UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Diseño y desarrollo de un juego de coches con Unity

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Castejón Soria, Javier

Tutor/a: Rey Solaz, Beatriz

Cotutor/a: Cerdá Boluda, Joaquín

CURSO ACADÉMICO: 2022/2023

Resumen

En este trabajo se muestra el desarrollo de un juego de conducción de coches en Unity combinando elementos de juegos de carreras arcade y de simuladores de conducción. El objetivo principal es el desarrollo del código que permite controlar el movimiento de un vehículo priorizando la jugabilidad del usuario sobre el realismo. El juego cuenta con unos controles simples, siendo posible controlarlo con las teclas del teclado, pero el movimiento del vehículo muestra también comportamientos propios de un juego de simulación de conducción como la suspensión, la pérdida de agarre de los neumáticos al girar o la degradación de los mismos. El juego cuenta con un circuito que presenta diferentes niveles de agarre haciendo así más desafiante la experiencia del jugador. El juego representa una combinación de los juegos de carreras arcade tradicionales empleando unos controles simples con los juegos de simulación ofreciendo así una experiencia de entretenimiento accesible para todos.

Resum

En aquest treball es mostra el desenvolupament d'un joc de conducció de cotxes a Unity combinant elements de jocs de carreres arcade i de simuladors de conducció. L'objectiu principal és el desenvolupament del codi que permet controlar el moviment d'un vehicle prioritzant la jugabilitat de l'usuari sobre el realisme. El joc compta amb uns controls simples, i és possible controlar-lo amb les tecles del teclat, però el moviment del vehicle mostra també comportaments propis d'un joc de simulació de conducció com la suspensió, la pèrdua de subjecció dels pneumàtics en girar o la degradació dels mateixos. El joc compta amb un circuit que presenta diferents nivells de subjecció fent així més desafiant l'experiència del jugador. El joc representa una combinació dels jocs de carreres arcade tradicionals utilitzant uns controls simples amb els jocs de simulació oferint així una experiència dentreteniment accessible per a tots.

Abstract

This work shows the development of a car driving game in Unity combining elements of arcade racing games and driving simulators. The main objective is the development of the code that allows to control the movement of a vehicle, prioritizing the user's playability over realism. The game has simple controls, being possible to control it with the keyboard keys, but the movement of the vehicle also shows behaviors typical of a driving simulation game such as the suspension, the loss of grip of the tires when turning or their degradation. The game features a circuit that presents different levels of grip thus making the player's experience more challenging. The game represents a combination of traditional arcade racing games using simple controls with simulation games thus offering an accessible entertainment experience for everyone.

Índice

I

Índice de figuras

Capítulo 1. Introducción

En el marco de este trabajo, propongo llevar a cabo un experimento o demostración del desarrollo del control de un juego de conducción. Al unir de manera sinérgica dos de mis mayores pasiones, el fascinante universo del automovilismo y la intrigante esfera de los videojuegos, aspiro a explorar cómo estas dos vertientes pueden converger de manera fluida.

La destreza necesaria para dirigir un vehículo virtual conlleva una interacción intrínseca entre el usuario y un dispositivo que implementa un programa específico, estableciendo un canal de comunicación mediante un periférico de entrada. El objetivo de la interacción al desarrollar un videojuego de conducción es evocar en el jugador emociones y sensaciones que influyan de alguna manera en su manera de relacionarse con el entorno digital. La gama de vivencias que se desprenden de esta experiencia es sumamente diversa, hincando sus raíces en la jugabilidad, un término que engloba la totalidad de las vivencias del jugador a lo largo de su interacción con el juego.

Diversos son los géneros de videojuegos en los cuales el desplazamiento del protagonista desempeña un rol fundamental, ya sea en los juegos de plataformas o en los "shooters", entre otros. No obstante, en el género de conducción, el movimiento del vehículo adquiere una preeminencia insoslayable, consolidándose como el pilar angular de la interacción, de tal manera que la jugabilidad se encuentra íntimamente entrelazada con dicho movimiento del vehículo.

Este es el motivo sustancial que otorga un protagonismo sobresaliente a la programación del conjunto de reglas físicas que confieren vida al universo plasmado en la interacción. De hecho, la manera en que las leyes físicas operan en relación con el control del vehículo ejerce un impacto significativo en la experiencia del usuario. A partir de esta premisa, es plausible lograr una experiencia que sea accesible y amigable para todos, o bien, en un extremo opuesto, forjar un auténtico desafío que ponga a prueba incluso a los pilotos de competencias automovilísticas reales.

Para lograr una adecuada implementación de los controles de forma que sean mínimas las fricciones entre el usuario y el programa, es importante conocer como estos sistemas interactivos influyen en los individuos. El estudio de este campo se realiza bajo el nombre de teoría de la interacción humano-computadora.

1.1 Teoría de la interacción humano-computadora (IHC)

La interacción humano-computadora es la disciplina que se dedica al estudio, evaluación y desarrollo de los sistemas informáticos que son usados por humanos. Tiene como objetivo disminuir los errores y la frustración causada por la interacción y lograr que esta sea más fluida y efectiva. Con este fin se observan por un lado las características de sistemas informáticos empleados y por otro lado cuál es el funcionamiento del lenguaje y psicología humana en este contexto y el impacto que estos sistemas tienen sobre nuestro cerebro.

Aunque a menudo se asume que la interacción humano-computadora hace referencia solo a la relación de un individuo con un sistema informático, esto no es así. La IHC se ocupa también de la interacción grupal y las dinámicas sociales que derivan del uso colectivo de la tecnología. [1]

La teoría de la interacción humano-computadora se trata por tanto de una ciencia con raíces en diversos campos de la investigación y el desarrollo científico como son la psicología cognitiva, la ingeniería de software, la ingeniería industrial, las ciencias de la comunicación y la sociología, entre otros. [2]

En este contexto importante comprender que el diseño de la interacción va a estar siempre centrado en el usuario. Esto implica conocer cuáles son las necesidades de los usuarios y saber cuáles son sus preferencias, de forma que se pueda efectuar un desarrollo de esta interacción de forma que atienda a conceptos como la facilidad de uso y la estética del mismo [3]. También se

precisa atender a dos cuestiones fundamentales que son la usabilidad y la experiencia del usuario. La usabilidad compete al conjunto de características que dotan al sistema interactivo de la capacidad para ser usado de un modo eficiente y seguro y útil de forma que el usuario final pueda aprender a usarlo y retener la información aprendida. Por otro lado, cuando se evalúa la experiencia del usuario en el proceso de interacción se hace referencia a lo gratificante que es para el usuario final hacer uso de esta tecnología, poniendo el foco en que este uso sea satisfactorio, creativo, emocionante o divertido [4]. Por lo tanto, según cual es la finalidad de la interacción que se esté desarrollando, se atenderá al concepto de usabilidad y experiencia del usuario de forma distinta para satisfacer las necesidades del usuario en cada caso.

Tradicionalmente se diseña un sistema informático el cual necesita mostrar una información a quien lo está utilizando usando luces, pantallas y timbres o altavoces. Esto significa que el canal a través del cual se establece la comunicación es a través de estímulos visuales y auditivos, aunque existen otros menos convencionales.

Atendiendo al caso de los videojuegos, la interacción se efectúa al haber información que la computadora comunica al usuario e información que el usuario comunica de vuelta al sistema manteniendo los criterios de usabilidad y procurando una buena experiencia del usuario, aunque esta comunicación puede tomar muchas formas distintas.

1.2 Interacción en los videojuegos

Los videojuegos cuentan con una de las industrias más pujantes de las últimas décadas y no se espera que este crecimiento se detenga pronto. Según el Foro Económico Mundial, se estima que este crecimiento siga y que el mercado de los videojuegos en su conjunto alcance una valoración de 321.000 millones de dólares en 2026 [5]. Este crecimiento no solo comprende las ventas de software sino también de hardware, las cuales siguen aumentando [6]. Este crecimiento ha fomentado la inversión privada, favoreciendo la innovación y el abaratamiento de tecnologías cada vez más asequibles para gran parte de la población.

A través de la investigación en tecnología a lo largo de los años se han desarrollado otro tipo de aparatos que logran el fin da transmitir información de otras formas menos convencionales. Este es el caso de lo que se conoce como tecnología háptica, que consiste en el uso del sentido del tacto para transmitir información como es el caso de la tecnología “force-feedback” o “motion simulators”, los cuales emplean motores que generan fuerzas para completar la experiencia de relacionarse con el entorno que está siendo simulado haciendo uso de la propiocepción y el sistema vestibular. [7]

Figura 1"Motion simulator" fabricado por la empresa *Cruden* [8]

Estas dos tecnologías mencionadas se emplean sobre todo en simuladores de conducción y pilotaje de vehículos y naves y se tienen en cuenta en el proceso de diseñar la interacción del usuario con el control del vehículo. Se conoce como “force-feedback” un sistema para simulación de conducción empleando volantes u otros objectos con motores integrados que generan las

fuerzas necesarias para causar en el usuario la impresión de estar conduciendo un vehículo real. La comunicación es en doble sentido, ya que el dispositivo genera fuerzas para transmitir información al usuario y este de mover el dispositivo para responder. Esta tecnología es capaz de simular por ejemplo el efecto del coche derrapando, pasando por un bache o de un avión volando [9]. El "motion simulator" por otro lado, consiste en un mecanismo que recrea la sensación de estar en un entorno móvil real, capaz de simular hasta seis grados de libertad como el que se muestra en la figura 1. [7]

Cabe destacar que su uso transciende la mera diversión y se emplean también en el entrenamiento de los pilotos profesionales de carreras como los pilotos de Fórmula 1, en la investigación científica, para estudiar la conducta humana en el tráfico y el diseño de carreteras [10] y en el entrenamiento de pilotos de aeronaves [11] como el simulador de Boeing 737 que aparece en la figura 2. De esta forma se logra abaratar los costes del entrenamiento de los individuos y reducir el coste de dicho entrenamiento, aunque el grado de realismo que se puede alcanzar es limitado.

Figura 2 Simulador de conducción de Boeing 737 [11]

Los métodos o el canal a través del cual el usuario se relaciona con el juego de conducción han ido cambiando con el paso del tiempo, el avance de esta industria y abaratamiento de la tecnología.

1.3 Historia de los videojuegos de conducción

La industria de los juegos de conducción cuenta con una larga historia la cual se remonta a la década de 1940 con los juegos electro-mecánicos. En ese momento no existían lo que hoy entendemos por maquinas arcade y los juegos consistían en artilugios mecánicos como el pinball.

