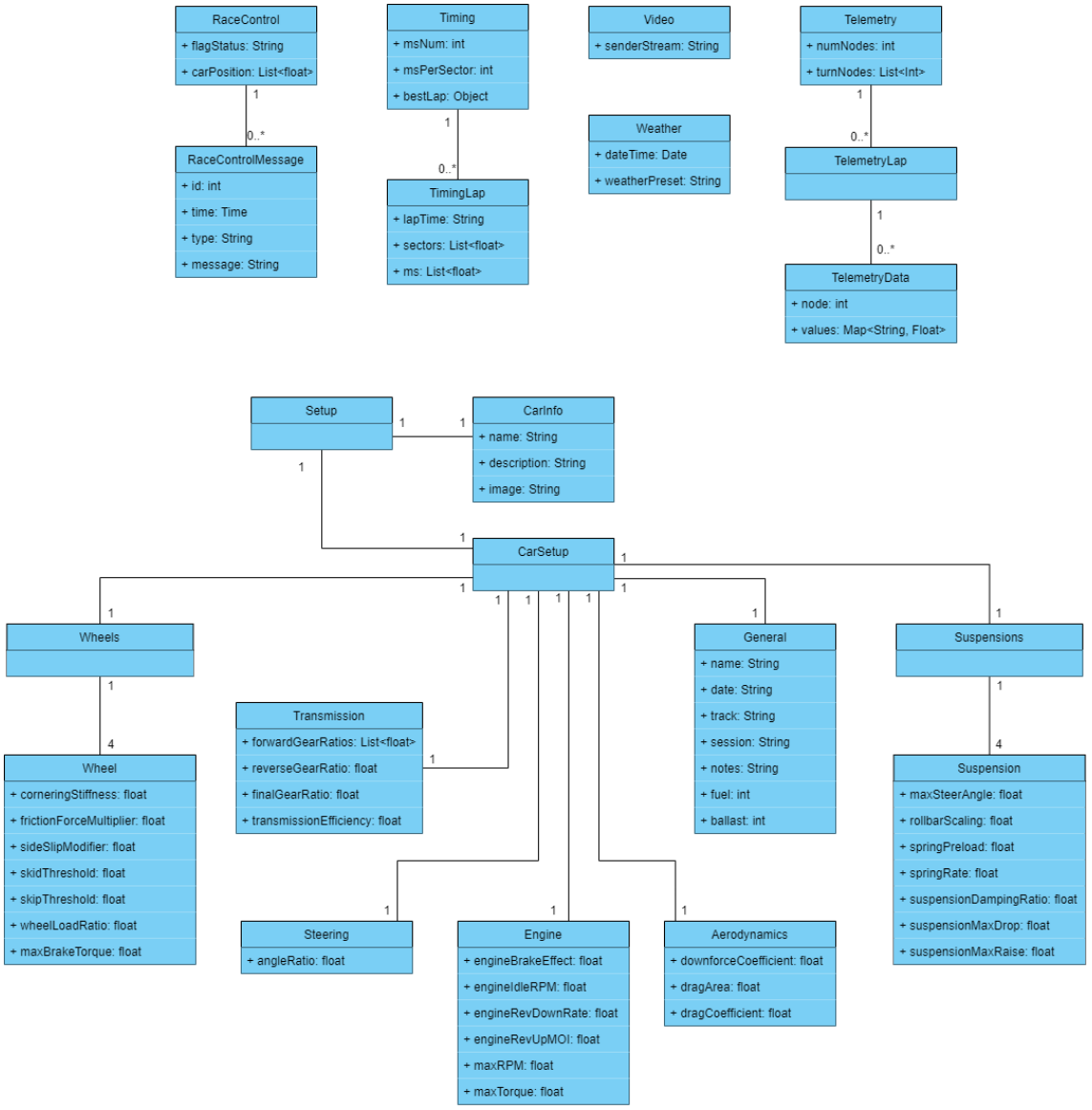


Ilustración 14. Diagrama de clases del modelo del dominio del sistema

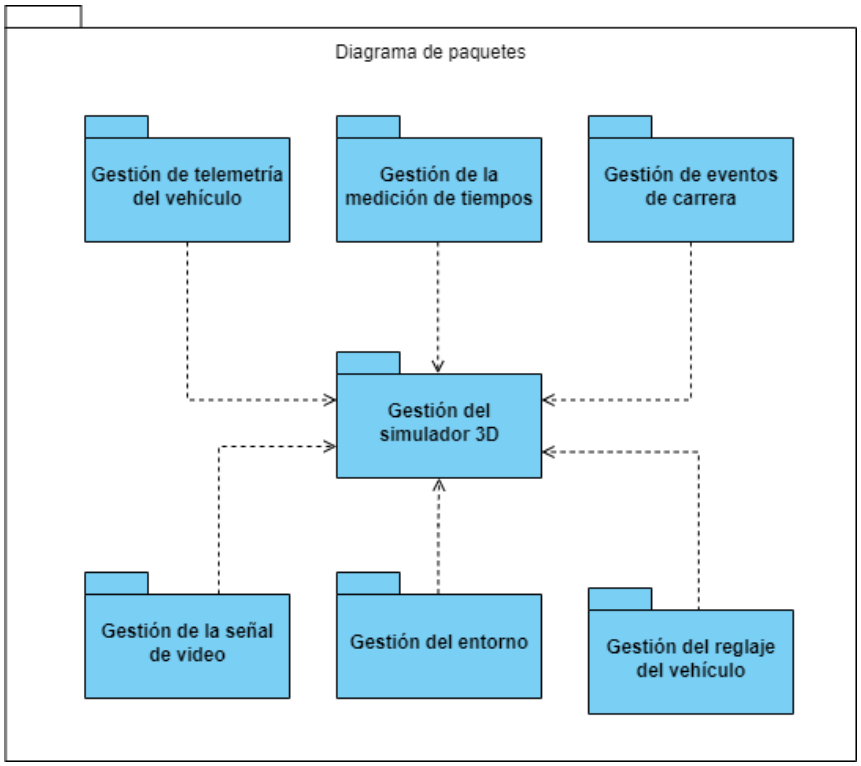


Ilustración 15. Diagrama de paquetes del sistema

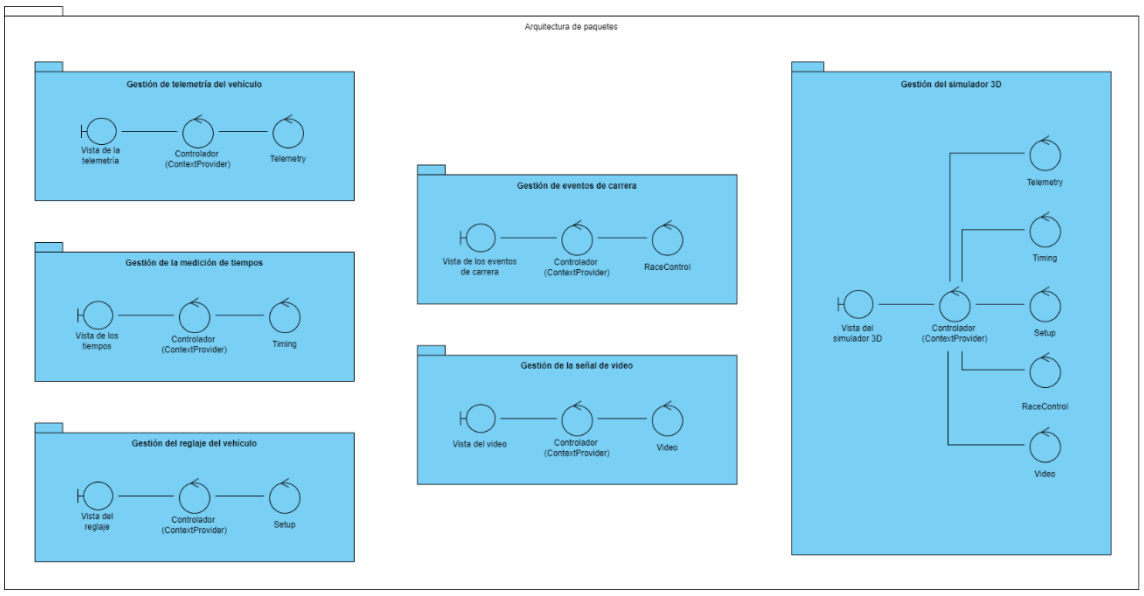


Ilustración 16. Diagrama de arquitectura de paquetes del sistema

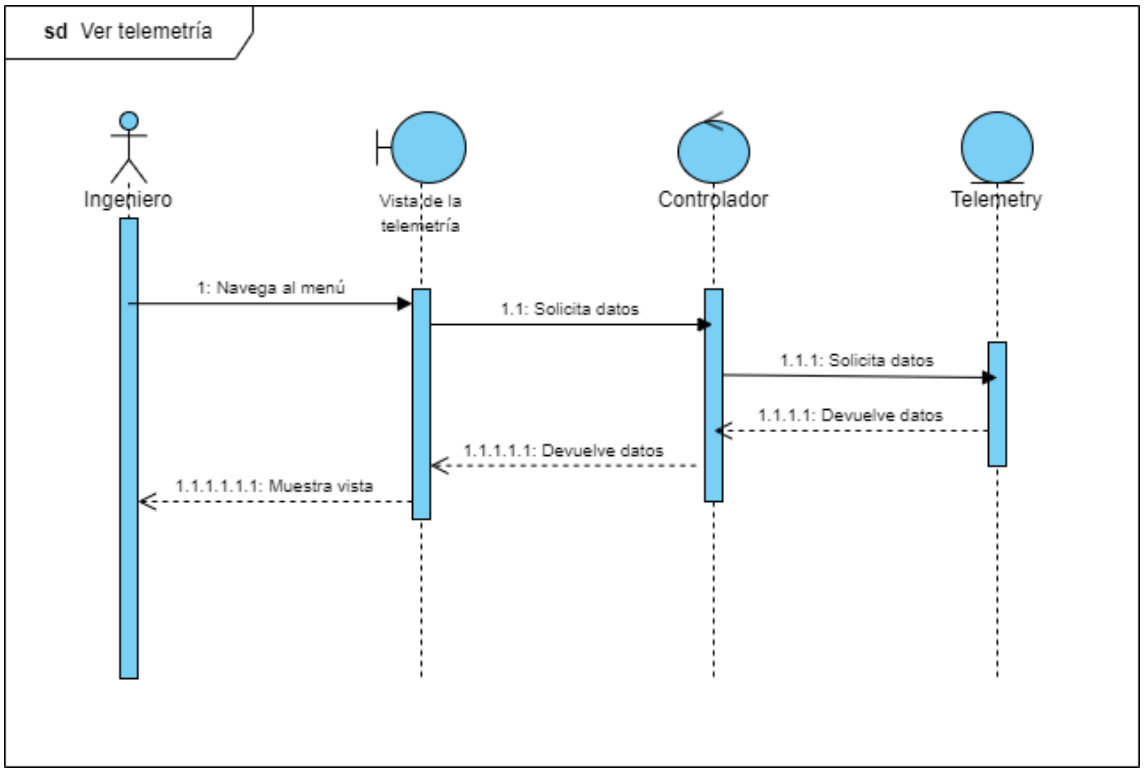


Ilustración 17. Ejemplo de diagrama de secuencia del sistema

5.5. Diseño del sistema

En esta fase, la última antes de empezar con el desarrollo de la aplicación, es donde se hace la definición formal y completa de todos los requerimientos del sistema. En el siguiente texto se tratan los puntos de mayor relevancia planteados en esta fase.

5.5.1. Patrón arquitectónico

El patrón arquitectónico elegido ha sido el patrón *Model View ViewModel*, por sus siglas, MVVM, ya que asienta una muy buena base a la aplicación, debido a la gran compatibilidad que tiene respecto a la manera a la que funciona *React.*

La descripción de los componentes de este patrón son las siguientes:

- *Model* o modelo: Representa el modelo de dominio de la aplicación, es decir, los datos. En el caso de la aplicación de *React*, este son objetos de *JavaScript* formados por distintos campos, que a su vez se encuentran contenidos en los llamados *ContextProviders.*

- *ViewModel* o Vista Modelo: Esta capa hace de intermediaria entre la vista y el modelo. Se encarga de realizar cualquier transformación necesaria, y de mantener la sincronía entre los datos almacenados en ambos

- *View* o Vista: Es la que muestra los datos del modelo al usuario. En el caso de *React* esta es reactiva, es decir, actualiza los datos que muestra automáticamente cuando los datos del modelo cambian, mediante el uso de eventos de forma interna. De esta forma, el acople entre vista y modelo es conseguido de una sencilla manera. También informa al *ViewModel* de interacciones con la vista mostrada

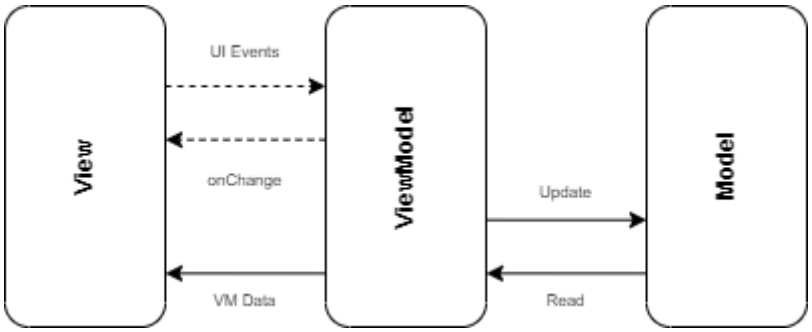


Ilustración 18. Arquitectura MVVM

5.5.2. Diagrama de despliegue

El diagrama de despliegue estructura como es el despliegue de los múltiples elementos que componen el sistema. Cada uno de los elementos que conforman este diagrama representa un componente bien de tipo *hardware* o *software*.

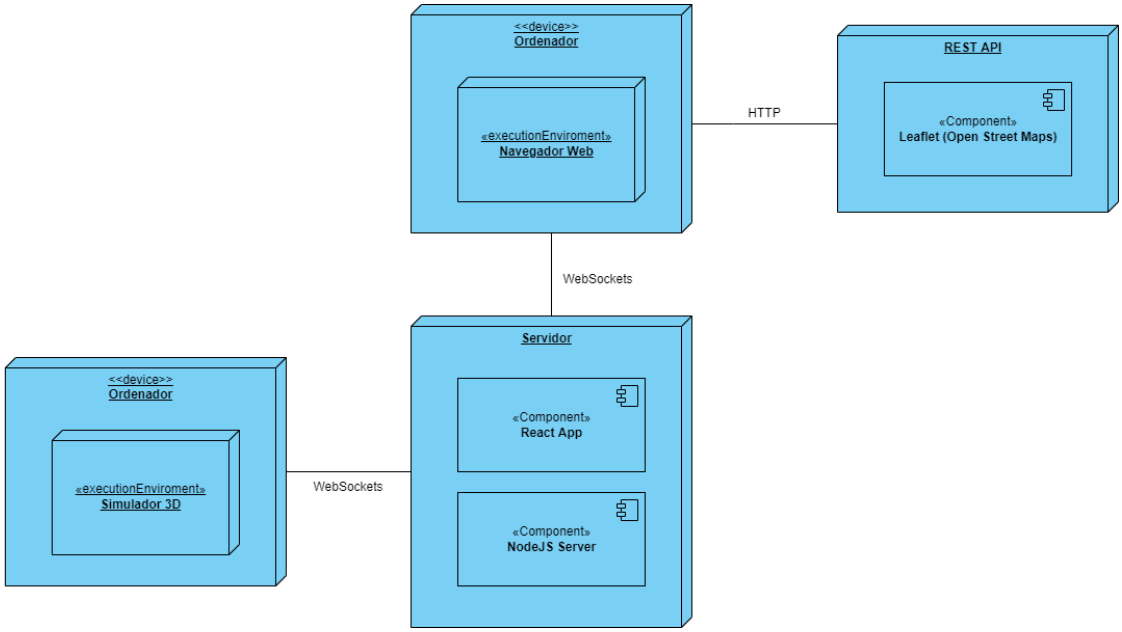


Ilustración 19. Diagrama de despliegue del sistema

5.5.3. Realización de los casos de uso

Los diagramas de secuencia de diseño representan la secuencia de pasos que se siguen al ejecutar cada caso, similar a lo realizado en la fase de análisis, aunque aún con más detalle que los diagramas presentados anteriormente. Ahora se definen las interacciones con todos los múltiples componentes del sistema, el formato de mensajes mediante el cual ejercen la comunicación, etc.