Por ejemplo, existe un juego de carreras mecánico que se conoce por el nombre de "Yatch Racer" el cual fue fabricado por la compañía *Automátic Sports Company* con sede en Londres en el año 1900 y cuenta con unos barcos de juguete con unas cuerdas las cuales se estiran y contraen para hacerlo avanzar. En la figura 3 se puede observar la caja en la que viene empacado el juego el cual es considerado el primer juego de carreras arcade de la historia. [12]

Figura 3 Yatch racer [13]

1.1.1 Juegos de conducción electro-mecánicos (1941 - 1976)

Las salas de máquinas recreativas se remontan hasta finales del siglo XVII en Europa, pero no fue hasta la década de 1940 que se comenzaron a juntar los conceptos de las máquinas recreativas con los componentes digitales como botones transistores, resistencias y luces. En 1941 la empresa norteamericana *International Mutoscope Reel Company* fabricó "Drive Mobile" que ubicaba al jugador en el centro de un cilindro con la imagen de un circuito visto desde arriba que rotaba y el jugador trataba de mantenerse dentro del circuito durante un minuto mediante la rotación del volante que incluía, como se puede apreciar en la figura 4. [14]

Figura 4 Máquina "Drive Mobile" [15]

En la figura 4 se puede observar una máquina de "Drive Mobile" en la cual el único elemento interactivo con el cual el usuario se puede comunicar con el juego es un volante. Este elemento que en una primera instancia se mostraba tan simple sería el primer paso de una constante evolución

Este juego supuso el inicio de una era de máquinas arcade dispuestas como cajas verticales y siguió avanzando y evolucionando. Las ilustraciones movidas por motores se sustituyeron por pantallas y los zumbadores por altavoces. Uno de los últimos títulos de este género que conto con mucho éxito fue "F-1", fabricado por *Namco* y distribuido por *Atari* en 1976.

1.1.2 Juegos de conducción "top-down" 2D (1972-1988)

En 1972 la tecnología había avanzado lo suficiente como para crear nuevas máquinas más capaces para ejecutar juegos más complejos y con la creación de *Atari* por el norteamericano Nolan Bushnell, juegos como "Pong" fueron posibles. Aparecieron videoconsolas como *Magnavox Odissey* que eran capaces de ejecutar juegos como "Wipeoput" donde el jugador es un punto que se mueve alrededor de una pista y es considerado el primer videojuego de carreras de la historia. [16] [17]

Figura 5 Gameplay de "Wipeout" [17]

En la figura 5 se puede observar como se muestra el juego. La pantalla reproduce un punto el cual representa al jugador y el circuito que debe seguir es una imagen que se coloca encima del televisor.

Juegos como este fueron una inspiración para el desarrollo de nuevos e innovadores videojuegos arcade como "Rally-X" publicado en 1980 por *Namco* el cual, como se puede observar en la figura 6 incluye un desplazamiento del vehículo por la pantalla tanto en el eje vertical como en el horizontal, un mini mapa que ubica al jugador en el mapa global e incluso música de fondo. [18]

Figura 6 Captura de pantalla de "Rally-X" [18]

Mientras algunos juegos alcanzaban niveles de complejidad más altos en cuanto a la jugabilidad, otros equipos se centraban en aspectos del desarrollo relacionado con la apariencia y calidad gráfica consiguiendo simular la sensación de profundidad y de movimiento en un universo tridimensional en un momento en el que la capacidad de generación de gráficos de los procesadores permitía únicamente gráficos bidimensionales. Los videojuegos en los que se aplican este tipo de técnicas se conocen como juegos pseudo-3D.

1.1.3 Juegos de conducción pseudo-3D (1976-1992)

Los juegos pseudo-3D supusieron un avance en la experiencia del usuario significante. En algunos casos, como en la máquina arcade "Road race", la revolución de la experiencia del usuario no solo venía por parte de unos gráficos con sensación de profundidad, sino que también incluía un elemento de feedback háptico, es decir que hace uso del sentido del tacto. Este consistía en emplear el manillar que se puede observar en la figura 7 para controlar el vehículo, el cual tiene la capacidad de vibrar en el momento en que el jugador colisione contra otro vehículo.

Figura 7 Máquina "Road Race" [12]

El escalado de *sprites* o las imágenes que representan un personaje u objeto supuso un avance tecnológico importante para la industria de los videojuegos ya que mediante transformaciones aplicadas a estas imágenes era posible crear el efecto de profundidad de una forma mucho más sencilla que hasta el momento. Esta técnica logro implementarse por primera vez en la máquina arcade llamada "Turbo", fabricada por Sega en 1981 [19].

Uno de los títulos más representativos de esta época fue el juego arcade llamado "Pole position" fabricado por *Atari* en 1982. Este estaba fuertemente inspirado en "F-1" y supuso un avance importante en el mundo de los videojuegos, considerado por IGN como el juego de carreras más influyente de toda la historia. [20] Fue el primer juego en construir una pista virtual basada en la trazada de un circuito real. Además, antes de cada carrera el jugador tenía que realizar una vuelta en solitario y el tiempo que obtuviese determinaría su posición en el inicio de la carrera, por lo que fue también el primer juego en contar con una vuelta de calificación [21]

Figura 8 Folleto publicitario de "Pole Position" [22]

En la figura 8 se puede observar como a través de reescalar sprites bidimensionales y aplicarles una transformación se logra transmitir la sensación de profundidad.

Por otro lado, el juego conocido como REVS supuso un gran cambio en la industria del momento, ya que su creador, Geoff Crammond lo desarrolló para que pueda ejecutarse en un ordenador doméstico, concretamente el BBC Micro. Publicó sin saberlo el que sería el primer intento de juego de simulación de carreras para ser jugado en casa. No es considerado un simulador de conducción porque no se cumplen los requisitos de usabilidad necesarios para que la interacción con el control del vehículo muestre un nivel de realismo suficiente. No obstante, el control del vehículo era más detallado que en otros juegos de su época, y además contaba con cinco circuitos de la Fórmula 3 inglesa, entre ellos, el circuito de Silverstone. [23]

Toda la industria de las máquinas arcade de conducción dio otro salto de desarrollo en la década de los 80, cuando se popularizó el uso de los "motion simulators" o simuladores de movimiento a través de los juegos "taikan", que es un término japonés empleado para referirse a las sensaciones corporales. Fueron dos máquinas arcade de Sega las que supusieron un mayor salto cualitativo en este aspecto. Primero publicaron "Hang on" en 1985, un juego basado en carreras de motos en el cual el jugador se coloca encima de un objeto con forma de moto que cuenta con un motor que lo hace moverse de forma coordinada con el juego, y en el cual el jugador debe mover la moto para controlar su personaje. El otro juego conocido como "Out Run", publicado en 1986 recogía la misma idea del "motion simulator" que ya había tenido éxito en su anterior entrega, pero esta vez con forma de coche. Además, este cuenta con un innovador sistema de "force-feedback" en el volante.

Figura 9 Máquina de "Out Run" [24]

En la figura 9 se observa un ejemplar de “Out Run” y al compararlo con los simuladores de conducción tanto anteriores como posteriores sorprende la cantidad y calidad de elementos interactivos que forjan la usabilidad. De los que se fabricaron, no todos contaban con esta tecnología, pero realmente fue avanzado a su época puesto que hoy en día se siguen diseñando simuladores de conducción que cuentan con “force-feedback” y “motion-simulator” aunque la capacidad de transmitir información verosímil no estaba al alcance de la tecnología de la época y por lo tanto recreativas como esta no son consideradas simuladores de conducción, aunque se seguiría mejorando en este aspecto.

Mientras unas compañías de videojuegos se esforzaban por traer a la industria nuevas innovaciones en el aspecto técnico que mejore la experiencia del jugador, el mundo de las videoconsolas domésticas estaba ganando una gran popularidad de forma paralela a partir de la década de los 80. A pesar de ver la luz años más tarde, “Super Mario Kart” fue un videojuego publicado por *Nintendo* en 1992 para la videoconsola “Super Nintendo Entertainment System” el cual aunque técnicamente supuso un avance por emplear “modo 7” para simular un entorno 3D gráficamente se mostraba muy similar a los títulos que años antes ya se habían podido jugar en las salas de recreativas. La innovación, en este caso, venía por parte de la jugabilidad ya que este título permitía jugar con los diferentes personajes de la franquicia de “Super Mario”, jugar contra otra persona con la pantalla dividida y añadió la mecánica de poderes y objetos especiales la cual posibilitaba al jugador atacar y defenderse de otros jugadores y añadía un elemento de estrategia y emoción a las carreras. Estos conceptos serían una referencia para la industria y sentaría un precedente que aun hoy en día sigue vigente.

Figura 10 Modo batalla de "Super Mario Kart" [25]

En la figura 10 se observa el aspecto que tenía una partida multijugador en “Super Mario Kart”, y si analizamos la innovación del lanzamiento en términos de experiencia de usuario se puede concluir que esta se ha visto beneficiada de un planteamiento del diseño del juego que se aleja del realismo de otros títulos y abraza un enfoque más creativo y alocado.

1.1.4 Juegos de conducción 3D y aparición del género de simulador de conducción (1988 - 1995)

Llegó un momento en que al diseñar estas experiencias interactivas se contó con el hardware que estaba extendido en el mercado de las máquinas recreativas de conducción y con la capacidad técnica de ejecutar software más complejo que permita opciones de desarrollo distintas. En algunos casos, se puso el foco en hacer que la sensación de conducción fuera lo más realista posible por lo que la usabilidad sería determinada por la capacidad de esta interacción de replicar de forma fiable y eficaz la experiencia de conducir un vehículo real en lugar de simplemente ser divertida para el usuario.

Figura 11 Folleto publicitario de "Hard Driving" [26]

Paralelamente a la creación de los primeros simuladores de conducción propiamente dichos, se desarrollaban otros juegos que, aunque se acercaron a la idea de la simulación se centraron en mejorar la experiencia del usuario en lugar de tratar de replicar el funcionamiento de vehículos reales de una forma eficaz. Por ejemplo, "Hard driving", cuya máquina arcade se aprecia en la figura 11, publicado por *Atari* en 1989, es un título que cuenta con gráficos basados en polígonos tridimensionales y el cual emplea un volante con "force feedback" capaz de generar fuerzas contrarias al movimiento del usuario más fuertes cuanto mayor es la intensidad del giro que se está tratando de realizar.

Otro título que supuso un avance tecnológico es "Virtual Racing" producido por Sega en 1992. Este contaba con gráficos 3D que lucían muy avanzados tanto en primera como en tercera persona. El elemento innovador que posee este juego reside en la capacidad de jugar carreras en modo multijugador entre varias máquinas las cuales deben estar conectadas entre sí. Es por este motivo que es considerado el tercer juego de conducción más influente según la lista IGN. [20]

Atendiendo a los criterios de usabilidad, como la capacidad de esta interacción de ser confiable y que se pueda aprender de ella, existe un juego del cual se puede extraer un notable avance en el desarrollo de experiencias interactivas y supondría el nacimiento del género de simulación de conducción en el mundo de los videojuegos o "SimRacing" para abreviar. [27]

Este título fue publicado en 1989 por una empresa llamada *Papyrus Design Group* con el nombre de "Indianápolis 500: The Simulation", el cual pasaría a ser el primer simulador de conducción y además se ejecuta en un ordenador personal. El juego simula las físicas del control del vehículo muy realistas para su tiempo ya que el diseño de estas se realizó basándose telemetría de los vehículos de la competición "Indy 500" reales. De esta forma se consiguen replicar efectos físicos

como la pérdida de agarre al realizar una curva demasiado rápido lo que lleva al usuario a querer tomar una línea más optimizada. [27] "IndyCar Racing" sería el sucesor de este juego publicado por la misma compañía en 1993.

Figura 12 Captura de pantalla de "Formula One Grand Prix" [29]

Ante la revolución del nivel de interacción mostrado por los productos de *Papyrus Design group* otros juegos salieron replicando este mismo planteamiento de desarrollo de experiencias basadas en las físicas de vehículos reales, pero adaptándolas a otras competiciones como son el caso de "Formula One Grand Prix" de *Micro Prose* en 1991 y "Rally Championship" de *Sega* en 1995. El primero se muestra en la figura 12, incluye los circuitos, pilotos y colores de la parrilla de la Fórmula 1 y se ejecutaba a unos sorprendentes 25 fotogramas por segundo mientras que el segundo se centra en el rally y simula la experiencia de pilotar un coche sobre diferentes superficies las cuales transmiten distintos niveles de agarre al vehículo.

1.1.5 Juegos de conducción modernos (1996 - presente)

En el inicio de la década de los 90 *Sega* y *Namco* poseían prácticamente el monopolio de las maquinas arcade de juegos de carreras con gráficos 3D pero la competencia por parte de las videoconsolas domesticas era cada vez más notable. A mitad de los años 90 la quinta generación de videoconsolas salió a la luz, con consolas como la "Sega Saturn", la "PlayStation" de *Sony* y la "Nintendo 64". A partir de este punto, la mejora gráfica de los videojuegos con el paso del tiempo se convertiría en una constante y las diferencias más significativas entre los títulos vendrían de mano de la innovación en el contenido de los juegos y en el tipo de experiencia que se trata de ofrecer al jugador. De esta forma, los videojuegos de consola irían desplazando a las maquinas arcades tradicionales.

Figura 13 Captura de pantalla de "Mario Kart Wii" [30]

En 1996 *Nintendo* publicó "Mario Kart 64", juego que seguía los pasos de su predecesor en tanto que plantea un enfoque frenético y fantástico sobre la conducción, aunque en este caso también cuenta con gráficos 3D. La saga de "Mario kart" seguiría consolidándose en la industria de los videojuegos como un referente en el género de los juegos de conducción, de karting y de estilo arcade con un planteamiento casual y accesible para todo tipo de jugadores.

A partir del juego "Mario Kart Wii" que se puede ver en la figura 13 publicado en 2008 todas las entregas cuentan con la posibilidad de usar un sistema de "motion-control" que permite al jugador controlar el vehículo mediante el movimiento del control en el caso de la "Wii", "Wii U" y "Switch", de la consola en el caso de la "Nintendo 3DS" y del dispositivo móvil en el caso de "Mario Kart Mobile". Esta forma de interacción supone un avance en términos de experiencia del usuario ya que aporta nuevas formas de interacción con la consola que permiten una mayor inmersión. En el caso del mando de la "Wii" supuso además una revolución en cuanto a la usabilidad de la consola implica un rápido aprendizaje y una mínima dificultad para recordar el funcionamiento de esta. [31]

Figura 14 Captura de pantalla de carrera en "Gran Turismo" [32]

Otras franquicias de videojuegos de conducción se han desarrollado paralelamente como es el caso de “Gran Turismo” que podemos observar en la figura 14 y cuya primera entrega se publicó en 1997 para la consola “Play Station” tras haber estado 5 años en desarrollo. Cada juego de esta saga ha añadido nuevos elementos con la finalidad de lograr un mayor realismo y una gran variedad en los vehículos por lo que constantemente se ha posicionado como un referente para la industria de los juegos de conducción. “Gran Turismo” destacó por el detalle en el realismo de los coches y en físicas, así como por contar con un sistema de licencias en el cual el jugador había de superar unas pruebas para avanzar a través del juego, “Gran Turismo 3” aportaba la implementación de multijugador online, “Gran Turismo 4” posibilitó el uso de un volante como control [33], “Gran Turismo 5” incluyó cambios en la climatología y conducción nocturna y finalmente “Gran Turismo 7” cuenta con un gran soporte para la disputa de competiciones de “eSports”. “Gran Turismo” contó incluso con un programa de televisión de entrenamiento de jóvenes pilotos conocido como “GT Academy” el cual trataba de ofrecer la oportunidad de competir en carreras reales a los jugadores más habilidosos en el cual se basa la película “Gran Turismo” de Neill Blomkamp de 2023 que se muestra en la figura 15.

La mejoría de la usabilidad y experiencia del usuario ha sido tan intensa en todos los aspectos que esta franquicia de juegos de conducción ha terminado siendo la segunda más exitosa de la historia solo después de “Mario Kart”.

Figura 15 Poster de la película "Gran Turismo" (2023) [34]

Tras este repaso la historia de los juegos de conducción se puede apreciar la evolución de los métodos de interacción en los controles, pero también la diversificación en cuanto a los géneros en función del tipo de experiencia que los desarrolladores pretenden evocar en el jugador

1.4 Diferenciación de géneros de los videojuegos de conducción

Atendiendo al diseño de la jugabilidad, usabilidad y experiencia del usuario desde la perspectiva de los estándares actuales de la industria los juegos de coches se pueden clasificar en dos subgéneros, los de "conducción arcade" y los "simuladores de conducción". La principal diferencia entre los dos es que en el primero el funcionamiento de los coches, las físicas y los controles se ven moldeados por el diseño del juego con el objetivo de hacerlo divertido y dejando en segundo plano la verosimilitud de la experiencia mientras que los simuladores, como su nombre sugiere se esfuerzan a recrear la realidad de los vehículos de la manera más fiel posible.

1.1.6 Arcade

Gran parte de los juegos de coches que se han popularizado masivamente y que la gente conoce son juegos de conducción arcade. Títulos como "Mario Kart", "Need For Speed", o "Trackmania" son capaces de proporcionar la sensación de velocidad y la emoción por adelantar a los contrincantes manteniendo unos controles simples, accesibles y permisivos.

Una gran simplicidad en las físicas del manejo del vehículo logra facilitar la adopción de este género por buena parte de la población, como fue el caso de "Mario Kart DS" cuyo aspecto se puede observar en la figura 16. Esta facilidad para comenzar a jugar sin tener unos conocimientos previos de conducción ni un equipamiento especializado implica que la barrera de entrada para nuevos jugadores es baja, pero la profundidad que pueden adquirir los controles cuando el jugador desarrolla la habilidad del manejo le ofrece a este una curva de aprendizaje en la que se puede progresar indefinidamente, especialmente cuando se cuenta con incentivos para hacerlo, como la competitividad entre los jugadores.

Figura 16 Captura de una carrera en "Mario Kart DS" [35]

Trackmania es una franquicia de juegos de carreras cuenta con varias entregas publicadas a lo largo de los años, pero todas ellas cuentan con una característica común que los une y se ha convertido en una especie de seña de identidad de sus títulos y es su simplicidad en los gráficos y controles. En los títulos de trackmanía el jugador controla el vehículo usando tan solo cuatro

botones, para acelerar, frenar y girar. Esta simplicidad podría hacer entender a alguien ajeno a la franquicia que se tratan de títulos donde los controles son planos y los jugadores no son competitivos; nada más lejos de la realidad.

"Trackmania: Nations Forever" es un juego gratuito que se puede controlar usando solamente cuatro teclas del teclado y que hasta un portátil sin tarjeta gráfica integrada puede ejecutar. Sin embargo, a pesar de su aparente simplicidad, miles de personas siguen jugando a este juego cada día con el objetivo de mejorar sus tiempos y encontrar la trazada más optima. En la figura 17 podemos observar los gráficos aparentemente simples de este título.

Estos controles simples han sido una inspiración para la ideación de este proyecto ya que por un lado simplificaría la programación, la experiencia del usuario, pero no implica necesariamente una peor jugabilidad o usabilidad ya que la finalidad no es recrear fielmente una conducción real.

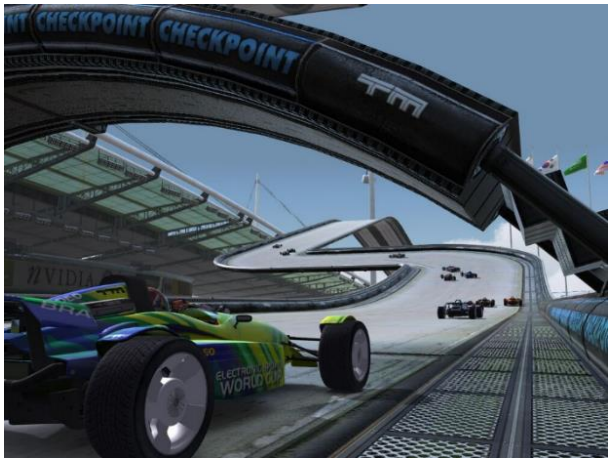

Figura 17 Captura de "Trackmania: Nations Forever" [36]

1.1.7 Simuladores de conducción

Son muchos los simuladores de conducción que hay disponibles en la actualidad ya que este subgénero de los juegos de carreras ha experimentado un constante crecimiento a lo largo de las últimas tres décadas. Dentro del subgénero de simuladores de conducción con estándares de usabilidad actuales los juegos cuentan con la posibilidad de usar volante y pedales para efectuar el manejo del vehículo y también es notable la intencionalidad de aportar una perspectiva realista, seria y competitiva a la experiencia del usuario.