A continuación, se adjunta a modo de ejemplo un diagrama de la realización de un caso de uso. Se insta al lector a que, si desea ampliar su conocimiento en esta área, acuda al Anexo II de este proyecto

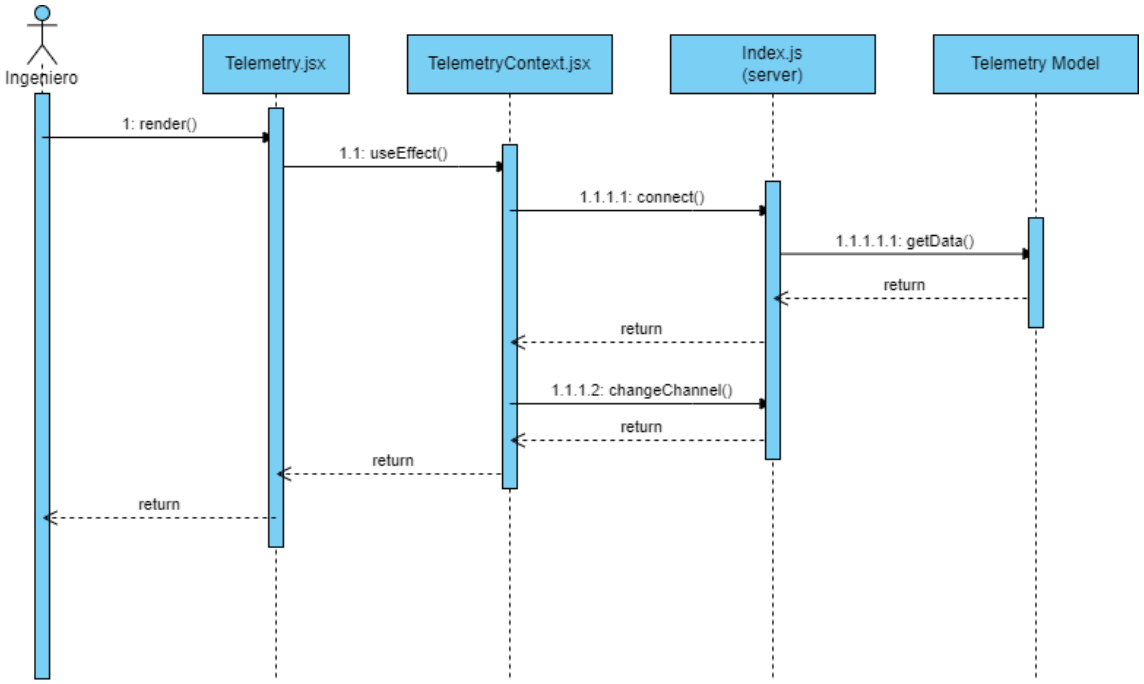


Ilustración 20. Ejemplo de diagrama de secuencia del diseño del sistema

5.5.4. Diseño de la Interfaz Gráfica (UI)

En el próximo apartado se presenta un resumen del proceso de prototipado y diseño de la interfaz gráfica de usuario de la aplicación, principalmente el panel de control web. Se irá mostrando el proceso desde el primer prototipo al diseño final, que en la fase de implementación será hecho realidad.

Inicialmente, se escogen los colores y tipografías base que van a componer la aplicación. Se decide que la aplicación este diseñada sobre un tema oscuro, el cual usa como colores de fondo y otras tonalidades cercanas al negro, de manera que resulte más agradable su visualización en ambientes de poca luz que su contraparte de tema claro, usando colores blancos.

Por otra parte, se ha escogido como color principal o primario de la aplicación uno que no pueda ser confundido con los colores estándar de facto que se suelen usar en el mundo del motorsport para indicar distintas informaciones: verde, amarillo y morado. También se han evitado colores que transmitan un mensaje común al usuario, como rojo para errores o naranja para avisos. Por tanto, el color elegido finalmente es un verde neón, el cual se presenta en la Ilustración 21Ilustración 21. Colores usados en la UI

Ilustración 21. Colores usados en la UI

En cuanto a la tipografía, se ha establecido un sistema tipográfico que enmarca distintos tipos y tamaños de textos, desde cabeceras de páginas a textos normales. Como fuente, se ha elegido una comúnmente usada en aplicaciones web, *Roboto*. A continuación, se presenta la Ilustración 22 que muestra el sistema tipográfico escogido.

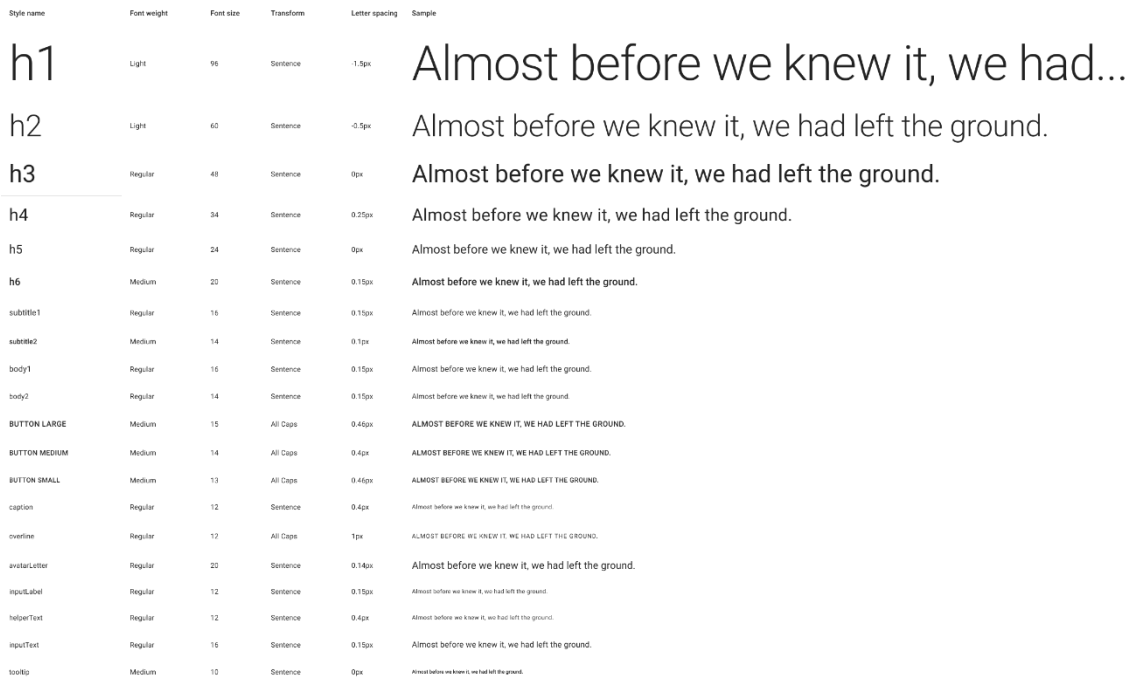


Ilustración 22. Sistema tipográfico escogido

Seguidamente, y partiendo como base de los distintos casos de uso que posee la aplicación, se realiza una primera aproximación al diseño de la interfaz mediante la creación de un prototipo de baja fidelidad o *wireframe*, por su nombre en inglés. De esta manera, se esbozan las principales características de la interfaz y su interacción, sin ser distraídos por elementos de tipo visual como colores o tipografías. A continuación, se muestra la Ilustración 23. Wireframe de la vista de Race Control, la cual contiene un ejemplo de las múltiples vistas de prototipado desarrolladas

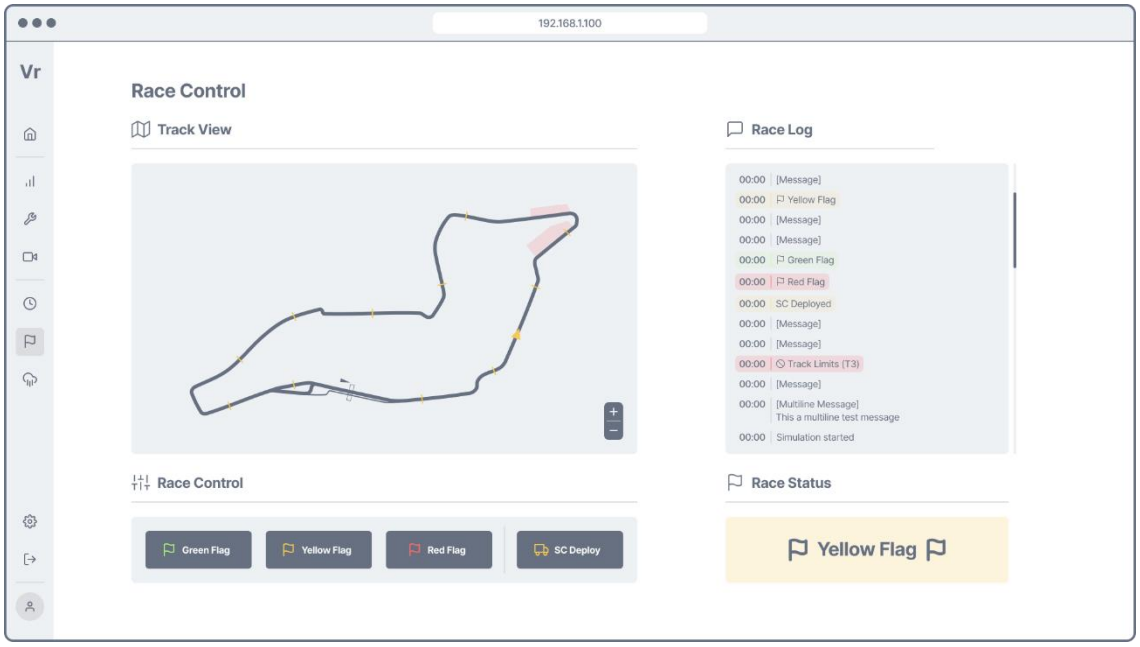


Ilustración 23. Wireframe de la vista de Race Control

Una vez realizado todo el prototipado de la interfaz, se realizan pruebas de usuario para comprobar la fácil compresión de la interfaz y sus elementos por parte del usuario. Estas se hacen mediante pruebas de interacción, navegando por los múltiples menús que esta presenta, y pidiendo al usuario que realice tareas dentro de esta, evaluando su interacción con la misma

Seguidamente, se procede a transformar los *wireframes* desarrollados más la experiencia obtenida en las pruebas de usuario en un diseño completo de la interfaz, donde se presenta la misma con los colores, tipografías y elementos que finalmente serán implementados. En la Ilustración 24 se muestra la misma pantalla presentada anteriormente como ejemplo de *wireframe*, pero esta vez una vez completado el diseño

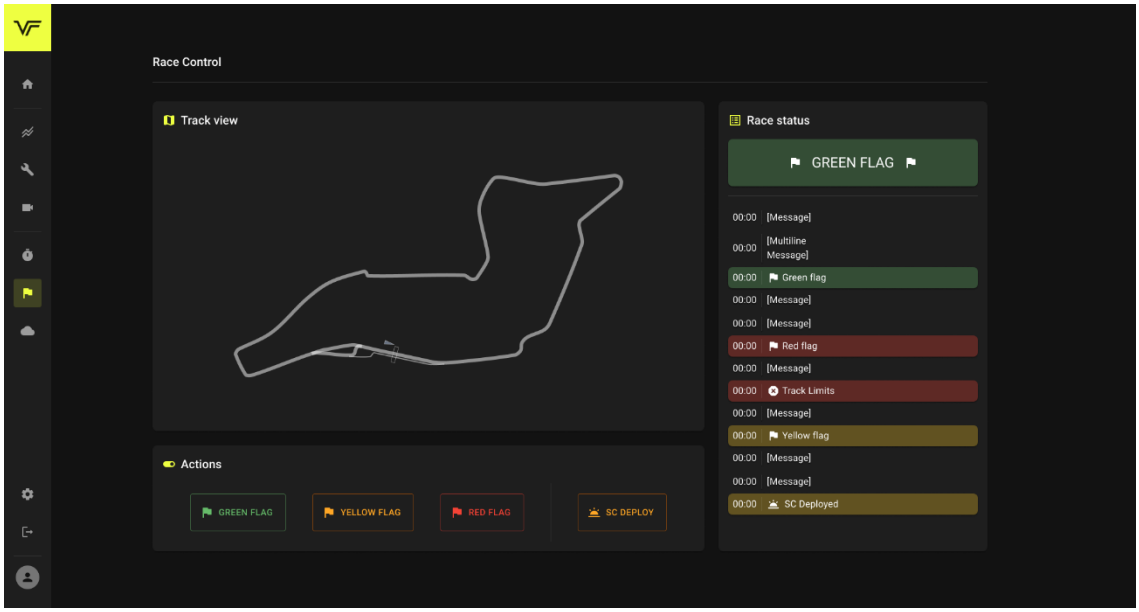


Ilustración 24. Interfaz de usuario completa de la vista de Race Control

De nuevo, se llevan a cabo pruebas de la interfaz desarrollada frente a usuarios finales, para comprobar la correcta comprensión de esta por estos. Todas las interfaces con su diseño completo serán más tarde usadas como referencias para su implementación, manteniendo tanto como sea posible la similitud entre ambas la versión aquí diseñada y la implementada

5.6. Implementación del sistema

En esta fase es donde se realiza la implementación del sistema en su totalidad, usando como base todo lo expuesto en las anteriores fases del proceso. En este apartado se pretende dar a conocer los detalles y decisiones detrás de cada una de las implementaciones relevantes de módulos realizadas, tanto en el panel de control web, como en el simulador 3D propiamente dicho.

5.6.1. Implementación del circuito 3D

Para la creación del circuito 3D, en el cual el piloto conduce dentro de la simulación, se ha perseguido el objetivo de conseguir el mayor realismo y similitud a su contraparte real posible, debido a la importancia que esta similitud guarda. Para ello, se ha hecho uso de técnicas de obtención, tratamiento y representación de datos de elevación del terreno reales. A continuación, se describe el proceso desde la obtención hasta el resultado final dentro del simulador 3D.

En primera instancia, se han de conseguir los datos de elevación del terreno. Si bien existen proveedores de estos datos los cuales ofrecen una cobertura a nivel global, estos presentan una baja precisión y nivel de detalle, del orden de entre 60m y 30m, es decir, se mide la altura de un punto especifico cada 60 o 30m, respectivamente. Esto, para aplicaciones que manejan mayores áreas o no requieren de alta precisión, puede ser suficiente, pero para este caso, es demasiado poco detallado. En la Ilustración 25, se muestran las diferencias entre 30m y 5m

Ilustración 25. Diferencia entre datos de 30m y 5m. En blanco, las mejoras del de 5m

Por tanto, se acuden a otros proveedores de datos en búsqueda de mayor nivel de detalle. Normalmente, los proveedores que pueden poseer mayor nivel de detalle que los globales, suelen ser administraciones nacionales o locales del país del que se desean los datos. En el caso de este proyecto, el circuito modelo, Imola, se encuentra en Italia, en concreto en la región de Emilia Romaña. El proveedor de datos usado es, en este caso, el *Geoportale de la región de la* Emilia Romaña. Mediante este ha sido posible la obtención de los datos de elevación del terreno (DTM) en una resolución de 5m, que es substancialmente inferior a lo que otros proveedores nacionales o globales pueden ofrecer.

Primero, debemos localizar las coordenadas del circuito y sus límites dentro de toda la región, de manera que podamos descargar solo las áreas necesarias donde se encuentra ubicado este. Para ello, nos ayudamos de la herramienta de visualización que el *Geoportale* nos ofrece.