No obstante, no todas las franquicias son iguales y existen diferencias entre estas. Algunas franquicias como la saga de videojuegos "F1", la cual está respaldada por la licencia oficial de la competición de Fórmula 1 y la *FIA*, cuentan con un modelo de control del monoplaza con un alto nivel de realismo empleando volante y pedales a la vez que ofrece la posibilidad de controlar el vehículo mediante un mando de "Playstation" o "Xbox". Esta variedad en los controles permite al juego ser atractivo para un mayor número de potenciales jugadores con un diverso nivel de habilidad. En términos de usabilidad se valora la capacidad del juego de adaptarse en cada caso a las necesidades y preferencias del jugador.

Además, aunque las últimas entregas de la saga de juegos de "F1" son consideradas simuladores de conducción debido a que la experiencia del usuario se centra en recrear un modelo de físicas realistas, la forma en la que el jugador interactúa con estas físicas también es personalizable. Al jugador se le ofrece la posibilidad de ajustar el grado en que las ayudas a la conducción intervienen, de forma que cuando los usuarios no desean enfrentarse a una experiencia de conducción realista o no cuentan con el equipamiento necesario para hacerlo, estas ayudas posibilitan una interacción simplificada donde partes de este control son automatizadas. A cambio de la reducción en la dificultad de la conducción, el vehículo se vuelve más lento ya que la forma en la que las ayudas a la conducción regulan el freno, la aceleración o el cambio de marcha no son óptimas para una conducción veloz.

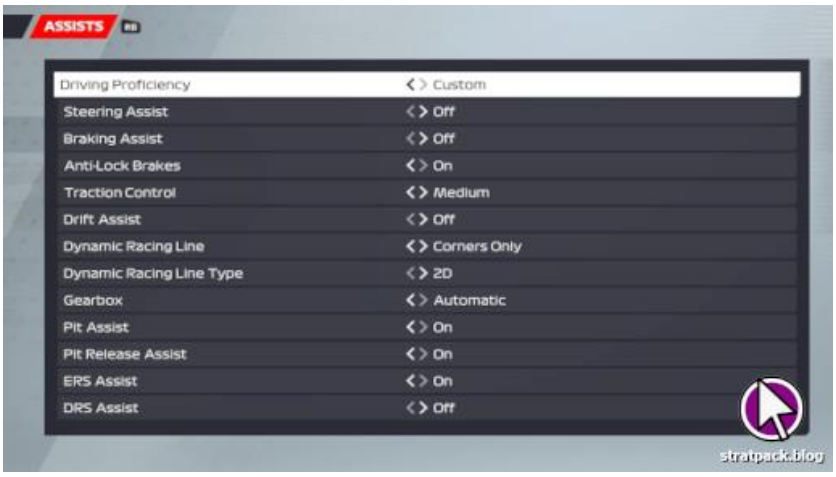

Figura 18 Asistencias de conducción de "F1 22" [37]

En la figura 18 se observa la ventana de opciones de asistencias a la conducción. Esta gran variabilidad de la experiencia del usuario diluye la división entre juegos de carreras arcade y simuladores y es el precedente sobre el cual este proyecto se basa para definir partes del diseño de la jugabilidad.

Existen otros títulos los cuales no ofrecen las mismas facilidades para adaptar el control del vehículo, sino que centran la jugabilidad en otros aspectos. "Asetto Corsa" los permite al jugador experimentar la conducción de una gran variedad de vehículos y circuitos posibilitando así que se pueda recrear casi cualquier tipo de carrera. Este juego cuenta además con una enorme

comunidad de *modders* que aportan gratuitamente casi cualquier vehículo que se quiera conducir a la experiencia. En la figura 19 se puede observar el aspecto realista que muestra "Asetto Corsa"

Figura 19 Imagen de la tienda de *Steam* de "Asetto Corsa" [38]

Y finalmente existe un grupo de juegos los cuales se desarrollan con el objetivo de crear un entorno de competición serio en el cual el usuario debe desarrollar una serie de habilidades propias de un piloto de competiciones reales. "Asetto Corsa Competizione" recrea la competición oficial de las competiciones GT3 y GT4 pero sin duda el simulador más exigente, realista y considerado por muchos el referente en su género es "iRacing".

Figura 20 Entrevista de CNN a Max Verstappen [39]

Títulos como "iRacing" simulan de forma efectiva muchos elementos propios del pilotaje real en carrera como la gestión de la temperatura y el desgaste de los neumáticos y los frenos, así como el dominio de la frenada y el giro del volante necesario para atacar una curva teniendo en cuenta la aerodinámica del vehículo, la distribución de pesos entre otras cosas. La búsqueda del tiempo más bajo en un circuito para acercarse a la perfección es el motor que lleva al usuario a pelearse con las mecánicas y físicas hasta que las comprende y se adapta a ellas. Pilotos profesionales como Max Verstappen quien aparece en la figura 20 han confesado jugar a estos simuladores no solo por afición sino como herramienta para mantener su mente y reflejos en forma e incluso han hecho aparición en competiciones virtuales oficiales y plantea crear un entorno en donde pilotos virtuales puedan dar el paso a competiciones de conducción reales.

Capítulo 2. Objetivos

Los objetivos del proyecto se detallarán a continuación de forma ordenada según el nivel de prioridad que ocupan en el proyecto

2.1 Objetivo general

El objetivo general es crear una experiencia interactiva a través de la cual el usuario pueda controlar el movimiento de un vehículo presionando teclas en el ordenador. Este primer objetivo se podría afrontar de muchas formas distintas, pero su consecución deberá satisfacer también el resto de los objetivos.

- Controlar el movimiento del coche de forma simple y generalista.

A la hora de plantear el desarrollo del código se priorizará la simpleza del mismo con el objetivo de facilitar tanto su programación como la implementación progresiva de nuevas funcionalidades. Un código fácil de modificar permite la adaptación del este según las necesidades que se tengan con relación al diseño.

- Conseguir un sistema de interacción donde la jugabilidad se potencie antes que en el realismo.

Previamente se ha mencionado la diferencia que hay entre los juegos de coches “arcade” y los simuladores de conducción. El proyecto que nos ocupa se plantea como un juego “arcade” de forma que la perspectiva que se toma para su diseño y desarrollo es la del usuario final y el funcionamiento de las físicas se diseñará a partir de ahí. Por ejemplo, el juego ha de poderse controlar usando las teclas del teclado.

- Modificar el movimiento del coche en función de algunos de los efectos físicos propios de simuladores de conducción.

En la introducción se ha mencionado como un juego no necesita necesariamente pertenecer a uno u otro género, sino que es posible que incluya elementos de ambos géneros. En este caso se buscará que el movimiento del vehículo muestre algunos de los comportamientos que usualmente se muestran en los simuladores de conducción como la existencia de la amortiguación, neumáticos que pueden no adherirse al suelo y producir un trompo al derrapar y degradación de los neumáticos, entre otros.

- Generar un circuito con más de una superficie con distintos niveles de agarre.

Se precisa de un circuito donde el vehículo pueda correr. Al añadir elementos que alteren la conducción en el circuito como variaciones en el nivel de agarre entre el interior y exterior del circuito se lograría implementar unos límites de pista de forma natural e integrada con el resto de la experiencia.

- Crear una interfaz de usuario que permita al jugador comprender qué está sucediendo por pantalla.

El usuario necesita saber qué está sucediendo en todo momento para poder desenvolverse bien y una interfaz gráfica que muestre datos básicos como el tiempo de vuelta o la velocidad podría ser suficiente para lograrlo.

Capítulo 3. Metodología

A continuación, se expone la metodología empleada en el desarrollo del proyecto dividiendo el proceso en tres etapas, una primera etapa de ideación, una segunda de planificación y finalmente el desarrollo.

3.1 Ideación

Antes de comenzar con este proyecto se me había propuesto la realización de trabajo de final de grado que consistía en una simulación en Unity de algunos fenómenos físicos que suceden en el rebote de un muelle. A mí no me convenció esa idea, pero me gustaba la idea de desarrollar una simulación física en un entorno de 3D. Comencé a hacer una lluvia de ideas sobre qué podría desarrollar que sirva como trabajo de final de grado, pero también que sea sobre una temática que realmente me atraiga. Finalmente me decanté por la conducción.

3.1.1 Entorno y software

Si uno quiere desarrollar un juego 3D tiene varias opciones a la hora de elegir un entorno para su desarrollo. En este caso empleo Unity porque es el entorno que más he usado durante mis estudios con este fin, aunque existen otros muy capaces como Unreal Engine que también hubiera posibilitado el desarrollo de este proyecto, aunque guardan algunas diferencias entre ellos, como el lenguaje de programación para el que están diseñados.

Figura 21 Logotipo de Unity

El desarrollo del juego se realiza en Unity, que nos permite crear entornos virtuales en tres dimensiones y simplifica el desarrollo del producto final. En este caso se completa el desarrollo en la versión de Unity 2023.1.11f1 la cual se ejecuta sin problemas en un ordenador portátil que cuenta con un procesador Intel i7 de 2017 y sin tarjeta de vídeo dedicada, por lo que merece la pena destacar el buen rendimiento que este programa tiene en los equipos. Unity funciona mediante escenas, y en las escenas se encuentran objetos que conforman el universo virtual para esa escena. Por ejemplo, si se desarrolla un juego que cuenta con un menú principal y dos niveles, generalmente, se emplea una escena para el menú y para cada nivel y luego en la ejecución del programa final este iría alternando entre escenas. En las escenas se crean objetos que pueden ser de distintos tipos y son altamente personalizables.

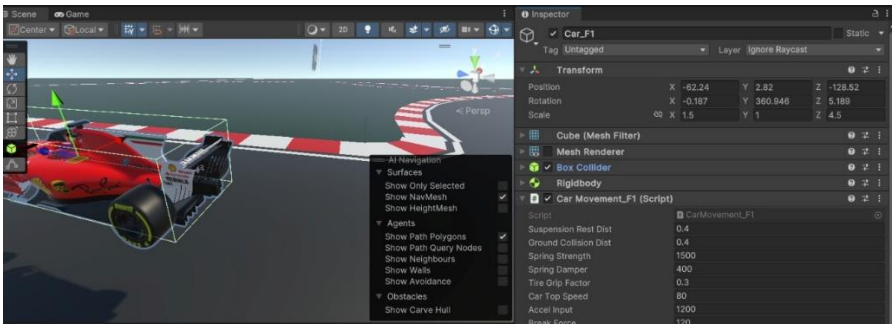

Figura 22 Imagen de los componentes del objeto

Los objetos contienen información de como estos objetos interaccionan con el resto del entorno y esta información la almacena en forma de "componentes" del objeto que se observan en el lado derecho de la figura 22, habiendo un componente que contiene la ubicación y el tamaño del objeto, otro que contiene la forma o el modelo 3D, otro para la textura, y también hay componentes que se encargan de definir parámetros físicos del objeto como la masa, la capacidad de colisionar con otros objetos y la forma en la que lo hacen. Estos componentes se pueden crear y eliminar para configurar el comportamiento del objeto en la escena, es decir, para que los objetos de la escena interaccionen entre sí de la manera deseada.

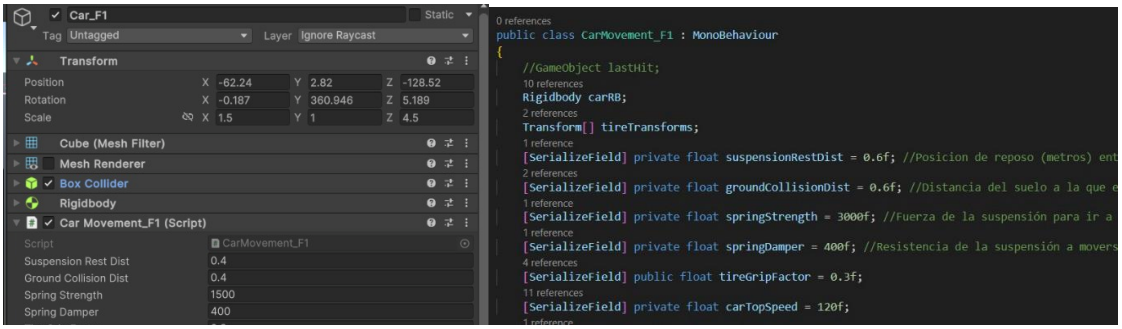

Figura 23 Objeto con componente de script

En Unity los objetos que creamos en una escena son altamente personalizables, pudiendo incluso introducir un componente de scripts que permite al objeto acceder a un script en C# y realizar las funciones que se han programado como se observa en la figura 23, consiguiendo de esta forma que las escenas de Unity puedan adquirir una gran complejidad. Mediante el uso de componentes de scripts se puede alterar la forma que tienen de interactuar los objetos de muchas formas distintas, ya que estos scripts pueden acceder a la información de los componentes de este u otros objetos, alterar esta información, y las teclas pulsadas por el usuario en el teclado u otros dispositivos. También posibilita generar fuerzas sobre los objetos de Unity, por lo que es posible a partir de la lectura de los "inputs" programar una serie de fuerzas sobre el coche que conforman el movimiento de un vehículo, creando así la interacción deseada. [40]

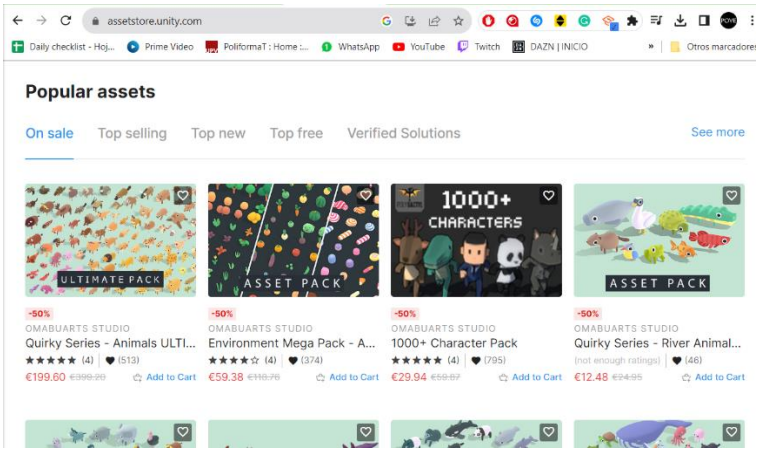

Figura 24 Tienda de *assets* de Unity [41]

En Unity los objetos se pueden crear a partir de formas básicas como cubos o esferas, pero también pueden existir objetos que importen imágenes, modelos 3D u otros tipos de recursos de otros entornos y se les denomina "asset". Una importante parte del proceso de desarrollo de videojuegos en Unity consiste en hacer uso de la tienda de "*assets*" que se muestra en la figura 24 donde usuarios de todo el mundo publican archivos que se pueden importar al proyecto

Para la elaboración del código se emplea "*Virtual Studio Code*" como editor de código, aunque cualquier editor que tenga soporte para la programación en C# podría ser usado indistintamente.

3.1.2 Dispositivos

En cuanto a los dispositivos requeridos para realizar el proyecto, una posibilidad es emplear un controlador de Xbox como dispositivo de entrada, pero con el objetivo de mantener una relativa simplicidad a la hora de controlar el vehículo y programar estos controles se usará el teclado como dispositivo de entrada. Por supuesto, se hace uso de un ordenador que ejecuta todo el software y gestiona los dispositivos.

3.2 Planificación

3.2.1 Tiempo para el desarrollo

No fue hasta abril que comencé con este proyecto, y por lo tanto el tiempo podría ser un problema ya que hasta que fuera verano, tendría que compaginarlo con otras cosas. Como tendría aproximadamente hasta finales de agosto para terminar el trabajo, eso me daba unos cinco meses en total. Sabiendo el tiempo que tendría disponible decidí reservar el último mes para la redacción de la memoria y el resto dedicarlo al desarrollo de la simulación.

3.2.2 Pruebas y consideración de alternativas

Aun habiendo seleccionado el entorno deseado para el desarrollo, existen herramientas como los "assets", es decir, contenido descargable creado por otros que podrían ser de utilidad. El primer paso debía ser investigar y probar varias opciones para ver cuál era más adecuada para los objetivos y la escala del proyecto. En esta primera etapa el objetivo era concretar de qué forma se iba a completar todo el desarrollo, poniendo el foco de atención en aquellos aspectos técnicos que pudieran suponer limitaciones a la hora de afrontar los objetivos planteados y valorando la facilidad y simplicidad de las herramientas seleccionadas. Una vez tomada la decisión, podemos comenzar con el desarrollo, veámoslo.

Capítulo 4. Desarrollo y resultados del trabajo

4.1 Primeros pasos

En un inicio el objetivo es averiguar de qué forma sería más viable afrontar el desarrollo de un sistema de controles que a partir de una serie de *inputs* haga mover un objeto de forma que se pueda personalizar y perfeccionar este movimiento hasta conseguir el resultado deseado

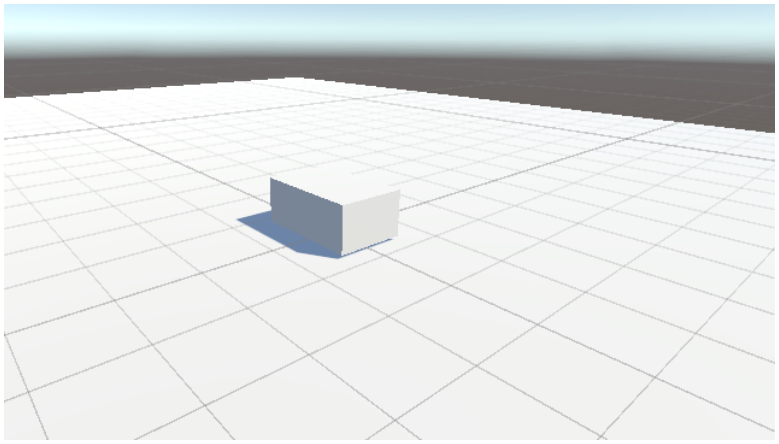

Figura 25 Prototipo de un cubo sobre un plano

Comienzo por crear un objeto con forma de ortoedro que simulará ser el coche y un plano para que el coche se pueda apoyar en algún lugar. En ambos objetos añado un componente "*collider*" el cual posibilita que estos objetos interaccionen colisionando entre sí, y en el caso del coche, también un componente "*rigidbody*" que será el encargado de representar este objeto en el espacio dotándolo de propiedades físicas como la masa y un componente de "*script*", como se observa en la figura 25.

Añado un componente de script para conseguir que el vehículo se mueva. De esta forma se ejecutará un pequeño código que aparece en la figura 26 el cual primero accede al componente "*rigidbody*" encargado de las físicas del coche y modifica como este se comporta mediante distintos métodos. Por ejemplo, el método "*MovePosition*" sirve para desplazar un objeto y al realizar muchos desplazamientos rápidos y pequeños se genera un movimiento constante. El método "*MoveRotation*" permite rotar el objeto de un modo similar. Estos métodos son capaces de mover el vehículo, pero por sí mismos no constituyen un control para que el vehículo se controlado por el jugador.

También hace falta que el script lea que teclas están siendo presionadas y ejecute los métodos previamente mencionados para generar el movimiento correspondiente. Esta es la función que tienen los métodos "*Input.GetAxis*()". Tras obtener el valor que representa una dirección en un eje, se emplea este valor para determinar la dirección en la que se ejecutarán los métodos de movimiento.

```
0 references
void Start()
{
    rb = GetComponent<Rigidbody>(); // Obtener el componente Rigidbody del objeto móvil
}

0 references
void FixedUpdate()
{
    float moveForward = Input.GetAxis("Vertical"); // Obtener la entrada vertical del teclado o controlador para mover el coche hacia delante o hacia atrás
    float turn = Input.GetAxis("Horizontal"); // Obtener la entrada horizontal del teclado o controlador para girar el coche

    Vector3 movement = transform.forward * moveForward * speed; // Crear un vector para el movimiento del coche en la dirección hacia adelante o hacia atrás
    Quaternion rotation = Quaternion.Euler(0.0f, turn * turnSpeed * Time.deltaTime, 0.0f); // Crear un quaternion para la rotación del coche

    rb.MovePosition(transform.position + movement * Time.deltaTime); // Mover el coche en la dirección del vector de movimiento
    rb.MoveRotation(rb.rotation * rotation); // Rotar el coche utilizando el quaternion de rotación
}
```

Figura 26 Código del control del primer prototipo

Para poder visualizar la escena hace falta un objeto “cámara” la cual tiene como parámetros una posición en la que se encuentra y una orientación a la que se dirige. Como para controlar el coche se necesita poder ver su posición en todo momento añado otro componente de script a la cámara. El script accederá a la posición en la que la cámara se encuentra, pero también la posición en la que se encuentra el coche y luego mueve la cámara con el método “Lerp” para que este movimiento sea suave y con el método “LookAt” para que apunte al coche.