Ilustración 26. Visor del Geoportale de Emilia Romaña

Una vez localizadas estas coordenadas, procedemos a descargar los cuadrantes que contienen el circuito y sus alrededores, marcados en color azul en la Ilustración 26. Visor del Geoportale de Emilia Romaña

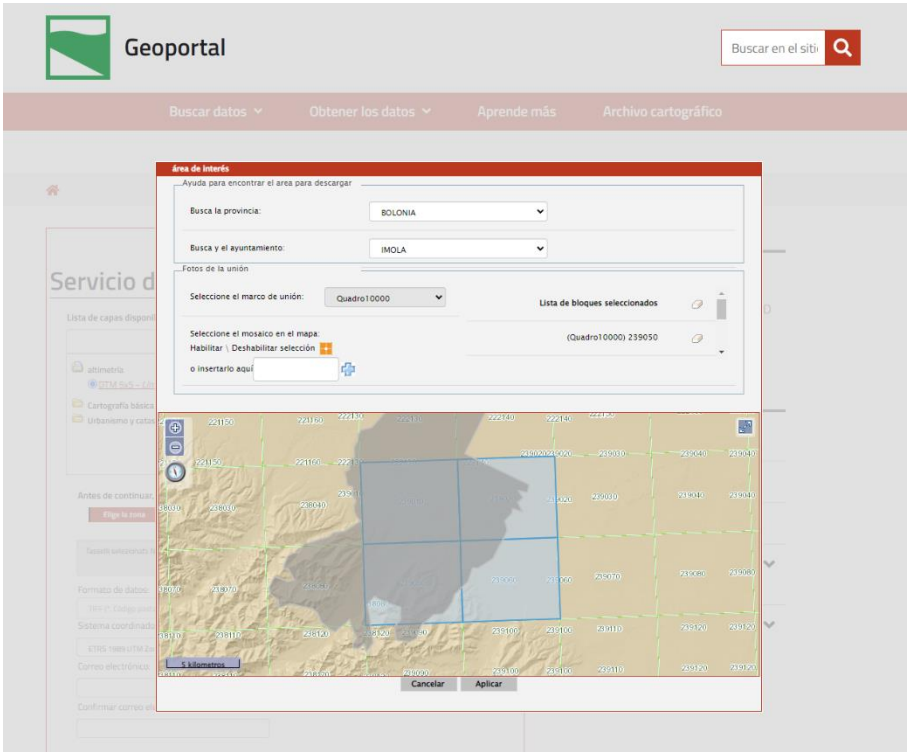


Ilustración 27. Cuadrantes DTM seleccionados para su descarga

Ahora que ya disponemos de los datos, debemos de tratar estos para su uso dentro del simulador. Esto es un proceso largo y complejo, compuesto por múltiples fases y procesos, los cuales se muestran a continuación. Para la ayuda en estas tareas se han usado dos herramientas: *ArcGIS Pro* y *TerreSculptor*

Primero, haciendo uso de *ArcGIS Pro*, importamos los ficheros descargados. Obtenemos un aviso de que estos no se encuentran georreferenciados, es decir, los datos que estos contienen no están asociados a un sistema de coordenadas conocido. *ArcGIS Pro* dispone de herramientas que nos puede ayudar a georreferenciar estos archivos al sistema de coordenadas que se indicaba en la descarga de los mismos en el *Geoportale*

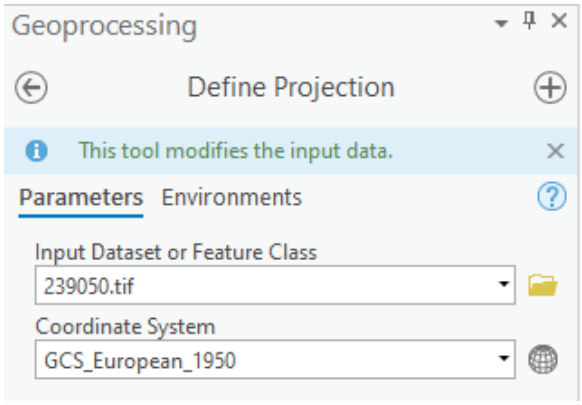


Ilustración 28. Georreferenciación de coordenadas de archivo DTM

Una vez completada esta georreferenciación, se nos representan los resultados en formato de imagen en blanco y negro, siendo el negro altura 0 y el color blanco la altura más alta de nuestros datos. Combinamos los 4 archivos descargados de DTM en uno y el resultado obtenido es el mostrado en la Ilustración 29

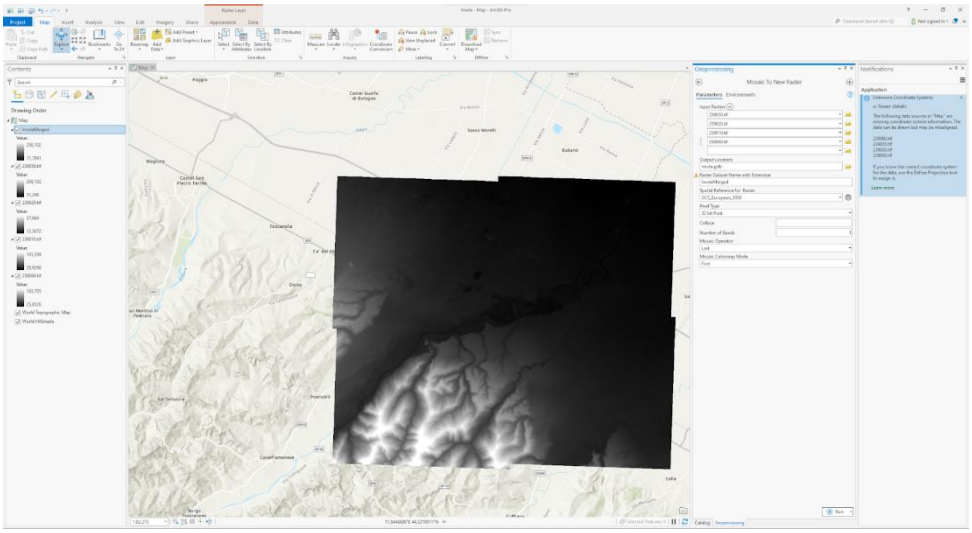


Ilustración 29. Resultado de importación, conversión y combinación de archivos DTM

Ahora, por limitaciones en el proceso de importación, cambiamos el sistema de coordenadas inicial (GCS European 1950) a uno más comúnmente usado (WGS84 32N), utilizando una herramienta proporcionada por el *software* para ello.

A partir de este punto, debemos de tener en cuenta en que dos sitios vamos a hacer uso de estos datos, y hacer las transformaciones necesarias para cada uno. El primer sitio, *Cesium*, el otro, *Unreal Engine*.

Cesium o *Cesium for Unreal*, explicado en el apartado de "Técnicas y herramientas" de este documento, dispone de la capacidad de importar datos propios y combinar estos con otras capas de las que este servicio dispone, como la de imágenes Satelitales, usada en este proyecto, sin perder en ningún momento la sincronía georreferenciada entre diversas capas.

Debido a limitaciones de como representa los terrenos este *plugin* dentro de *Unreal Engine* (No lo hace con los archivos de terreno nativos de *Unreal Engine*, con lo que las herramientas con las que se puede trabajar son limitadas*)*, se decide usar los datos que este provee como referencia para la construcción del circuito, de manera que se superponga a un terreno importado directamente sobre *Unreal Engine,* el cual hace uso de los mismos datos, pero siendo estos importados conforme a los formatos nativos de *Unreal Engine.*

El terreno importado directamente en *Unreal* Engine tiene la desventaja de perder la georreferenciación y la escala que el original guarda. Para hacer coincidir ambos terrenos (*Cesium* e importado directamente en *Unreal Engine*), debemos de insertar alguna referencia común en ambos. Para ello, insertamos cuatro pilares en forma de "montañas", los cuales haremos coincidir manualmente a mano, los cuales se pueden visualizar en la Ilustración 30

Ilustración 30. "Pilares" de referencia

Ahora que disponemos de una referencia para hacer coincidir ambos terrenos de las dos plataformas, procedemos a la importación y/o trasformaciones específicas de cada una.

En el caso de *Cesium*, simplemente nos basta con exportar nuestros datos de altura en blanco y negro como una imagen del tipo *PNG*. Esta, es importada a través de su panel web y descargada más tarde a través de los ajustes de este *plugin* en *Unreal Engine.*

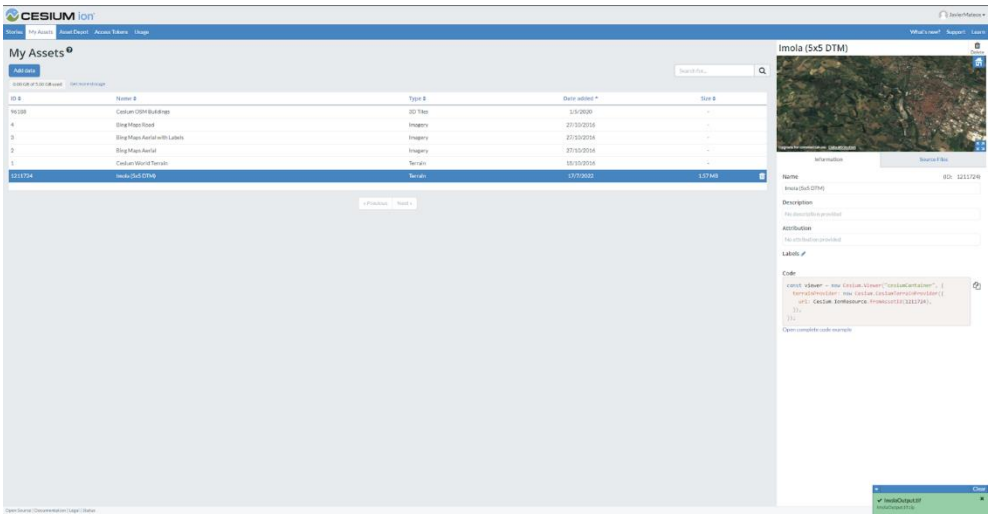


Ilustración 31. Portal web de Cesium

En el caso de *Unreal Engine*, debemos de realizar alguna transformación más antes de poder ser importado. Para ello, usaremos el *software* de TerreSculptor, el cual nos permite escalar y recortar el terreno de tal manera que encaje con alguno de los tamaños admitidos por el motor gráfico y nos deshagamos de los datos sobrantes, respectivamente.

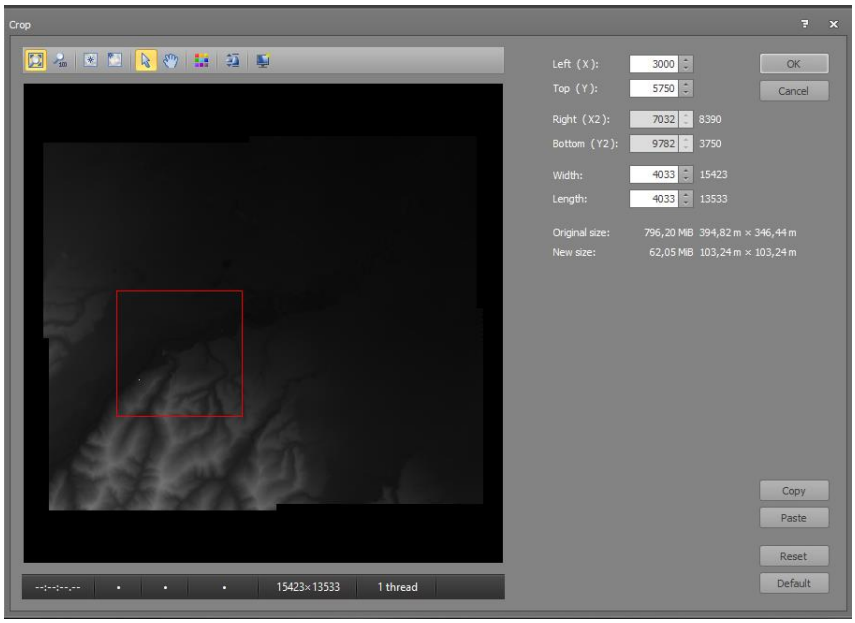


Ilustración 32. Operación de recorte del terreno

Ahora, combinamos ambos terrenos y los hacemos coincidir dentro de *Unreal Engine*. De esta manera obtenemos ambos terrenos georreferenciados y coincidentes, el de *Cesium* junto a la capa satelital para referencia de posicionamiento de objetos y otros, y el de *Unreal Engine*, en color gris, que posee todas las capacidades de edición que los terrenos nativos en este motor poseen.

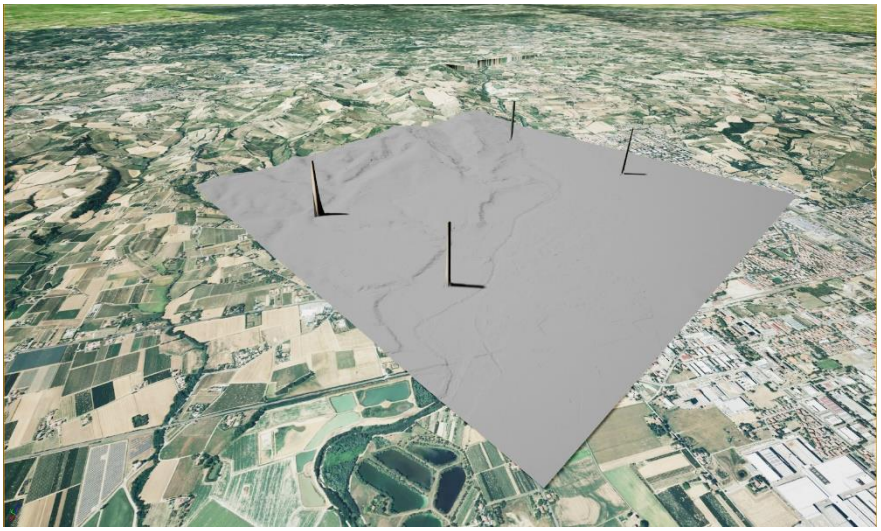


Ilustración 33. Resultado final de terreno

Como ejemplo, se muestra el proceso de replicar el asfalto del circuito mediante el uso de la capa satelital con transparencia como referencia, ejecutando el posicionamiento del asfalto sobre el terreno nativo importado de *Unreal Engine.*

Ilustración 34. Ambas capas superpuestas vistas desde arriba

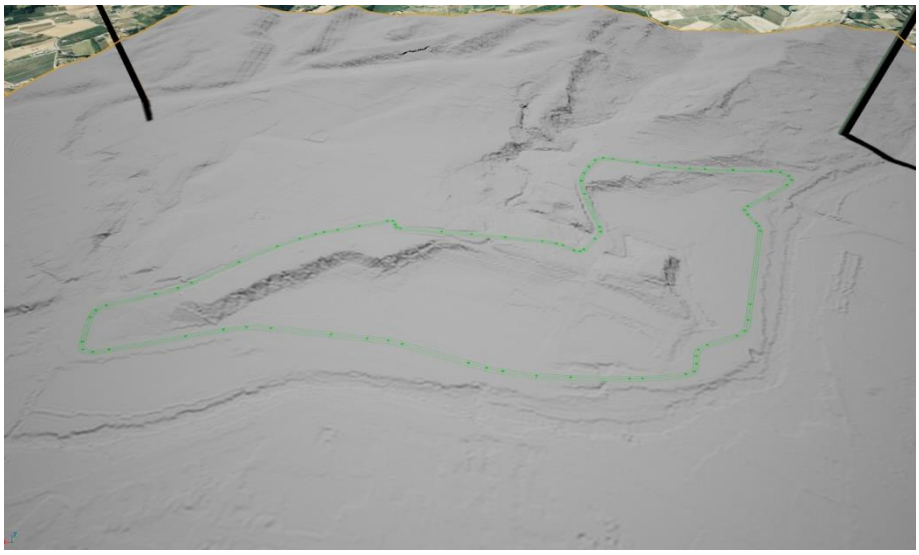


Ilustración 35. Trazado del circuito final

La superficie del circuito propiamente dicho se crea mediante el uso de la herramienta *Terrain Spline* del simulador, la cual permite la creación de *splines* las cuales se adhieren y siguen el contorno del terreno de forma suave, de manera que se consigue que el trazado este "pegado" al suelo.

El trazado dibujado mediante esta herramienta carece de forma de visualizarlo, a excepción de dentro del editor, como se ve en la Ilustración 21. Por ello, debemos asignarle un objeto que replicara a lo largo de toda su longitud. En este caso, un segmento de pista, modelo y texturizado desde cero, mostrado en la Ilustración 36. Segmento de pista usado, dentro del software Quixel Mixer:

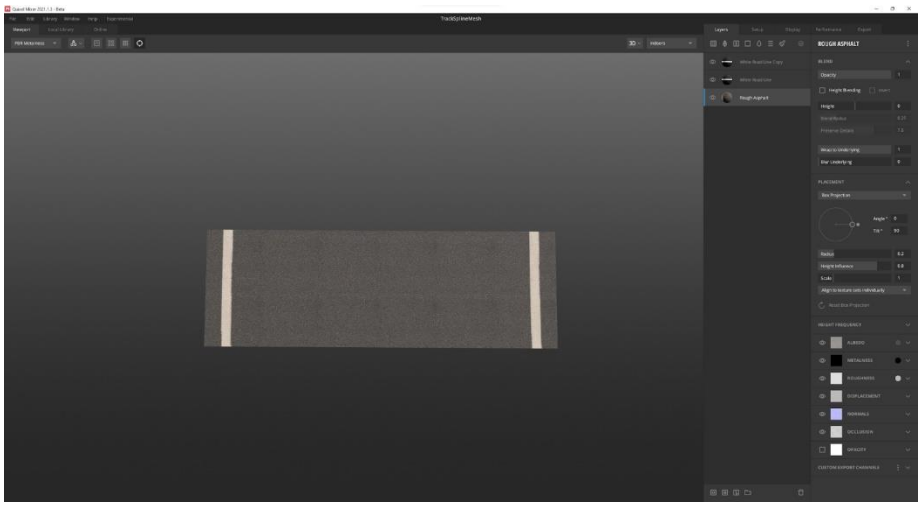


Ilustración 36. Segmento de pista usado, dentro del software Quixel Mixer

Aparte de la propia geometría del trazado, la cual coincide con la imagen satelital superpuesta, una buena métrica para comprobar la correcta representación del circuito dentro del simulador es su longitud. El circuito de la vida real posee 4,909 km medidos en su línea central, el representado en el simulador, medidos desde la misma posición, posee la misma cantidad, como muestra la Ilustración 37. Longitud del circuito implementado, en km.

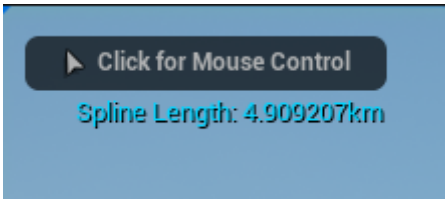


Ilustración 37. Longitud del circuito implementado, en km

Con la utilización de esta técnica, se consigue un resultado suficientemente fidedigno y alternativo a técnicas más costosas usadas en la industria, como el escaneo mediante LiDAR de superficies.

Finalmente, y con objetivo de dar más calidad visual al entorno, se le aplica una serie de texturas (hierba y grava) mediante el uso de un *Material Landscape*, usando como base para este material la implementada por el usuario *Unreal Sensei*, disponible públicamente (Unreal Sensei, 2021). El resultado final se muestra en la Ilustración 38. Texturas del terreno

Ilustración 38. Texturas del terreno

5.6.2. Implementación de la Realidad Virtual

En el tipo de simulador aquí desarrollado, la solución de visualización del simulador 3D más común suele ser empleando un conjunto de proyectores, dispuestos en círculo, de manera que proporcionen al piloto una visión de 180 grados del entorno 3D. Este planteamiento, aparte de costoso, tiene un problema dada la naturaleza de proyección de las imágenes: la profundidad de los objetos.

La proyección, al ser una imagen 2D, el ojo no es capaz de captar la profundidad de los objetos, simplemente "se la imagina", en función de sombras u otros. Por tanto, se busca la manera de paliar este efecto con nuevas tecnologías, ya que la profundidad es un aspecto clave para el piloto. Por ejemplo, el piloto puede pegarse más a los objetos o muros si distingue profundidad, de manera que pueda hacer una mejor trazada, ganando así tiempo en el cómputo total de la vuelta. La tecnología elegida para solucionar este problema es la Realidad Virtual (VR)

La VR permite al piloto distinguir la profundidad entre objetos, debido a las diversas proyecciones que esta usa, que simulan como vemos nosotros en la vida real, y aumentando los grados de visión que una solución de proyectores puede ofrecer, al poder este ver en cualquier grado de giro. Por tanto, el simulador gana en inmersión y realismo, abaratando costes.

La implementación en *Unreal Engine* de un proyecto en VR es sencilla. Este se encuentra preparado con una serie de *plugin* los cuales hacen el proceso de integración muy intuitivo. En este caso, se han usado las gafas de VR *HP Reverb G2* en conjunción con uno de los múltiples *plugins* que ofrecen una integración con el motor, el de *OpenXR.*

Al desarrollador solo le resta insertar una cámara de tipo VR y configurar si es sujeto va a estar sentado o de pie durante su uso, siendo en este caso la primera opción. Se puede visualizar la cámara VR y su posición en la Ilustración 39

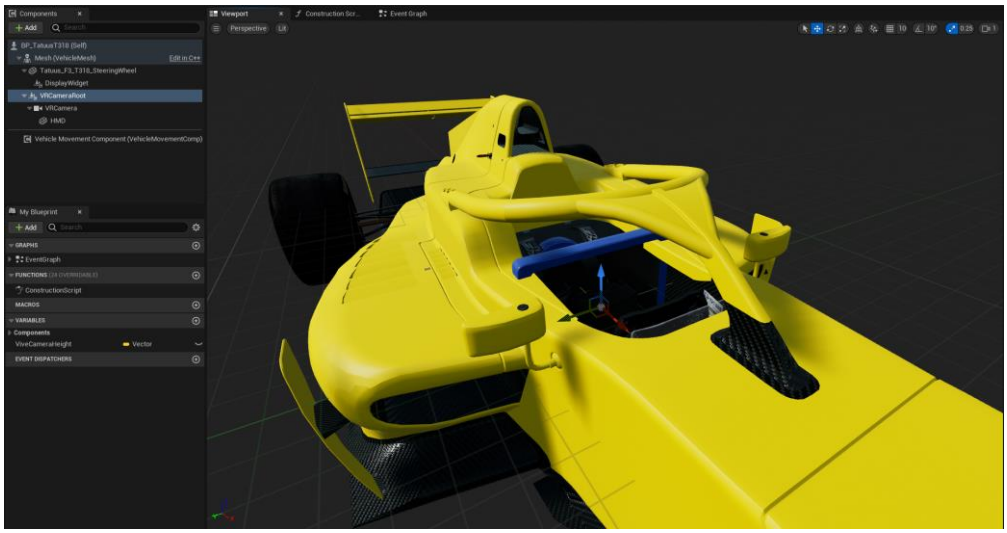


Ilustración 39. Cámara VR (Azul) dentro del vehículo

5.6.3. Implementación del vehículo

El vehículo implementado dentro del simulador 3D corresponde a un vehículo existente y en uso en la vida real, el *Tatuus F3 T-318.* Esta decisión se debe a dos motivos, los cuales se detallan a continuación.

El primero de estos es que, al ser un vehículo existente en la realidad, cumple con el objetivo propuesto de dar un enfoque comercial y/o real al simulador, ya que los equipos cliente buscan en este tipo de productos entrenar con el vehículo que usan en la vida real.

El segundo motivo, y el más relevante de los dos, es que se dispone de datos de técnicos y de rendimiento, obtenidos a través del manual públicamente disponible de este vehículo (Tatuus, 2021). Estos datos son usados como entradas para el modelo de dinámica de vehículos. Sin estos, parámetros del coche como potencia, ajuste de las suspensiones, relación de caja de cambios, etc., tendrían que ser "inventados", lo cual dificulta la obtención de un comportamiento realista, ya que el modelo matemático de dinámica de vehículos arrojaría resultados erróneos o irrealistas.

Para la implementación del vehículo en *Unreal Engine* debemos dividir la tarea en dos partes: La visual y la funcional.

Para la primera, se debe preparar e importar un modelo 3D del vehículo, el cual ha sido comprado en una página de venta de modelos 3D. Este modelo, se encontraba en parte "roto", causando estos varios problemas: Había perdido la asignación de los materiales a las distintas partes que lo componen, se encontraba a una escala incorrecta, el centro de masa de cada componente estaba desplazado, etc. Por ello, hubo que corregir estos problemas previos a su importación haciendo uso de *Blender* con sus herramientas y los datos de escala de los cuales se dispone a través de la documentación del vehículo, visibles en la Ilustración 40.

TATUUS
RACE CAR MANUFACTURER

1.4 OVERVIEW

FRONT TRACK	1575 mm
REAR TRACK	1530 mm
WHEELBASE	2900 mm
OVERALL LENGTH	4855 Max mm
OVERALL WIDTH	1850 Max mm
OVERALL HEIGHT	950 mm (from reference plane)

Ilustración 40. Dimensiones del vehículo. Disponibles en su manual

Una vez estos problemas fueron corregidos, se procede a su importación como *Skeletal Mesh* dentro del motor gráfico, separado en 6 componentes: El cuerpo, el volante y las 4 ruedas, estas últimas cada una de forma independiente. En la Ilustración 41se puede visualizar el vehículo con sus texturas y las capsulas de colisión (simplificadas) de cada objeto de esta división, en color morado:

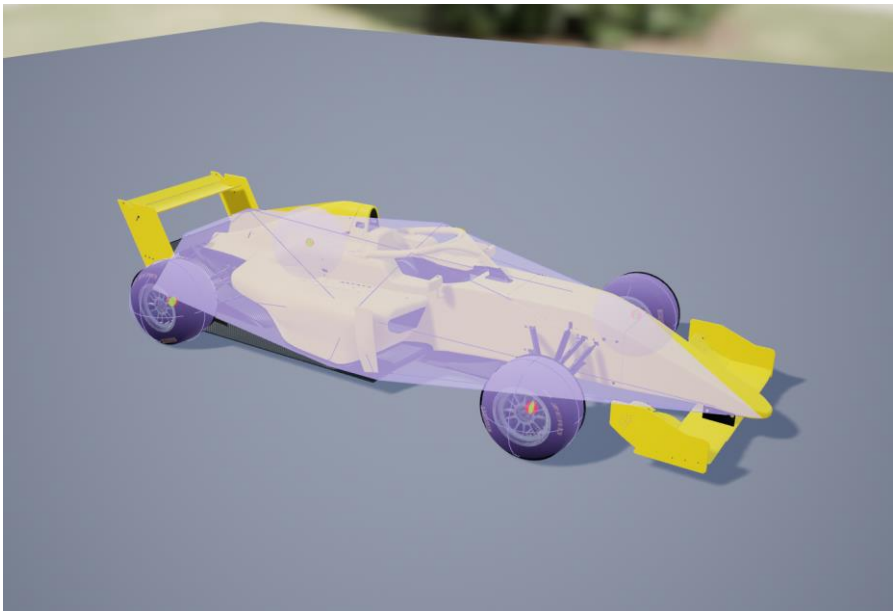


Ilustración 41. Componentes del vehículo dentro de Unreal Engine

Para finalizar este apartado, establecemos las animaciones necesarias sobre los elementos móviles del coche (ruedas y volante) para que estos giren cuando lo haga el piloto en la vida real. En la Ilustración 42, se visualiza la implementación de las animaciones de las ruedas mediante el sistema *AnimGraph* del motor gráfico.

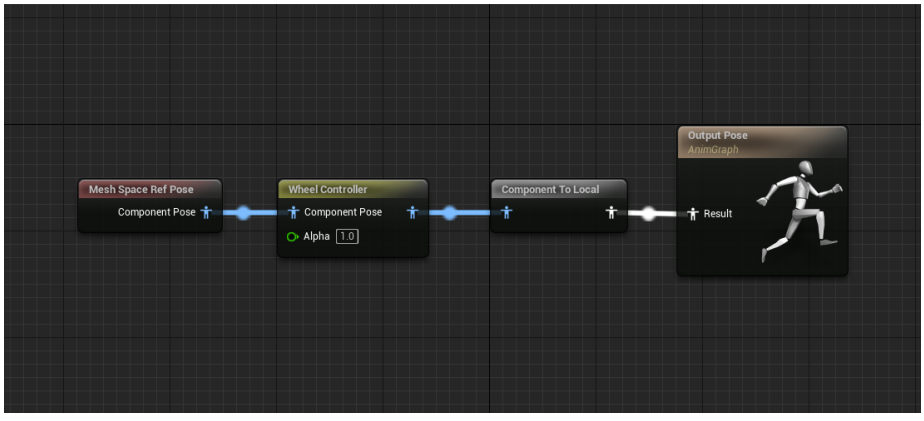


Ilustración 42. Animación de las ruedas del vehículo

Por otro lado, para la parte funcional del vehículo debemos de crear y configurar una serie de estructuras de datos que determinaran el comportamiento del vehículo. Estos datos eran más tarde usados por el modelo de dinámica de vehículo usado, *Chaos Vehicles*.

El primer paso consiste en instanciar las estructuras que guardarán los parámetros técnicos de las ruedas, que usara más tarde el modelo de dinámica de vehículos. Se crean dos configuraciones diferentes, en función del eje donde se encuentran: Ruedas delanteras y Ruedas traseras. Dentro de ambos se establecen los parámetros necesarios, como anchura de la rueda, radio de la rueda, si recibe potencia del motor, ajustes de la suspensión, ajustes del freno, etc. Estos parámetros se pueden visualizar en la Ilustración 43.

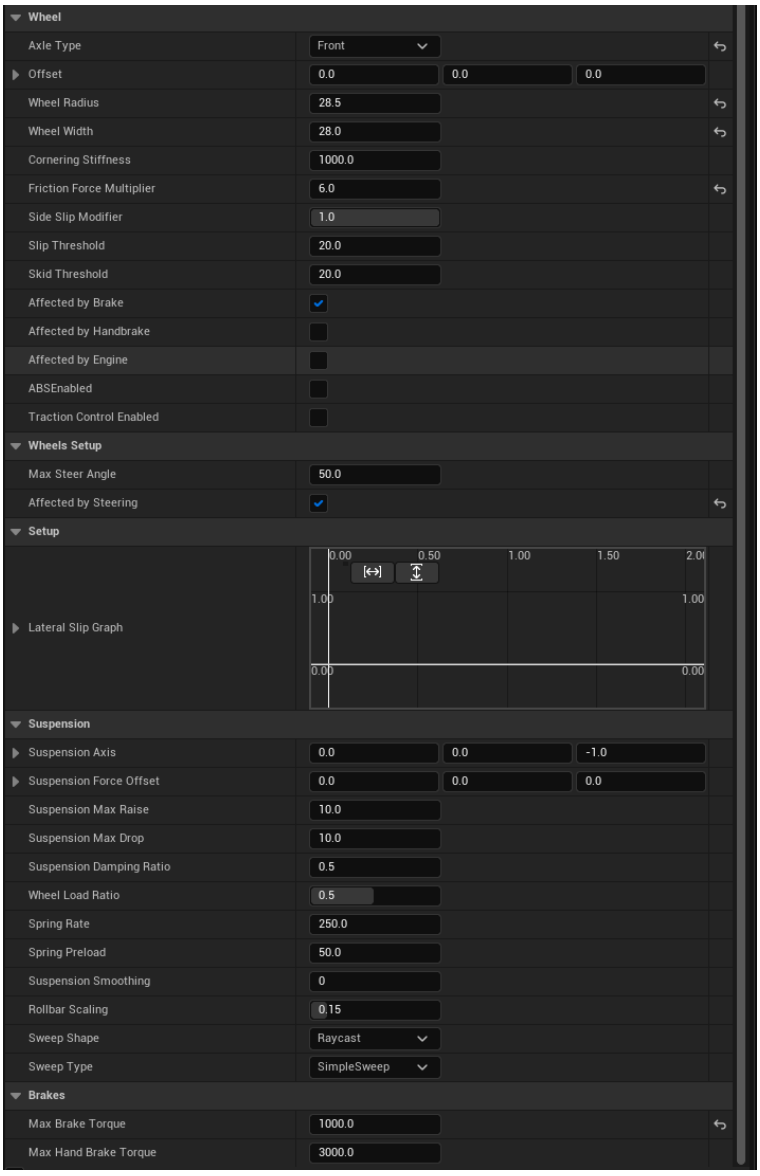


Ilustración 43. Parámetros de las ruedas delanteras del vehículo

Seguidamente, debemos de establecer el resto de los parámetros del vehículo. Para ello instanciamos otra estructura de datos donde introduciremos datos del motor, dirección, transmisión, etc. Algunos de estos pueden ser visualizados en la Ilustración 44.