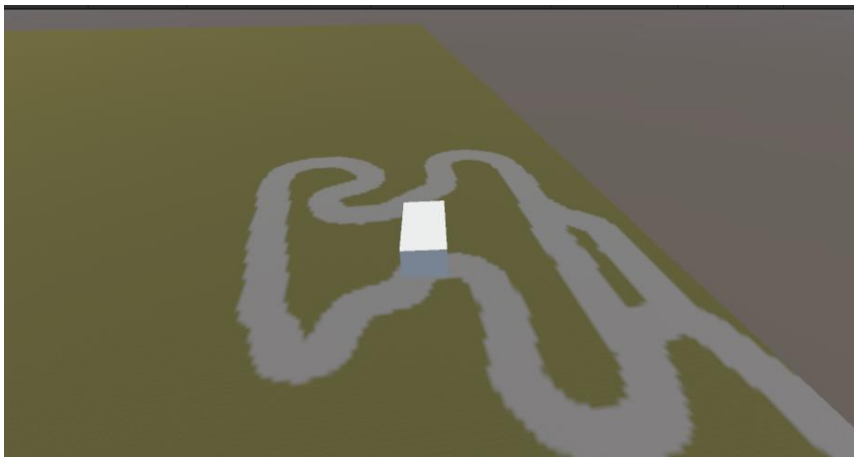

Figura 27 captura del primer prototipo

Aunque el cubo se podía mover, hacia adelante y atrás, y rotaba para girar, a veces volaba, ignorando la gravedad como se muestra en la figura 27. Además, también ignoraba el material del que el suelo había sido creado, cuando el resultado esperado era que, si la adherencia de la superficie es baja, el vehículo conserve mejor la velocidad. Había creado un objeto que se mueve con las teclas, pero el movimiento que describía no era similar al movimiento de un coche, ya que ni si quiera simulaba tener ruedas.

Llegados a este punto era momento de pararse, reflexionar y extraer conclusiones sobre el primer prototipo. Por un lado, la idea de usar un objeto simple como un cubo para desarrollar el prototipo del sistema de movimiento tenía como ventajas un bajo nivel de complejidad ya que se emplean únicamente los elementos estrictamente necesarios, y por otro lado una alta personalización ya que es el código que controla el objeto el cual tiene todo el protagonismo y determina el resultado.

Sin embargo, el lado negativo de este planteamiento es que al comenzar de cero es muy complicado idear un código que dé el resultado que se desea. Enfrentarse a un papel en blanco sobre el cual se quiere plasmar una idea sin saber cómo es de alguna forma como reinventar la rueda.

Tras entender esto decidí buscar otras opciones. El objetivo era conocer de qué maneras otras personas se habían enfrentado ya al objetivo de desarrollar el control de un vehículo en Unity.

Navegando por internet me topé con un *asset* llamado “Vehicle Physics Pro” que como su nombre indica, prometía ser una herramienta que facilitara la creación de vehículos con dinámicas realistas y personalizables como el que aparece en la figura 28. Con esta herramienta era posible conducir uno de los vehículos que trae de ejemplo de forma muy sencilla. También incluye una ciudad por donde se puede conducir el vehículo. [42]

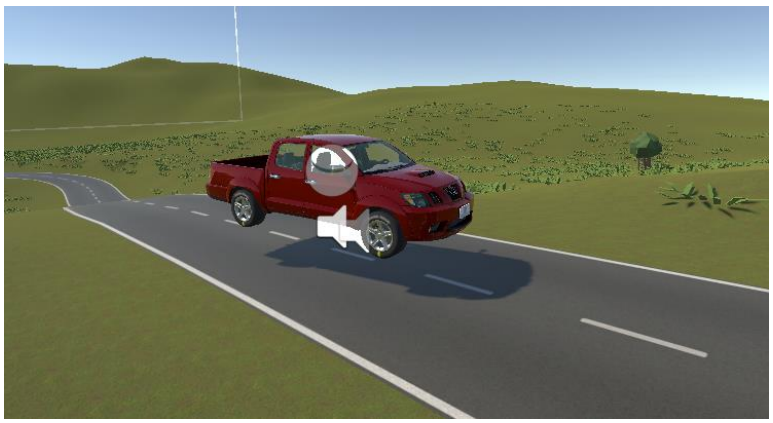

Figura 28 Vehículo prefabricado de "Vehicle Physics Pro" en Unity

Aunque inicialmente era prometedora la idea de emplear este asset para crear el movimiento del vehículo, pronto me encontraría con un problema. Intentando personalizar el movimiento del vehículo, no todas las opciones están disponibles, sino que algunas configuraciones se reservan para la versión de pago del asset, por lo que, si se quiere diseñar el movimiento de un coche teniendo un alto grado de libertad en el diseño, la versión gratuita supondría una importante limitación técnica.

La versión gratuita cuenta con la posibilidad de editar todos los parámetros de cada componente de los vehículos que permite crear, pero diseñar el movimiento del coche para que se comporte de una manera determinada es complicado ya que este asset cuenta con una gran profundidad en las mecánicas que conforman el movimiento del vehículo. Esta complejidad implica que es necesario estudiar y entender bien el funcionamiento del asset para poder usarlo correctamente.

La versión de pago, por otro lado, facilita el uso del *asset* y simplifica mucho las opciones de personalización de los parámetros del movimiento del vehículo, facilitando así la creación de un juego de conducción arcade, por ejemplo. Además, me topé con el problema de que este vehículo tampoco respondía ante los cambios en la superficie. Me enfrentaba en este momento a tener que comprender de forma profunda y completa el funcionamiento de un asset que cuanto más lo usaba, más complejo parecía ser.

Fue en este punto en que planteo el primero de los objetivos específicos; “conseguir una solución simple y generalista”. El asset Vehicle Physics Pro había terminado por convertirse en un problema en lugar de la solución.

Figura 29 Miniatura del vídeo [43]

Investigando todavía sobre más alternativas para realizar el proyecto di finalmente con el vídeo [43] cuya miniatura corresponde a la figura 29 y que inspiraría la lógica fundamental que se encuentra en la estructura del código que haría que el coche realmente se comporte como un coche. Me resultó atractivo el planteamiento que se muestra en el vídeo ya que, al igual que en el primer prototipo, también se partía de un cubo al cual se le aplica un script para crear el movimiento del vehículo. No obstante, el vídeo plantea una solución más lógica y completa que no consiste simplemente en añadir una velocidad a un objeto, sino que permitiría un grado de detalle y realismo mucho mayor.

Una diferencia fundamental entre el funcionamiento del primer prototipo y el que plantea el vídeo es que en el prototipo cuando se pulsa la tecla correspondiente se programa un desplazamiento constante mientras que en el vídeo se programan fuerzas mediante el método "AddForceAtPosition". Esta diferencia es el motivo por el que en el prototipo no afecta el material del suelo a la velocidad del desplazamiento mientras que con este planteamiento se pueden añadir fuerzas sobre el componente "rigidbody" que simulen un movimiento realista.

En un coche real las ruedas están en contacto con el suelo y soportan el peso del vehículo. El peso del chasis se transmite a las ruedas a través de la suspensión, y luego las ruedas lo transmiten al suelo por lo que podríamos decir que las ruedas actúan como intermediario entre estos transmitiendo fuerzas. Para comprender el planteamiento que se seguirá hay que entender el movimiento del coche como una consecuencia de las fuerzas que las ruedas transmiten al chasis por lo que, si se conocen cuáles son estas fuerzas se puede definir el movimiento a partir de ellas.

En otras palabras, se tratará de simplificar el movimiento del coche para que pueda ser definido como un resultado de las fuerzas que van de las ruedas al chasis. El objetivo, por tanto, será calcular estas fuerzas.

A diferencia que con el asset anterior, no hará falta crear un coche con diferentes componentes conectados, sino que todo esto se simplifica y el coche será un solo objeto con forma de ortoedro el cual tiene cuatro puntos en la ubicación de las ruedas los cuales serán los puntos donde se originarán las fuerzas que comunican al vehículo con el resto del mundo. A base de crear fuerzas en estos puntos de forma independiente, como si alguien los agarrase y estirase hacia una dirección es posible crear un control para el vehículo de forma mucho más simple que con el asset anterior pero mucho más completa y lógica que trasladando el objeto completo como en el primer intento. Para ilustrar mejor el proceso emplearé imágenes extraídas del vídeo ya que son muy ilustrativas.

4.2 El vehículo

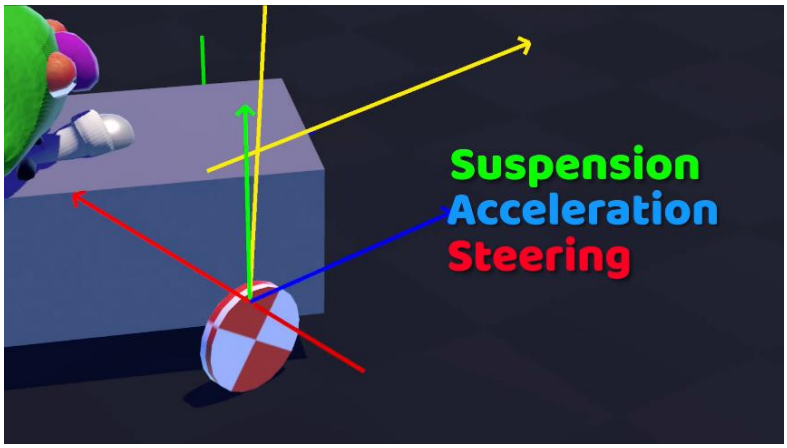

Figura 30 Ilustración de las fuerzas de las ruedas descompuestas [43]

A continuación, veremos en profundidad el funcionamiento del vehículo. La idea fundamental es que las fuerzas que se tendrán que generar en la ubicación de las ruedas pueden ser descompuestas para cada uno de los tres ejes o componentes de forma que al sumarlos simule las fuerzas que una rueda generaría sobre el chasis del coche como se aprecia en la figura 30. El eje vertical comprende las fuerzas relativas a la suspensión, el eje que va hacia adelante y atrás será el que comprenda las fuerzas relacionadas con la aceleración y el freno, y el eje que va del lado izquierdo al lado derecho comprende las fuerzas relacionadas con el agarre de las ruedas al suelo y posibilita que el coche gire.

Aplicar este planteamiento nos permite también acercarnos al objetivo específico número uno; conseguir un control simple y generalista, ya que al programar nosotros las fuerzas de cada eje nos permite hacer los cambios que sien necesarios más adelante si quisiéramos que este mismo código sirva para el movimiento de otro vehículo completamente distinto. Además, nos acercamos también al segundo objetivo, anteponiendo la jugabilidad al realismo, ya que la motivación detrás de programarlo de esta manera es poder manipular el movimiento del coche para que se comporte como queramos, y después alterarlo para que resulte verosímil, en vez de hacer el planteamiento opuesto, programando los procesos físicos a de forma estricta con el detalle suficiente para que el resultado total sea realista en su conjunto.

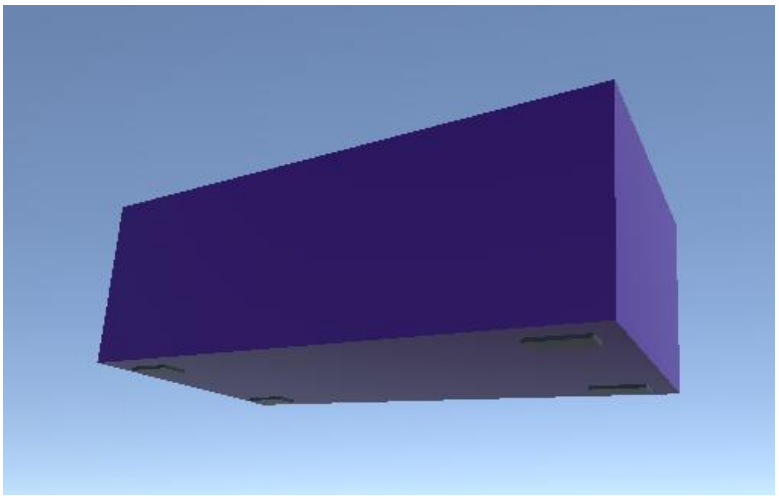

Figura 31 Prototipo de vehículo con la ubicación de las ruedas

Para empezar, hace falta que tengamos un objeto “coche” que contará con cuatro objetos hijos “ruedas” dispuestas como aparece en la figura 31. Que los objetos “rueda” sean hijos del objeto “coche” implica que cuando el coche se mueva las ruedas se moverán con este.

El objeto coche cuenta con un componente de script que aparece en la figura 32 el cual repite cada instante de tiempo los cálculos de las fuerzas para cada una de las ruedas mediante un bucle “*foreach*”. Primero emplea el método “*Raycast*” que permite saber la información y posición del objeto situado justo debajo de la rueda se determina si la rueda está en contacto o no con el suelo, y en caso afirmativo, se ejecutan una serie de funciones que serán las encargadas de generar las fuerzas que conforman el movimiento.

```
0 references
void FixedUpdate()
{
    float carMass = carRB.mass;

    foreach (Transform tireTransform in tireTransforms) //Para cada Transform de cada rueda
    {
        RaycastHit hit;
        if (Physics.Raycast(tireTransform.position, -tireTransform.up, out hit, groundCollisionDist) && (tireTransform.name == "FL
        {

            surfaceGrip = hit.collider.material.dynamicFriction;
            ProcessInput();
            CalculateSuspension(tireTransform);
            CalculateSteering(tireTransform);
            steeringVelGripFactor = CalculateGrip(tireTransform, carMass);
            carSpeed = CalculateAcceleration(tireTransform, carMass, steeringVelGripFactor);
        }
    }
}
```

Figura 32 fragmento del script de control del vehículo

Una vez se tiene el objeto que será el vehículo y puede detectar la superficie por la que se desplaza, se puede comenzar con la programación de las fuerzas, empezando por el eje más fundamental, el eje vertical.

4.2.1 Eje vertical

Las fuerzas que influyen en el eje vertical son, por un lado, la gravedad que sufre el cuerpo del coche que lo empuja hacia abajo y la fuerza de la suspensión que es la que tendrá que programarse. La fuerza de la gravedad la aplica Unity a los objetos con un cuerpo físico con masa así que no nos tendremos que preocupar por eso. En la figura 33 podemos ver una representación de la dirección de las fuerzas que corresponden a la suspensión.

Figura 33 Ilustración de la suspensión [43]

Comenzamos por entender cómo queremos que el coche responda y se comporte. Queremos que el cuerpo del coche no toque el suelo, se mantenga un poco por encima de este y de forma estable,

sin balanceos. Ahora veamos qué es lo que hace la suspensión de un vehículo en la vida real y definirlo con una fórmula para después poder programarlo. Primero podemos simplificar la fuerza de la suspensión como la suma de dos fuerzas, la de un muelle y la del amortiguador. Por un lado, el muelle se encargará de que el vehículo se mantenga a una altura determinada ya que ejercerá una fuerza para levantarlo, pero es la amortiguación la que logra que el vehículo sea estable, veámoslo con más detalle.

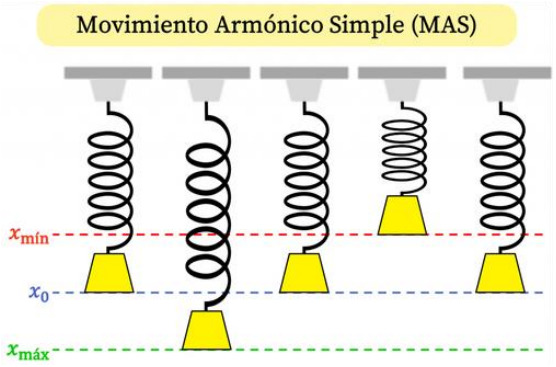

Figura 34 Ilustración del movimiento del muelle [44]

Un muelle tiene una longitud de reposo x_0 y una constante de elasticidad *k*. Cuando la longitud del muelle *x* es distinta de la longitud de reposo x_0 el muelle ejerce una fuerza *F* descrita por la fórmula **(1)** en dirección a la posición en reposo como ocurre en la figura 34. Con estos dos parámetros podemos programar una fuerza vertical que logre que el vehículo se mantenga a una altura determinada, pero observamos que no es estable.

$$\boldsymbol{F}_{muelle} = k * (x - x_0) \qquad \textbf{(1)}$$

Una vez se perturba el estado de reposo del muelle, este vuelve a su posición inicial, pero según se deduce de la formula (1) a medida se acerca a esta posición va acumulando energía cinética *v* y en lugar de quedarse en esa posición, sigue avanzando y esta energía cinética se transforma en energía potencial entrando así en un bucle que describe un movimiento armónico simple donde la energía total es constante ya que no se le opone ninguna resistencia. Aplicado a nuestro vehículo, esto causaría que se mueva hacia arriba y abajo descontroladamente y no se detuviese.

$$F_{amortiguación} = -\boldsymbol{v} * \boldsymbol{\lambda} \qquad \textbf{(2)}$$

Para contrarrestar este efecto se usa la amortiguación. Un amortiguador es un recipiente que contiene líquidos o gases que cuando se comprime o extiende tienen que pasar por unas válvulas ejerciendo una resistencia proporcional a la velocidad del muelle *v* y de sentido contrario a ésta como se determina por la fórmula (2) donde λ es una constante. La fuerza resultante de sumar la suspensión con la amortiguación viene definida por la fórmula (3) y consigue que el movimiento tienda a estabilizarse como se muestra en la figura 35.

$$F_{vertical} = \boldsymbol{k} * (\boldsymbol{x} - \boldsymbol{x_0}) - \boldsymbol{v} * \boldsymbol{\lambda} \quad \textbf{(3)}$$

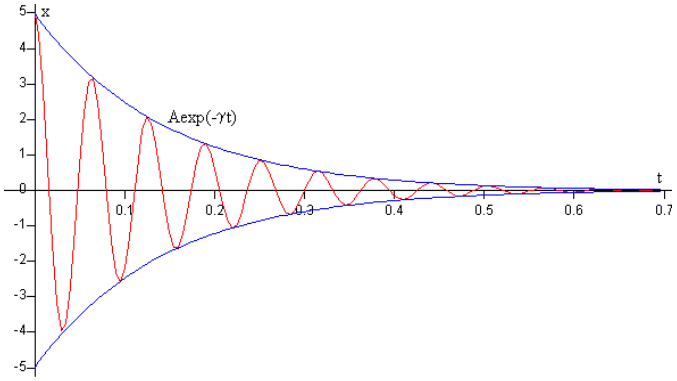

Figura 35 Trayectoria muelle con amortiguación [45]

En la figura 36 podemos observar una ilustración que muestra las direcciones de las fuerzas que participan de este movimiento. Como el "offset" es decir, la distancia hacia la posición de reposo es negativa esta distancia se multiplica por la constante que determina la rigidez del muelle y crea una gran fuerza en el mismo sentido negativo. Sin embargo, como la suspensión almacena una velocidad también en sentido negativo, el amortiguador ejerce una fuerza contraria a la velocidad, es decir una fuerza en sentido positivo.

Figura 36 Ilustración de una suspensión y fórmula de la fuerza de suspensión [43]

El código de la figura 37 muestra la programación de estas fuerzas. Algunas variables importantes son *"springDir"* que es un vector unitario que contiene la dirección de la suspensión de la rueda en sentido positivo y *"springForce"* que corresponde a la fórmula (3) planteada anteriormente.

La línea 88 del código corresponde con la función que genera las fuerzas, y tiene dos parámetros. El primer parámetro *(springDir * springForce)* contiene la dirección, el sentido y el módulo del vector de fuerza que se genera mientras que *"tireTransorm.position"* representa la ubicación donde esta fuerza se origina. Recordemos que cada función que genera fuerzas como esta se ejecuta cuatro veces por cada instante de tiempo, una vez por cada rueda.

```
1 reference
77  void CalculateSuspension(Transform tireTransform)
78  {
79      RaycastHit hit;
80      Physics.Raycast(tireTransform.position, -tireTransform.up, out hit, groundCollisionDist);
81      Vector3 springDir = tireTransform.up;
82      Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);
83
84      float springOffset = suspensionRestDist - hit.distance; //offset entre el reposo y la posicion de la rueda
85      float springVel = Vector3.Dot(springDir, tireWorldVel); //velocidad de la rueda en el eje de la suspensión
86      float springForce = (springOffset * springStrength) - (springVel * springDamper); //la fuerza que aplica la suspension
87
88      carRB.AddForceAtPosition(springDir * springForce, tireTransform.position);//AddForce para cada rueda
89  }
```

Figura 37 Función "*CalculateSuspension*()"

Hay parámetros que tienen valores constantes los cuales se definen al inicio del código como *"springStrength"* o *"springDamper"* que corresponden a las constantes del muelle y la amortiguación respectivamente. Los valores que toman se han elegido de forma arbitraria en base al resultado que se espera obtener de la función y ajustándolos "a ojo".