Ilustración 44. parámetros de motor y transmisión del vehículo

Una vez completados estos pasos, los resultados visuales y funcionales son combinados bajo una *Blueprint* de tipo *Pawn* (Objeto que puede ser movido por el usuario) del motor gráfico, de manera que el vehículo creado pueda ser instanciado en el entorno 3D.

Por último, debemos de manejar la entrada del usuario, ejercida mediante un dispositivo hardware de tipo volante (*Thrustmaster T300 Alcantara*), y comunicar la acción pertinente al modelo de dinámica de vehículos. Esto se ha programado usando el sistema de programación visual *Blueprints*, como se puede visualizar en la Ilustración 45.

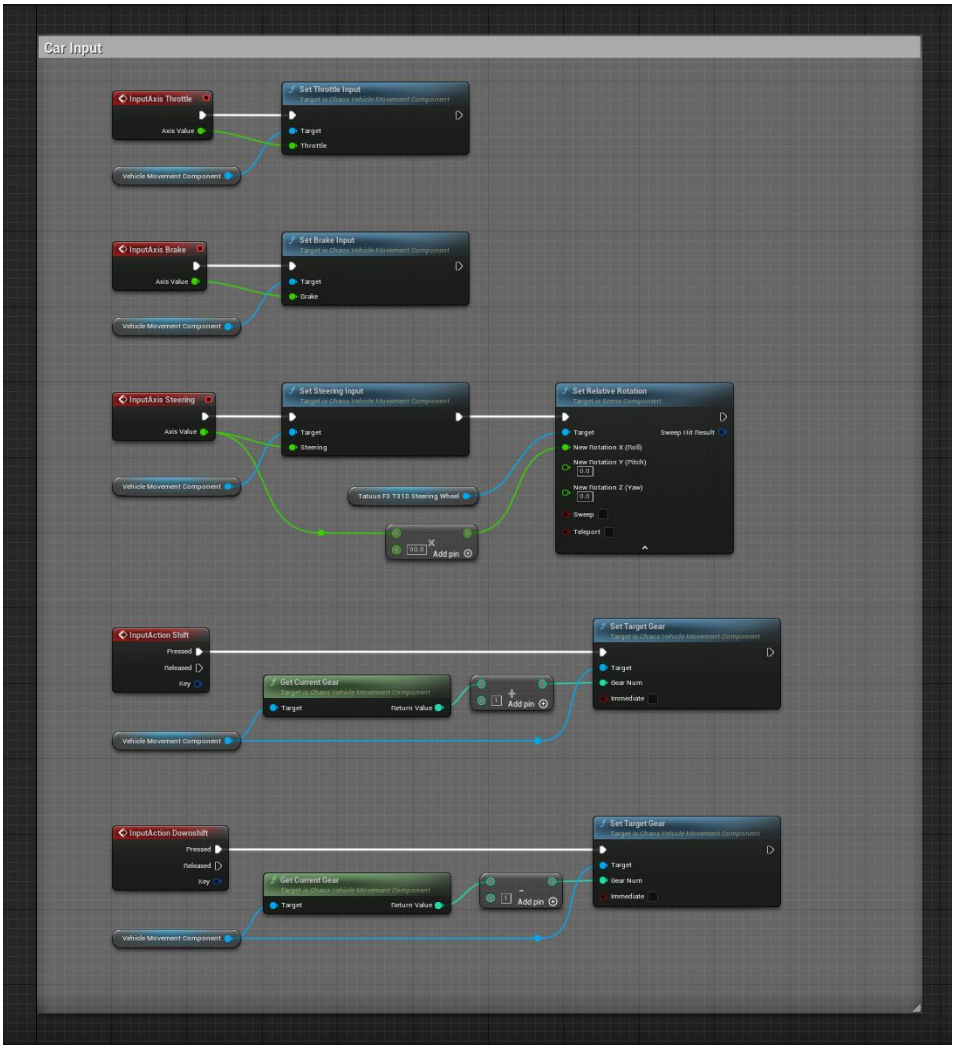


Ilustración 45. Manejo entradas del usuario

5.6.4. Implementación de la telemetría del vehículo

La obtención y visualización de la telemetría del vehículo es un trabajo combinado entre ambos sistemas del simulador: simulador 3D y panel de control web. Por tanto, en el siguen punto se tratará el flujo que sufre esta información desde que se captura en un sistema hasta que se visualiza en el otro.

Todo comienza en el simulador 3D, donde, usando el sistema de *splines* del simulador, se ha trazado una *Spline* la cual viaja por la línea central de la pista. Esta se traza *grosso* modo, es decir, de manera que la línea siga el contorno del asfalto, pero sin guardar ninguna equidistancia los puntos que la componen, como se puede observar en la Ilustración 46.

Ilustración 46. Spline base que sigue la pista por su línea central

Para una correcta medición de la telemetría, esta debe ser capturada en intervalos regulares de distancia, de manera que, a distintas vueltas se pueda comprar el estado del coche en un mismo punto, y sacar la diferencia de tiempo de ese punto respecto múltiples vueltas. Para ello, debemos de dividir la *Spline* anterior en segmentos equidistantes. Para ello, obtenemos su longitud total, y mediante la distancia entre puntos que se desea en cm, se obtiene la separación entre cada uno. Con esta separación, creamos una nueva *Spline*, que, siguiendo el trazado de la anterior, inserte un nuevo punto cada distancia de separación calculada. Este sistema se puede observar en la Ilustración 47.

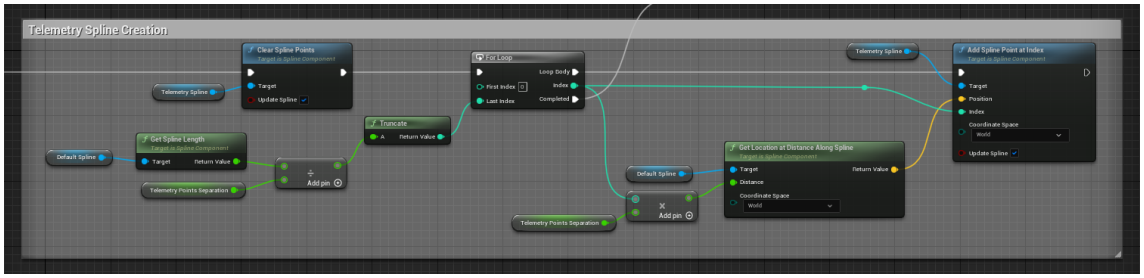


Ilustración 47. Sistema de generación de Spline de telemetría

En este caso, se ha decidido separar los puntos cada 2m, ya que provee una medida suficientemente detallada para la comparación de diferentes telemetrías medidas. El resultado obtenido visualmente es el mostrado en la Ilustración 48

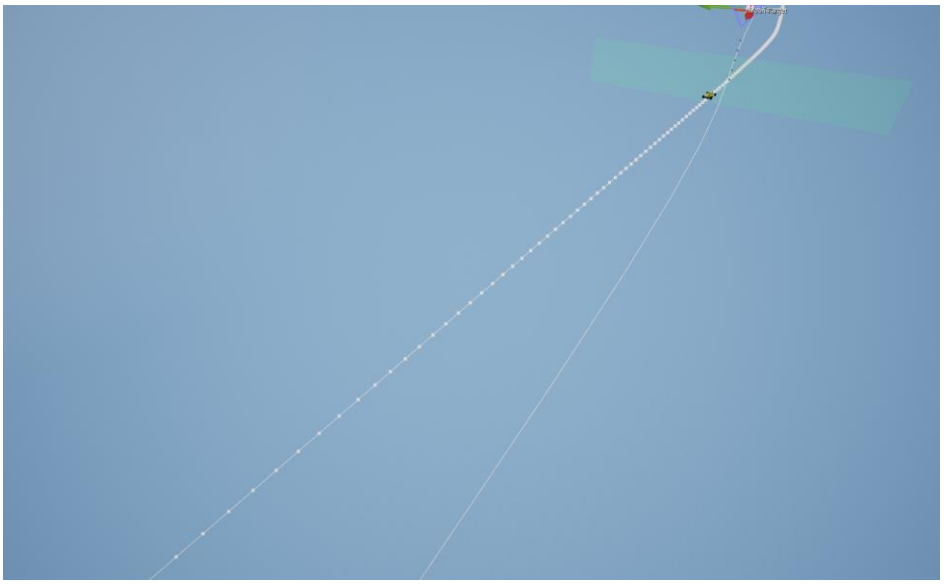


Ilustración 48. Secciones de 2m de separación

Una vez realizados estos pasos, podemos pasar a la implementación *per-se.* Para ello, debemos de distinguir dos pasos: La medición y él envió.

El primero, debemos de realizarlo cada vez que circulemos por encima de alguno de los puntos que hemos establecido antes. Para ello, recuperamos la posición actual del vehículo en la pista y buscamos el punto de la *spline* que se acerque más a él. Si el ID de este (Siendo el ID el índice del punto dentro de la *spline*) es mayor que el anterior enviado, significa que no es el mismo ni inferior, es decir, hemos avanzado en la pista, por tanto, debemos de enviar nueva telemetría. También se realiza la comprobación de que no es la vuelta de salida. Hasta que no se pasa una vez por meta no se empieza a enviar telemetría. Este sistema se puede visualizar en la Ilustración 49

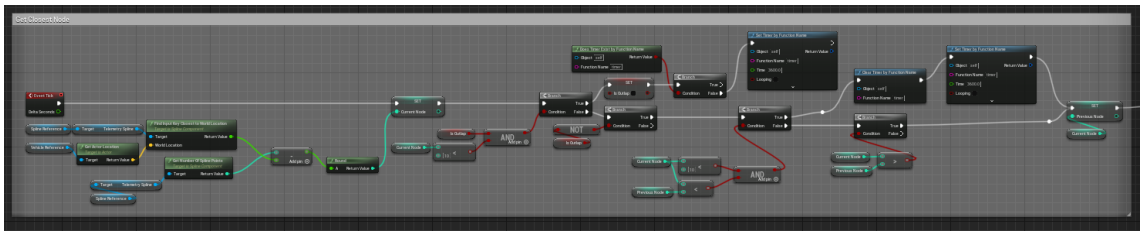


Ilustración 49. Sistema de detección de punto de envió de telemetría

Seguidamente, pasamos al paso de envió. Para ello, primero debemos de leer los datos actuales del vehículo de los diversos canales de los que se tiene interés (acelerador, freno, marcha engranada, etc.) sumado al tiempo transcurrido desde el ultimo envío. Mediante la instancia del vehículo que se encuentra posicionado en el entorno 3D, podemos acceder a las variables que almacenan estos datos, para guardarlos en una estructura personalizada, que más tarde dará formato al mensaje que se envía a través de *sockets.* Esto se puede observar en la Ilustración 50

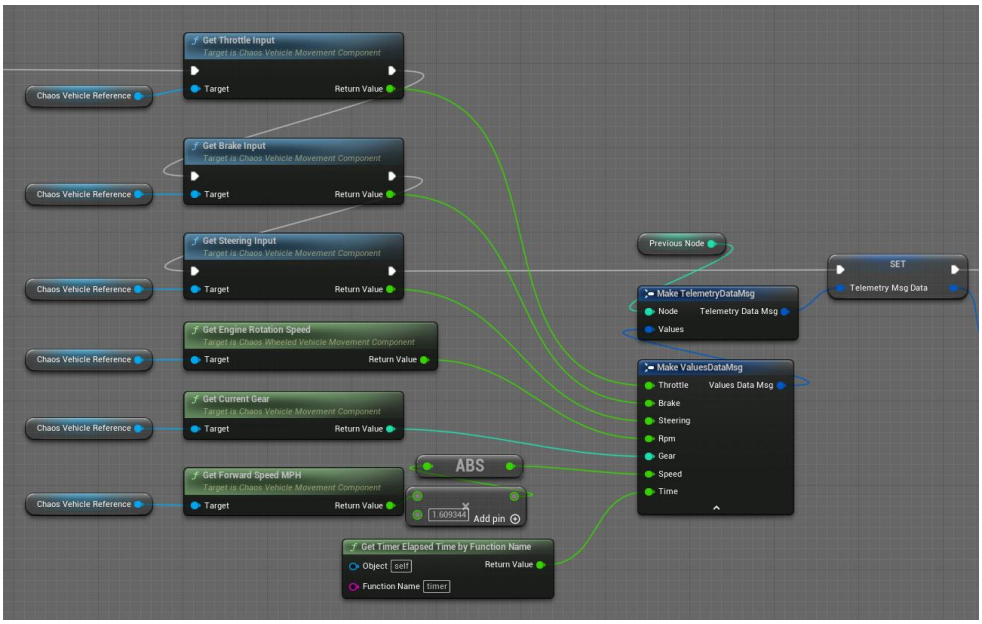


Ilustración 50. Obtención de datos del vehículo

Ahora solo resta él envió, el cual se realiza haciendo uso de la biblioteca de *SocketIOClient*. Se transforma la estructura en un mensaje de formato JSON y se ejecuta la función de Emitir mensaje que brinda la mencionada biblioteca. Esta funcionalidad se puede visualizar en la Ilustración 51

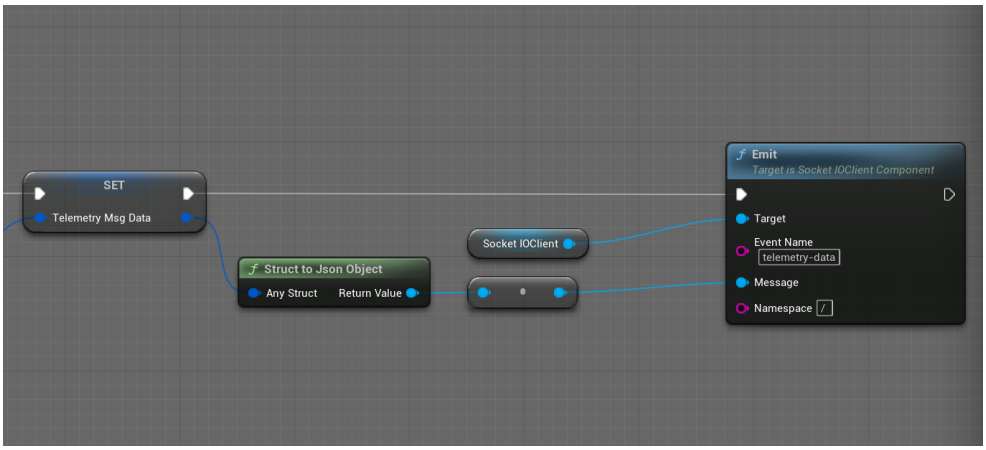


Ilustración 51. Envío de telemetría

Seguidamente, el servidor recibe esta información, que la guarda en su copia local y la retransmite a los clientes conectados a través del panel web.

La visualización de los datos en panel de control web se realiza mediante la biblioteca *Charts.js* en conjunto con su integración para *React.* Los datos recibidos, antes de ser mostrados, necesitan sufrir una serie de transformaciones. El proceso transcurre como sigue.

Primero, distinguimos que numero de nodo es. Si es el 0, creamos una nueva vuelta, si no, añadimos a la actual. Si es nueva vuelta, comprobamos cual es la mejor de todas, y calculamos el tiempo *delta* de cada vuelta entorno a esa. El tiempo *delta* dicta cuando de rápido o lento, en función del signo del número, se es en un punto en concreto respecto a la vuelta más rápida.

```
const checkBestLap = () => {
  // If no lap is set, set the first one
  if (localBestLap.lap === null && localTelemetryData.length === 1) {
    localBestLap = {
      lap: localTelemetryData.length - 1,
      values: localTelemetryData[localTelemetryData.length - 1].channels.find(
        (channel) => channel.label === "time"
      ).values,
    };
  } else {
    localTelemetryData.forEach((lap, index) => {
      if (
        localBestLap.lap !== null &&
        parseFloat(
          lap.channels.find((channel) => channel.label === "time").values[
            lap.channels.find((channel) => channel.label === "time").values
              .length - 1
          ]
        ) < parseFloat(localBestLap.values[localBestLap.values.length - 1])
      ) {
        localBestLap = {
          lap: index,
          values: lap.channels.find((channel) => channel.label === "time")
            .values,
        };
      }
    });
  }

  setBestLap(localBestLap);
};
```

Ilustración 52. Comprobación de mejor vuelta

A continuación, añadimos el nuevo nodo de datos recibido a la lista de datos representados en la gráfica. Dada la falta de una integración correcta por parte de la adaptación de la biblioteca de *Chart.js* a *React*, la interfaz no se ve actualizada automáticamente, como suele ser el caso en este tipo de aplicaciones reactivas. Por tanto, debemos hacer un refresco manual. Para ello, recalculamos todos los ejes de la gráfica de forma dinámica, calculamos el delta de esa vuelta respecto a la mejor, reinsertamos los datos, les aplicamos el estilo correspondiente, y aplicamos los cambios.

Finalmente, después de este largo proceso, el ingeniero puede visualizar la telemetría en tiempo real y pasada de la sesión actual.

5.6.5. Implementación de medición de tiempos

Para la medición de tiempo, reutilizamos parte del sistema implementado para la medición de la telemetría, en concreto, la *spline* inicial, que sigue *grosso* modo el trazado.

Partiendo de la s*pline* mencionada, se repite el proceso de tomar la longitud de esta y crear una nueva siguiendo su trazado subdividido de forma equidistante, aunque esta vez en lugar de dividir por metros, se dividide por cuantos puntos se desean en la totalidad de su longitud. En este caso, se ha dividido todo la *spline* en 12 puntos, los cuales representan los mini sectores en los cuales se mide el tiempo. El resultado visual se puede observar en la Ilustración 53, siendo los mini sectores los rectángulos de color verde transparente.

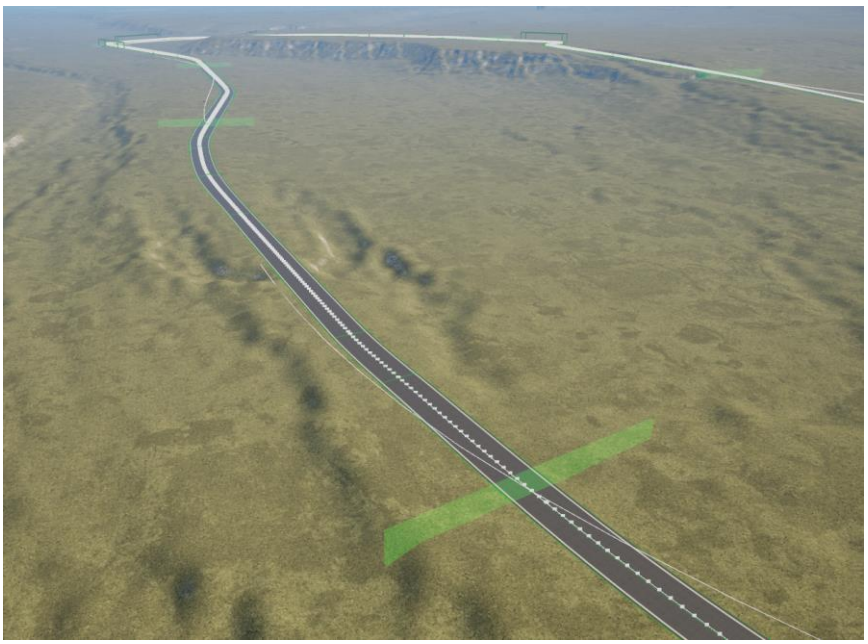


Ilustración 53. Representación visual parcial de los mini sectores

Cada vez que el vehículo atraviesa estos puntos, se dispara un evento el cual ejecuta una función. El código que compone esta función se encarga de tres labores: Medición del tiempo, transformación de su formato y envío del mismo.

Para la medición del tiempo se emplea un cronometro para contabilizar el tiempo transcurrido desde el ultimo envío, es decir, el último paso por uno de estos mini sectores.

Este tiempo debe ser transformado al formato correcto para su envío y visualización. Dada la naturaleza de los tiempos que se miden en este tipo de competiciones, el formato seguido es “00:00.000”, correspondiendo a minutos, segundos y milésimas respectivamente. Esto es guardado en la estructura personalizada correspondiente para su envío, junto con el índice del punto medido.

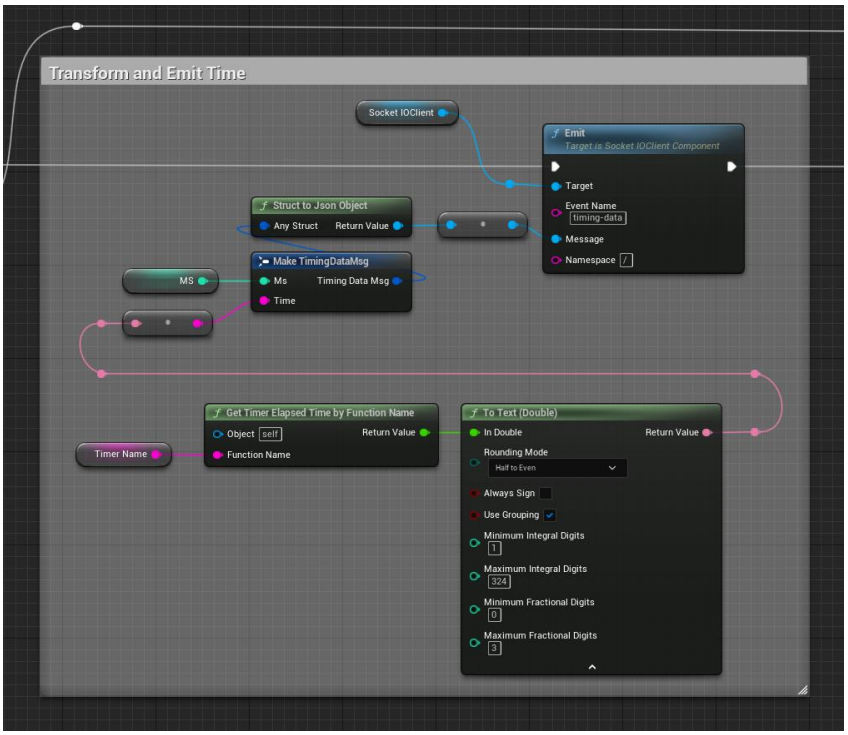


Ilustración 54. Medición, formateo y envío del tiempo transcurrido

En el servidor, se añaden los datos del mini sector a la copia local. Seguidamente se selecciona la vuelta más rápida de la sesión para ser distribuida junto con el resto de los datos al cliente una vez completadas las acciones necesarias.

```
/* Timing */
socket.on(wsConstants.messages.TIMING_DATA_MSG, (data) => {
  console.log(`📈 Timing data: MS${data.ms} - ${data.time}`);

  const msTime = getMSTime(data);

  // Add minisector to timing data
  addMinisector(data.ms, msTime);

  // Add sectors if needed
  addSector(data.ms, msTime);

  // Update best lap minisector data
  checkBestMinisector(data.ms, msTime);

  // Send data to all clients
  socket
    .to(wsConstants.rooms.TIMING_ROOM)
    .emit(wsConstants.messages.LAPS_DATA_MSG, {
      laps: sessionData.timing.laps,
    });

  socket
    .to(wsConstants.rooms.TIMING_ROOM)
    .emit(wsConstants.messages.BEST_LAP_DATA_MSG, {
      bestLap: sessionData.timing.bestLap,
    });
});
```

Ilustración 55. Código de manejo de tiempos de vuelta del servidor

Por último, en el panel de control se muestran los tiempos en formato de tabla, coloreando los mini sectores en función de si son el mejor de la sesión, es mejor que la anterior vuelta, o es peor que la anterior vuelta, de morado, verde o amarillo, respectivamente. La lógica iterativa para la selección del color correspondiente se puede visualizar en la Ilustración 56.

```
/* Table data & styling */
const MSCellStyling = (params) => {
  if (params.value == null) return "";

  var lapType = "";
  var msNum = params.field.replace("ms", "");

  try {
    if (
      parseFloat(params.value) ===
      parseFloat(bestLap.ms.find((x) => x.ms === parseInt(msNum)).time)
    )
      lapType = "best";
    else if (
      params.id > 0 &&
      parseFloat(params.value) <
        parseFloat(
          laps[params.id - 1].ms.find((x) => x.ms === parseInt(msNum)).time
        )
    )
      lapType = "improvement";
    else if (
      params.id > 0 &&
      parseFloat(params.value) >
        parseFloat(
          laps[params.id - 1].ms.find((x) => x.ms === parseInt(msNum)).time
        )
    )
      lapType = "worst";
  } catch (error) {
    lapType="";
  }


  if (params.id > 0) {
    return clsx("timing-styles", {
      best: lapType === "best",
      improvement: lapType === "improvement",
      worst: lapType === "worst",
    });
  } else {
    return clsx("timing-styles", {
      best: lapType === "best",
    });
  }
};
```

Ilustración 56. Código de cambio de color del mini sector

5.6.6. Implementación de reglajes

La funcionalidad del sistema comienza en el panel de control web, donde se presentan todos los ajustes que componen el reglaje del vehículo, clasificados por pestañas. Una vez el ingeniero aplica los cambios, estos datos son enviados al servidor, que a su vez los envía al resto de clientes, incluido el simulador.

Para aplicar los reglajes en el vehículo del simulador en *runtime* se ha tenido que crear funcionalidad propia haciendo uso de C++, ya que, por defecto, el sistema de *Blueprints* no es capaz de acceder a todas las variables internas del vehículo que dictan la configuración de este. Por tanto, se crea una función de C++, visible en la Ilustración 57, la cual se encarga de actualizar los parámetros del vehículo introducidos como entradas.

```
void ASetupManager::ChangeVehicleSetup(UPARAM(ref) UChaosWheeledVehicleMovementComponent* VehicleRef, FGener
{
    // General
    VehicleRef->Mass = GeneralSetup.Mass + GeneralSetup.FuelMass + GeneralSetup.BallastMass;

    // Aerodynamics
    VehicleRef->DownforceCoefficient = AerodynamicSetup.DownforceCoefficient;
    VehicleRef->DragCoefficient = AerodynamicSetup.DragCoefficient;
    VehicleRef->DragArea = AerodynamicSetup.DragArea;

    // Differential
    VehicleRef->DifferentialSetup.DifferentialType = DifferentialSetup.DifferentialType;
    VehicleRef->DifferentialSetup.FrontRearSplit = DifferentialSetup.FrontRearSplit;

    // Engine
    VehicleRef->EngineSetup.EngineBrakeEffect = EngineSetup.EngineBrakeEffect;
    VehicleRef->EngineSetup.EngineIdleRPM = EngineSetup.EngineIdleRPM;
    VehicleRef->EngineSetup.EngineRevDownRate = EngineSetup.EngineRevDownRate;
    VehicleRef->EngineSetup.EngineRevUpMOI = EngineSetup.EngineRevUpMOI;
    VehicleRef->EngineSetup.MaxRPM = EngineSetup.MaxRPM;
    VehicleRef->EngineSetup.MaxTorque = EngineSetup.MaxTorque;

    // Steering
    VehicleRef->SteeringSetup.AngleRatio = SteeringSetup.AngleRatio;
    VehicleRef->SteeringSetup.SteeringType = SteeringSetup.SteeringType;

    // Transmission
    VehicleRef->TransmissionSetup.bUseAutomaticGears = TransmissionSetup.UseAutomaticGears;
    VehicleRef->TransmissionSetup.bUseAutoReverse = TransmissionSetup.UseAutoReverse;
    VehicleRef->TransmissionSetup.ChangeDownRPM = TransmissionSetup.ChangeDownRPM;
    VehicleRef->TransmissionSetup.ChangeUpRPM = TransmissionSetup.ChangeUpRPM;
    VehicleRef->TransmissionSetup.FinalRatio = TransmissionSetup.FinalRatio;
    VehicleRef->TransmissionSetup.ForwardGearRatios = TransmissionSetup.ForwardGearRatios;
    VehicleRef->TransmissionSetup.GearChangeTime = TransmissionSetup.GearChangeTime;
    VehicleRef->TransmissionSetup.ReverseGearRatios = TransmissionSetup.ReverseGearRatios;
    VehicleRef->TransmissionSetup.TransmissionEfficiency = TransmissionSetup.TransmissionEfficiency;

    // Wheels
    for(int i = 0; i < WheelsSetup.Num(); i++)
    {
        VehicleRef->Wheels[i]->CorneringStiffness = WheelsSetup[i].CorneringStiffness;
        VehicleRef->Wheels[i]->FrictionForceMultiplier = WheelsSetup[i].FrictionForceMultiplier;
        VehicleRef->Wheels[i]->MaxBrakeTorque = WheelsSetup[i].MaxBrakeTorque;
        VehicleRef->Wheels[i]->MaxSteerAngle = WheelsSetup[i].MaxSteerAngle;
        VehicleRef->Wheels[i]->RollbarScaling = WheelsSetup[i].RollbarScaling;
        VehicleRef->Wheels[i]->SideSlipModifier = WheelsSetup[i].SideSlipModifier;
        VehicleRef->Wheels[i]->SkidThreshold = WheelsSetup[i].SkidThreshold;
        VehicleRef->Wheels[i]->SlipThreshold = WheelsSetup[i].SlipThreshold;
        VehicleRef->Wheels[i]->SpringRate = WheelsSetup[i].SpringRate;
        VehicleRef->Wheels[i]->SuspensionDampingRatio = WheelsSetup[i].SuspensionDampingRatio;
        VehicleRef->Wheels[i]->SuspensionMaxDrop = WheelsSetup[i].SuspensionMaxDrop;
        VehicleRef->Wheels[i]->SuspensionMaxRaise = WheelsSetup[i].SuspensionMaxRaise;
        VehicleRef->Wheels[i]->WheelLoadRatio = WheelsSetup[i].WheelLoadRatio;
    }
}
```

Ilustración 57. Código C++ implementado para aplicar reglajes

Esta función se prepara para que pueda ser llamada mediante el sistema de *Blueprints*, visible en la Ilustración 58, de manera que se combinan ambos sistemas. El resto de la implementación de la lógica, como la recepción y tratamiento del mensaje recibido, se realiza desde *Blueprints*.

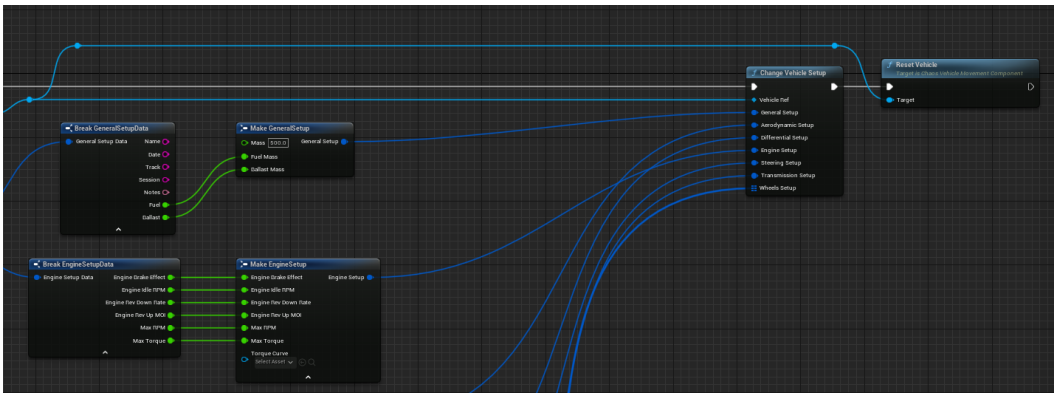


Ilustración 58. Vista parcial de la integración entre C++ y Blueprints

Debido al funcionamiento interno del modelo de dinámica de vehículos, al aplicar cambios en tiempo real al vehículo, este sufre un reseteo de sus variables, por lo cual se pierde el movimiento actual del vehículo. Por ello, se menciona en el manual de usuario que estos cambios deben de hacerse en parado para evitar esta situación.

5.6.7. Implementación de eventos de carrera

El origen y destino de estos eventos pueden suceder en ambos sentidos. Estos tienen la posibilidad de realizarse desde el simulador 3D al panel de control y viceversa. Se distinguen dos tipos implementados, eventos de carrera (Bandera verde, roja, etc.) y superación de límites de pista.

Para la primera, tras la acción del ingeniero, el panel de control envía un mensaje del tipo de esta clase de eventos al servidor, el cual redistribuye hasta el simulador 3D. Este, muestra al piloto en la pantalla del volante la situación comunicada mediante la alternación de la pantalla mostrada a la correspondiente a la situación recibida durante unos segundos. La implementación de esta recepción y cambio de pantallas se puede ver en la Ilustración 59. El resultado final se muestra en la Ilustración 60

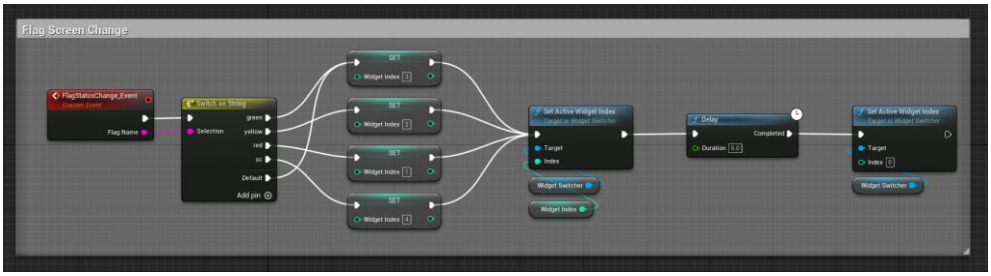


Ilustración 59. Cambio de pantalla en función del mensaje recibido

Ilustración 60. Pantalla mostrada al piloto temporalmente

Para la detección de límites de pista por parte del simulador 3D, se establecen una serie de zonas en las cuales, si el vehículo se introduce en ella, provoca la invocación de un evento el cual realiza él envió del correspondiente mensaje al servidor, conteniendo este un identificativo de en qué zona ha ocurrido. Estas zonas se pueden ver en la Ilustración 61en forma de rectángulo con bordes verdes

Ilustración 61. Zonas de detección de límites de pista

5.6.8. Implementación de condiciones del entorno

Para la implementación del ciclo día-noche y las condiciones atmosféricas dinámicas se ha hecho uso del *plugin Ultra Dynamic Sky*, el cual proporciona una forma sencilla de implementar ambas funcionalidades buscadas e interfaces para interactuar con ellas. Por tanto, el proceso de modificar cualquiera de estos dos ajustes queda de la siguiente manera.

Primero, el ingeniero ejecuta el cambio deseado desde el panel de control web y aplica los cambios. En función del tipo de cambio realizado se envía un tipo u otro de mensaje.

Si el simulador recibe un mensaje de tipo fecha y hora, ejecuta el código necesario para establecer la fecha y hora a la seleccionada, usando una estructura de datos que almacena el tiempo en formato UTC, como se muestra en la Ilustración 62

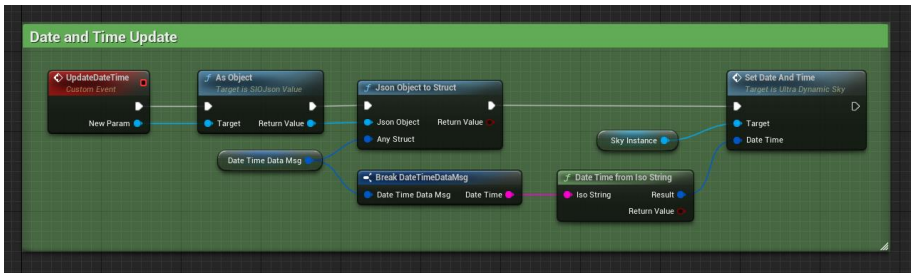


Ilustración 62. Cambio de fecha y hora

En el caso de la climatología, esta necesita hacer una transformación de mensaje plano a un objeto propio del *plugin* para su correcto funcionamiento. Si bien se han intentado alternativas, solo queda como opción la transformación manual de estos datos, visible en el medio de la Ilustración 63, quedando un código un tanto repetitivo.

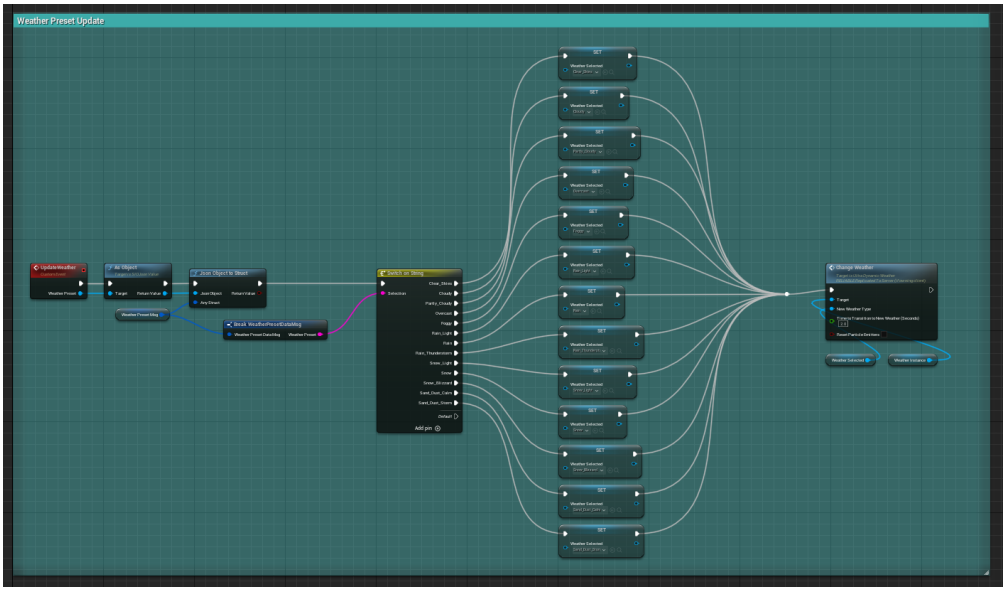


Ilustración 63. Cambio de climatología

5.6.9. Implementación de la señal de video

Para la señal de video del simulador se ha implantado un sistema one-to-many usando el protocolo de transmisión de video *WebRTC*. Un cliente emite señal de video mientas que puede haber de 1 a n espectadores simultáneos, estando limitado n a el máximo que pueda soportar la conexión del sistema.

Para ello, se ha usado el servidor como intermediario en las comunicaciones. Esta arquitectura de *streaming* de video la cual denota el comportamiento de la conexión entre servidor y clientes se llama SFU (*Selective Forwarding Unit*). En SFU los clientes se conectan directamente al servidor, en vez de entre ellos como WebRTC está planteado originalmente, de manera que el servidor pueda enrutar la señal de video hacia todos los clientes.

Otras arquitecturas (mostradas en la Ilustración 64) como *Mesh* hacen que cada cliente mantenga una conexión con el resto de los clientes, lo cual es una solución poco escalable. Por otro lado, la arquitectura MCU, si bien es similar a *SFU*, se diferencia en que el servidor procesa la señal de video y la reenvía a los clientes, mientras que *SFU* simplemente funciona como un enrutador de la señal.

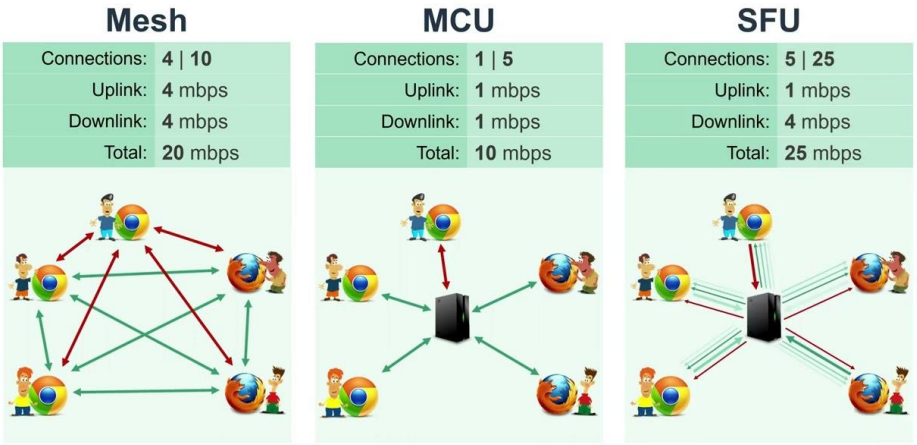


Ilustración 64. Comparación de arquitecturas de video

El protocolo de conexión y envío de la señal se puede ver desde dos perspectivas: El emisor y el receptor.

El emisor del video es en este caso el Launcher del simulador 3D. Este, se encarga de capturar la pantalla en forma de señal de video y enviarla al servidor. La captura de video no se hace directamente desde *Unreal Engine* debido a limitaciones con la VR y la integración de *WebRTC* que este tiene, por tanto, se captura la pantalla desde una aplicación externa obteniendo el mismo resultado.

Para realizar esta captura debemos en primera instancia hacer uso de la *API desktopCapturer* de *Electron.* Esta permite recuperar las pantallas disponibles y su *ID*, el cual será necesario para la obtención de la señal de video mediante la *API* Web *mediaDevices*. En la Ilustración 65 se puede este procedimiento, especificando la resolución y opciones de captura

```
const getStream = async () => {
  const sourceId = await ipcRenderer.invoke(channels.GET_VIDEO_SOURCE);
  const tempStream = await navigator.mediaDevices.getUserMedia({
    audio: false,
    video: {
      mandatory: {
        chromeMediaSource: "desktop",
        chromeMediaSourceId: sourceId,
        minWidth: 1920,
        maxWidth: 1920,
        minHeight: 1080,
        maxHeight: 1080,
      },
    },
  });

  return tempStream;
};
```

Ilustración 65. Obtención de la señal de video

Una vez disponemos de la señal de video, se realiza una conexión peer-to-peer con el servidor. Esta se establece realizando las comunicaciones que dicta el protocolo de *WebRTC* al *endpoint* de la *API* "/broadcast". Una vez se completa este proceso, la conexión se ha establecido con el servidor y la señal de video es enviada directamente a este. Es procedimiento es visible en la Ilustración 66

```
const handleNegotiationNeeded = async (peer) => {
  const offer = await peer.createOffer();
  await peer.setLocalDescription(offer);

  const payload = {
    sdp: peer.localDescription,
  };

  const { data } = await axios.post(
    process.env.REACT_APP_SERVER_URL + "/broadcast",
    payload
  );
  const desc = new RTCSessionDescription(data.sdp);
  peer.setRemoteDescription(desc).catch((e) => console.log(e));
};
```

Ilustración 66. Conexión del cliente con el endpoint "/broadcast"

Por parte del cliente que visualiza la señal de video se realiza un proceso similar. Primero debemos de establecer la conexión con el servidor mediante el *endpoint* de la *API* de "/video-signal". Se realiza los envíos pertinentes según marca el protocolo, y se establece como fuente del video de la interfaz el *streaming* recuperado. Esta implementación se puede ver en la Ilustración 67.

```
// Peer on track event
const handleOnTrack = (event) => {
  videoRef.current.srcObject = event.streams[0];
  setIsConnected(true);
};

// Peer negotiation needed event
const handleNegotiationNeeded = async (peer) => {
  const offer = await peer.createOffer();
  await peer.setLocalDescription(offer);

  const payload = {
    sdp: peer.localDescription,
  };

  const { data } = await axios.post(
    process.env.REACT_APP_SERVER_URL + "/video-signal",
    payload
  );
  const desc = new RTCSessionDescription(data.sdp);
  peer.setRemoteDescription(desc).catch((e) => console.log(e));
};
```

Ilustración 67. Conexión del cliente con el endpoint "/video-signal"

5.6.10. Implementación del dispositivo hardware

Para la integración de dispositivos *hardware* de tipo *joystick* o volante con Unreal Engine es necesario tomar una serie de acciones. Si bien el motor gráfico facilita la integración de estos mediante el *plugin Raw Input*, es necesario una configuración dependiente del volante que se desea usar.

En el caso de este proyecto, se ha desarrollado entorno al volante del cual se disponía, el *Thrustmaster T300 Alcantara*. Para configurarlo de forma correcta con *Raw Input* es necesario seguir los siguientes pasos.

Primero, debemos de consultar el *Vendor ID* y *Product ID* de nuestro volante, único para cada fabricante y modelo de dispositivo. En este caso, el *Vendor ID* de Thrustmaster es **044F** y el *Product ID* es **B66E.** Introducimos los datos en los ajustes del *plugin,* como se observa en la Ilustración 68

Ilustración 68. Identificación del volante en Raw Input

Ahora, debemos asociar los ejes y botones del volante a la contraparte del motor gráfico. Esto es debido a que, distintos volantes pueden tener distintas asignaciones o nombres de ejes. De esta manera, se asocian los ejes de diferentes volantes todos a un mismo *input*, facilitando la generalización del código donde se usen estas entradas. La asignación realizada a los ejes del volante seleccionado se puede ver en la Ilustración 69

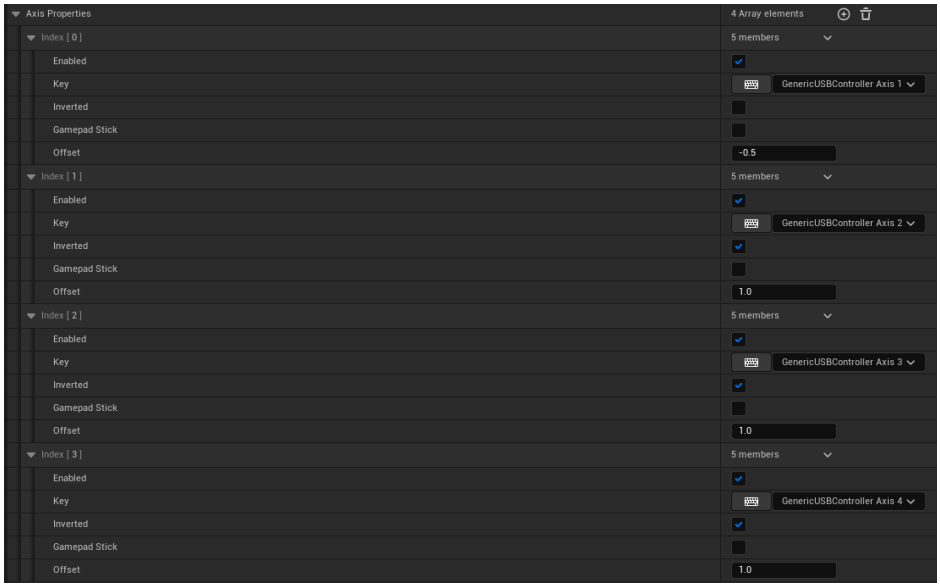


Ilustración 69. Configuración de ejes del volante

5.6.11. Implementación de la comunicación en tiempo real

La comunicación en tiempo real se realiza mediante el uso de *WebSockets* a través de la biblioteca *Socket.io.* Esta permite el intercambio de información entre clientes y servidor de forma instantánea, que, debido a la naturaleza de este proyecto, esta inmediatez es idónea. En el siguiente punto se denotan los aspectos claves del sistema desarrollado para el intercambio de información mediante este método.

Toda comunicación se realiza entre clientes y servidor, siendo el ultimo el que se encarga de redistribuir la información que recibe de un cliente al resto.

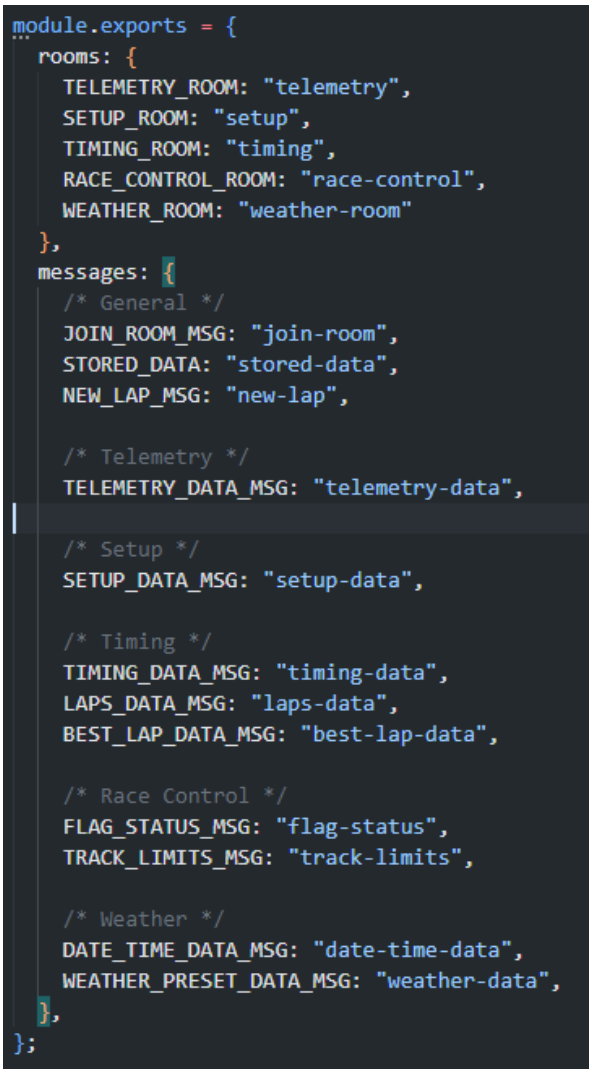
Por cada subsistema se crea una conexión de *sockets*, es decir, si un ingeniero se conecta al panel de control web se crea un socket para la telemetría, otro para los reglajes, otro para los tiempos de la sesión, etc. Esto permite separar cada *socket* en un canal distinto de manera que solo recibe los mensajes que este debe recibir (de telemetría, de tiempos, etc.). De esta manera se ahorra el computo consumido en la distinción de los mensajes y la decisión si es del tipo deseado o no.

Cuando un *socket* se conecta al servidor, este le pide al mismo que le introduzca en el canal deseado, realizando lo propio el servidor. La petición del cliente se muestra en la Ilustración 70.

```
// Change room
socket.emit(
  WSConstants.messages.JOIN_ROOM_MSG,
  WSConstants.rooms.SETUP_ROOM
);
```

Ilustración 70. Cambio de canal por parte del cliente

Los distintos canales y mensajes que pueden enviar los clientes se definen en un archivo de constantes que comparten el cliente y el servidor, de manera que sea sencillo aplicar cambios a la nomenclatura de los mensajes en un futuro si es necesario. Esto se puede observar en la Ilustración 71.



```
module.exports = {
  rooms: {
    TELEMETRY_ROOM: "telemetry",
    SETUP_ROOM: "setup",
    TIMING_ROOM: "timing",
    RACE_CONTROL_ROOM: "race-control",
    WEATHER_ROOM: "weather-room"
  },
  messages: {
    /* General */
    JOIN_ROOM_MSG: "join-room",
    STORED_DATA: "stored-data",
    NEW_LAP_MSG: "new-lap",

    /* Telemetry */
    TELEMETRY_DATA_MSG: "telemetry-data",

    /* Setup */
    SETUP_DATA_MSG: "setup-data",

    /* Timing */
    TIMING_DATA_MSG: "timing-data",
    LAPS_DATA_MSG: "laps-data",
    BEST_LAP_DATA_MSG: "best-lap-data",

    /* Race Control */
    FLAG_STATUS_MSG: "flag-status",
    TRACK_LIMITS_MSG: "track-limits",

    /* Weather */
    DATE_TIME_DATA_MSG: "date-time-data",
    WEATHER_PRESET_DATA_MSG: "weather-data",
  },
};
```

Ilustración 71. Constantes de mensajes y canales

Previo al cambio de canal, el nuevo *socket* recibe toda la información disponible en el servidor sobre la sesión actual, y almacena en sus variables internas las partes que le corresponden.

El cliente establece una función a invocar cuando recibe un mensaje del tipo deseado. Si es el cliente el que realiza algún cambio, este se lo comunica al servidor mediante el envío de un mensaje del tipo correspondiente, el cual contiene los datos modificados.

Por parte del servidor, este recibe todos los mensajes de los múltiples clientes, actualiza la información de la cual dispone en una variable local, realiza, si son necesarias, transformaciones a los datos, y reenvía los datos al resto de clientes, sin contar el cliente desde donde la ha recibido. En la Ilustración 72se puede ver este proceso, incluyendo transformación de datos, para el mensaje de tipo “Fecha y hora”

```
socket.on(wsConstants.messages.DATE_TIME_DATA_MSG, (data) => {
  console.log(
    "📅 Date & time data received: " +
      new Date(data.dateTime).toLocaleDateString("es-ES", {
        year: "numeric",
        month: "2-digit",
        day: "2-digit",
      }) +
      " - " +
      new Date(data.dateTime).toLocaleTimeString("es-ES", {
        hour: "2-digit",
        minute: "2-digit",
      })
  );

  // Store data
  sessionData.weather.dateTime = data.dateTime;

  // Emit to others
  socket
    .to(wsConstants.rooms.WEATHER_ROOM)
    .emit(wsConstants.messages.DATE_TIME_DATA_MSG, data);
});
```

Ilustración 72. Recepción de mensajes de fecha y hora por parte del servidor

5.6.12. Implementación de la concurrencia

El panel de control web permite la conexión y el trabajo de múltiples ingenieros al mismo tiempo. Esto supone un reto ya que, todos los ajustes que se pueden modificar desde el panel deben de permanecer sincronizados entre todos los clientes conectados.

Para ello se idea un sistema de escucha, envío y actualización mediante el uso de *React*, por parte del *frontend* y *Socket.io* por parte del *backend*. Este funciona como sigue.

Se supone la siguiente situación. Hay tres ingenieros (A, B y C) conectados al mismo tiempo al panel de control web. El ingeniero A decide cambiar el combustible que porta el vehículo de la sesión. Podríamos detectar este cambio y enviar un mensaje al resto de clientes. Esto, por cómo funciona *React* nos haría caer en un bucle infinito, ya que se lee y escribe sobre la misma variable, la cual se actualiza automáticamente en la vista, y al actualizarse en la vista, se volvería a enviar.

Por tanto, todos los cambios que haga un cliente no se envían hasta que no se presione un botón de "Aplicar cambios" que haga esto. De esta manera, el ingeniero A realiza los cambios oportunos y, al presionar este botón se leen las variables y se envían sus cambios, momento en el que los ingenieros B y C los reciben.

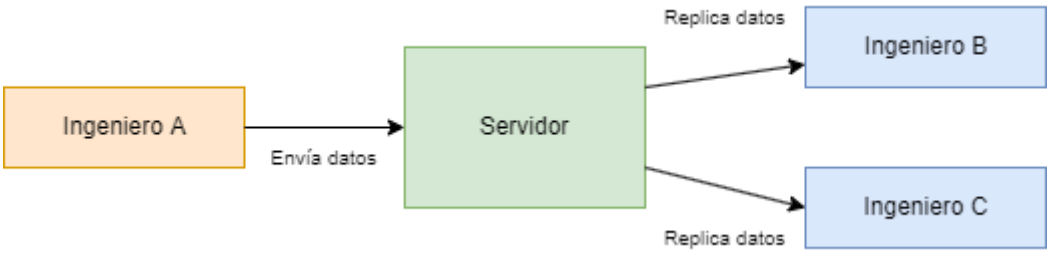


Ilustración 73. Diagrama de concurrencia de datos

5.7. Pruebas del sistema

A lo largo del desarrollo del sistema se han realizado pruebas incrementales para verificar el correcto funcionamiento de este. Estas pruebas, se han realizado a diferentes escalas: Desde pruebas de componentes individuales (unitarias) a pruebas de subsistemas de mayor tamaño. En el siguiente punto, se dan a conocer algunas de estas pruebas realizadas.

5.7.1. Pruebas unitarias

Se ha realizado una serie de pruebas a nivel unitario para la comprobación de sistemas de forma independiente. Por ejemplo, en el caso de *React* se han hecho pruebas de los componentes personalizados implementados. Un caso de prueba realizado es el siguiente que se muestra, aplicado sobre el componente "FlagIndicator.jsx".

- ¿Qué ocurre si el parámetro de entrada "flag" vale nulo?
- ¿Qué ocurre si el parámetro de entrada "flag" vale un valor conocido?
- ¿Qué ocurre si el parámetro de entrada "flag" vale un valor desconocido?
- ¿Qué ocurre si el parámetro de entrada "flag" es de tipo número?

Probando distintas situaciones a las que se puede ver expuesta la unidad y viendo el funcionamiento de esta bajo las mismas ayudan a perfeccionar la calidad del producto corrigiendo posibles errores que puedan surgir.

5.7.2. Pruebas de subsistemas

También se han realizado pruebas sobre subsistemas de mayor nivel. Por ejemplo, se han realizado pruebas sobre el subsistema de medición de tiempos (véase Ilustración 74). Haciendo uso de la aplicación *Postman* se han podido enviar datos correctos, inválidos, incompletos, desordenados, etc., de manera que sea posible el análisis de la reacción del sistema a estos datos.

Esta tarea se ve simplificada si se ha ejecutado unas buenas pruebas unitarias, ya que muchos de los posibles untos de fallos ya han sido probados y controlados

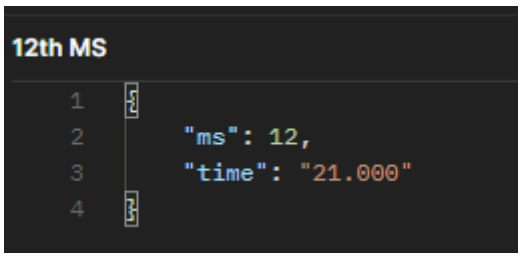


Ilustración 74. Petición de prueba de nuevo tiempo

6. Conclusiones

Tras el desarrollo del proyecto *VRacing*, en el siguiente punto se destacan las conclusiones a las cuales se ha llegado, bien desde el punto de vista del producto alcanzado como desde el punto de vista de la fase de desarrollo de este.

Inicialmente se propuso una serie de objetivos de tipo funcional los cuales se perseguían en el desarrollo de este proyecto. Esta lista de objetivos ha sido posible alcanzarla en su mayoría, no obstante, con algunos matices:

- Interacción entre viento y vehículo: Debido al modelo de dinámica de vehículos empleado en el desarrollo del proyecto, no ha sido posible la integración de efectos adversos del viento sobre el control del vehículo, ya que el modelo de dinámicas utilizado carecía de interfaces mediante las cuales poder ejercer los efectos que genera esta interacción.

- Varias señales de video de múltiples cámaras: Debido a limitaciones impuestas por el uso de la Realidad Virtual en el simulador 3D, solo ha sido posible sacar una señal de video, la cual, por su importancia, se ha elegido que sea la perteneciente a la visión del piloto.

- Integración de nuevo contenido en tiempo de ejecución: Inicialmente se planteaba la posibilidad de añadir nuevo contenido de circuitos y vehículos sin necesidad de modificar el proyecto de base. Esto se ha visto lastrado ya que, si bien existe une integración de un sistema similar, el cual podría haberse adaptado, para versiones anteriores de *Unreal Engine*, para su versión más reciente y usada en este proyecto, la número 5, no existe todavía un sistema adaptable compatible.

En lo que respecta a los objetivos de carácter personal, estos se han cumplido con éxito. Se ha puesto en uso el conocimiento de técnicas y herramientas conseguido en la realización de los estudios de grado y el aprendizaje e investigación de otras nuevas, como *React* y *NodeJS* en conjunto con las múltiples bibliotecas de código usadas.

También ha sido posible dar un enfoque al producto viable comercialmente y el desarrollo de un plan de negocio para este mismo, el cual cubre múltiples áreas (legales, económicas, financieras, etc.), todo enmarcado dentro de la convocatoria de TCUE.

Respecto a lo personal, ha sido una gran experiencia el desarrollo de este proyecto. El hecho de tener una oportunidad donde aplicar todos los contenidos recibidos durante el estudio de grado ha sido de gran ayuda para ver la aplicación del proceso de planificación y desarrollo de una aplicación *software* real. Añadido a esto, la posibilidad de realizarla de un ámbito del cual tengo una gran pasión, como es el mundo del *motorsport*, ha sido un gran aliciente a la buena experiencia de desarrollo.

También agradecer la tutorización por parte del tutor del proyecto aquí presentando, el cual, mediante su ayuda, ha aportado múltiples puntos de vista en diversos ámbitos a lo largo del desarrollo del proyecto.

7. Líneas de trabajo futuras

El proyecto desarrollado, *VRacing*, si bien cumple con todos los objetivos planteados inicialmente y se ha obtenido un producto completo y funcional, este puede seguir mejorando a través de la adición de nuevas características y mejora de las actuales. Por tanto, se plantean las siguientes líneas de trabajo y características para la mejora del proyecto aquí presentado:

- Incrementar la cantidad detalle del escenario 3D: En el proyecto desarrollado, el escenario 3D donde ocurre la simulación, el circuito, se ha implementado persiguiendo requisitos funcionales y no visuales, albergando este todo lo necesario para el correcto funcionamiento de los distintos subsistemas del producto, aunque visualmente este en gran parte vacío.

  Por tanto, se propone como futura línea de trabajo la inserción de objetos u otros elementos en el escenario 3D de manera que este gane más calidad visual.

- Mayor cantidad de vehículos y circuitos: Actualmente el proyecto, si bien está preparado modularmente para la inclusión de nuevos vehículos y circuitos, este solo contiene uno de cada tipo. Se plantea como nueva característica la adición de nuevos vehículos y circuitos al simulador.

  Además, sería de gran interés si estos se pudiesen integrar como módulos externos en tiempo de ejecución, sin tener que modificar el ejecutable original del simulador. Esto es lo que en el mundo de los videojuegos se denominan *mods*.

- Dinámicas de vehículos más extensas: Se plantea el desarrollo o implementación de un sistema de físicas o dinámica de vehículos más extenso, que sea capaz de simular mayor cantidad de variables del vehículo, de manera que se consigan resultados más fidedignos antes distintas situaciones

- Mayor cantidad de datos de telemetría: Actualmente, la cantidad de datos de telemetría que se pueden extraer está limitada por el modelo de dinámica de vehículos usado.

  Haciendo uso de un modelo de simulación de dinámica de vehículos más complejo y extenso, sería posible extraer mayor cantidad de datos de telemetría que denotasen el comportamiento del vehículo ante distintas situaciones, de manera que se pudiesen encontrar ajustes óptimos para el reglaje del vehículo.

- Gestión de usuarios: Actualmente no existe control sobre qué usuarios tiene acceso al panel de control del simulador y qué zonas de este pueden visualizar.

  Por ello, se plantea como línea de trabajo la implantación de un sistema de control de usuarios, de manera que permita la gestión de las cuentas de estos y sea posible llevar acabo un control de las áreas del panel de control que pueden visualizar cada uno.

- Almacenamiento en base de datos: Se plantea como futura línea de trabajo el almacenamiento de los datos generados por el uso del proyecto (tiempos, telemetría, etc.) en un almacenamiento de tipo permanente como las bases de datos, de manera que sea posible llevar un histórico de las sesiones y poder realizar comparaciones con sus datos.

- Sonido del vehículo: Muchas veces los pilotos usan el sonido que emite el motor de combustión del vehículo como referencia para saber el comportamiento del coche, cuando cambiar de marcha, etc. Se propone como línea futura de trabajo la adicción de un sistema de sonido que module el mismo en función de las revoluciones por minuto (RPM) del motor

Referencias

Circuit de Barcelona-Catalunya. (Junio de 2022). *Circuit de Barcelona-Catalunya.* Obtenido de http://premsa.circuitcat.com/2022/Web/Oficials/2022%20Manual%20del%20Oficial%20de%20Carrera%20(junio).pdf

Durán Toro, A., & Bernárdez Jiménez, B. (Octubre de 2000). *LSI Universidad de Sevilla.* Obtenido de http://www.lsi.us.es/docs/informes/lsi-2000-10.pdf

Grand View Research. (2021). *Grand View Research Website.* Obtenido de https://www.grandviewresearch.com/industry-analysis/simulation-software-market

Reese, L. (Septiembre de 2022). *Mouser*. Obtenido de https://eu.mouser.com/applications/automotive-racing-telemetry/

Tatuus. (Febrero de 2021). *Formula Regional by Alpine.* Obtenido de https://formularegionaleubyalpine.com/wp-content/uploads/jet-engine-forms/830/2021/04/TM_T318_4.00.pdf

TWI Global. (Septiembre de 2022). *TWI Global*. Obtenido de https://www.twi-global.com/technical-knowledge/faqs/faq-what-is-simulation

Unreal Sensei. (Abril de 2021). Obtenido de https://www.youtube.com/watch?v=3hmRN8bXMM0