Ya hemos conseguido que el vehículo no choque con el suelo y se mantenga levitando y estable pero ahora lo que tenemos es una especie de aerodeslizador que si el suelo está inclinado simplemente se desliza hasta abajo. Hemos completado la programación de las fuerzas en el eje vertical, pero para conseguir el movimiento completo del vehículo necesitaremos programar también los otros dos ejes.

4.2.2 Eje lateral

Figura 38 Ilustración de la fuerza lateral [43]

En la figura 38 podemos ver una representación de la dirección de las fuerzas que corresponden al eje lateral.

Para entender las fuerzas que van a tener lugar en el eje lateral de nuestro vehículo hay que comenzar por definir cómo queremos que se comporte. Si el coche está situado en una pendiente hacia adelante o atrás las ruedas tendrían que girar y el coche desplazarse hacia abajo, pero si la inclinación de la superficie es hacia uno de los lados, este no se tendría que deslizar, sino que las ruedas deberían quedarse en su posición y agarrarse al suelo. Para que pueda ocurrir eso las ruedas tendrán que ejercer una fuerza al coche que contrarreste la velocidad lateral del coche.

Eso será posible si obtenemos para cada rueda cuál es la velocidad lateral que poseen y creamos una fuerza contraria suficiente para compensar esta velocidad. Si para cada instante repetimos ese proceso conseguiremos anular toda velocidad lateral, pero ¿Cómo calculamos la fuerza lateral necesaria que la rueda debe ejercer sobre el resto del vehículo?

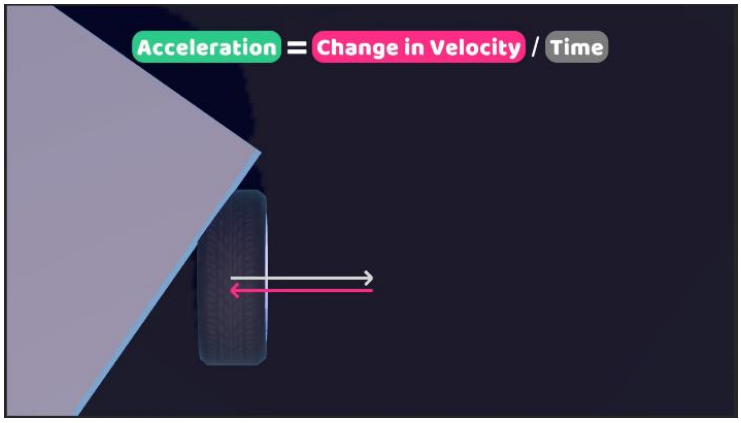

Figura 39 Ilustración de la velocidad lateral y la fuerza lateral contraria [43]

Primero necesitamos entender bien la física que hay detrás de este proceso, comenzando por la relación que tendrán en este caso la fuerza, aceleración, masa y velocidad. La rueda posee una velocidad y la variación de la velocidad es la aceleración. Queremos contrarrestar la velocidad que la rueda tiene para cada instancia de tiempo por lo que tendremos que añadirle una aceleración instantánea equivalente y opuesta como se dibuja en la figura 39. Por lo tanto, la fuerza lateral que programaremos sobre la masa del vehículo tendrá como objetivo provocar esta aceleración.

$$\boldsymbol{F = m * a} \qquad \textbf{(4)}$$

Recordemos que la fuerza es igual a la masa por aceleración (4). Por lo tanto, como la información que conocemos es la aceleración que queremos lograr, multiplicamos esa cantidad por la masa del vehículo y obtendremos la fuerza exacta que detendría el movimiento lateral del coche. A esto le añadimos el detalle de que la velocidad de la rueda la obtenemos en *m/s* pero estamos generando fuerzas instantáneas $(m/s)/dt$ donde *dt* es el tiempo que pasa entre cada vez que se ejecuta el código. Esto implica que la aceleración debe tener en cuenta el valor de estos periodos de tiempo como se muestra en la fórmula (5).

$$F_{lateral} = \frac{\boldsymbol{m_{coche}}}{\boldsymbol{4}} * \frac{\boldsymbol{-v_{lateral}}}{\boldsymbol{dt}} \qquad \textbf{(5)}$$

En cuanto a la masa, como las fuerzas se aplican por separado para cada rueda, la masa que usaremos será un cuarto de la masa del vehículo para que la fuerza acumulada de las cuatro ruedas sea la deseada.

```
1 reference
float CalculateGrip(Transform tireTransform, float carMass)
{
    Vector3 steeringDir = tireTransform.right;
    Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);

    float steeringVel = Vector3.Dot(steeringDir, tireWorldVel);
    float desiredVelChange = -steeringVel;
    float steeringAccel = desiredVelChange/ Time.fixedDeltaTime;

    carRB.AddForceAtPosition(steeringDir * steeringAccel * carMass /4, tireTransform.position);
```

Figura 40 Función "*CalculateGrip()*" versión 1

En la figura 40 observamos el código que genera las fuerzas descritas. Llamo "*seeringVel*" a la velocidad lateral de las ruedas y "*desiredVelChange*" a la velocidad que queremos generar en estas que es equivalente y opuesta. Como se muestra en la fórmula 5 la aceleración que provocará esta variación de velocidad es "*desiredVelChange / Time.fixedDeltaTime*". Al generar la fuerza

en la línea 118 se usa "*steeringDir*" para determinar la dirección del eje lateral. La figura 40 muestra un código que más adelante será modificado cuando se implemente un agarre variable, por lo que este no es su estado final.

Con esto ya hemos conseguido un vehículo que se mantiene a cierta altura del suelo y que, aunque se desliza hacia adelante y hacia atrás cuando el suelo esta inclinado, ya no se desliza hacia los lados.

4.2.3 Eje longitudinal

Figura 41 Ilustración del sentido de las fuerzas de aceleración y freno [43]

En la figura 41 podemos ver una representación de la dirección de las fuerzas que corresponden a la aceleración y el freno.

Ahora vamos a hacer que el coche pueda acelerar y frenar. Primero entendamos como queremos que acelere el vehículo. Cuando se presione el botón de acelerar se generará una fuerza en las ruedas traseras del vehículo hacia adelante que lo hará avanzar. Con una función que implemente ese funcionamiento es suficiente para que le vehículo se mueva, pero con el fin de añadirle un poco de realismo, esto no se ha programado así.

Un coche no consigue tener todo el rato la misma aceración, ni la velocidad puede llegar a infinito. Para eso emplearemos curvas de animación como la de la figura 42.

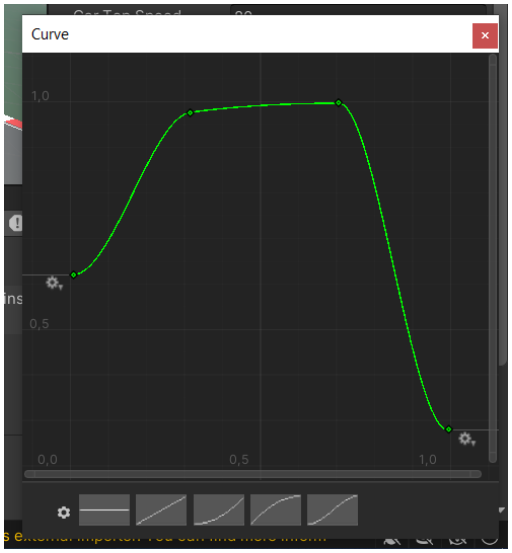

Figura 42 Curva de animación de la potencia entregada por el motor

Las curvas de animación son una herramienta que tiene Unity que nos permite evaluar un parámetro según una función que podemos dibujar de forma muy práctica desde el editor de Unity. Sirven para añadir dinamismo y evitar que los movimientos sean lineales, consiguiendo así personalizar más profundamente la experiencia final y añadir más realismo.

Sin entrar en conceptos técnicos de cómo funcionan los motores, generalmente la relación entre potencia entregada y las revoluciones describen una curva en la que, cuando el motor funciona a pocas revoluciones entrega menos potencia que cuando se encuentra en un rango de revoluciones óptimo. En el caso que nos ocupa, no se simulan revoluciones, ni tampoco hay varias marchas. Usamos la curva que se muestra para calcular cuanta fuerza debería generarse en el vehículo (eje vertical de la imagen) según lo cerca que estamos de la velocidad máxima del vehículo (eje horizontal).

Para comprenderlo de forma conceptual, entendamos la curva de animación como una función $f_{pot}(x)$ la cual evalúa lo cerca que se encuentra la velocidad *v* de la velocidad máxima que el vehículo puede alcanzar v_{max} y según lo cerca que esté, la aceleración variará entre 0 y F_{max}.(6)

$$F_{aceleración} = \boldsymbol{f}_{pot}\left(\frac{v}{v_{max}}\right) * \boldsymbol{F_{max}} \quad \textbf{(6)}$$

En cuanto a la frenada, basta con añadir a cada rueda una fuerza constante que se oponga a la velocidad que tiene en todos los sentidos para que el vehículo se pueda detener. El código se muestra en la figura 43.

```
1 reference
146 float CalculateAcceleration(Transform tireTransform, float carMass, float steeringVelGripFactor)
147 {
148     Vector3 carDir = tireTransform.forward;
149     Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);
150
151     float carSpeed = Vector3.Dot(carDir, tireWorldVel);
152     float normalizedSpeed = Mathf.Clamp01(Mathf.Abs(carSpeed)/carTopSpeed);
153     float availableTorque = powerCurve.Evaluate(normalizedSpeed) * accelInput;
154
155     if (forwardIsPressed && !backwardIsPressed && (tireTransform.name == "RLTire" || tireTransform.name == "RRTire"))
156     {
157         carRB.AddForceAtPosition(carDir * availableTorque, tireTransform.position);
158     }
159     else if (backwardIsPressed && !forwardIsPressed)
160     {
161         carRB.AddForceAtPosition(-tireWorldVel * breakForce, tireTransform.position);
162     }
```

Figura 43 Función "*CalculateAcceleration()*"

El parámetro "*availableTorque*" hace referencia a la $\boldsymbol{F_{aceleración}}$ definida anteriormente y "*brakeForce*" a la fuerza del freno, la cual es una constante definida al inicio del código. De la misma forma que en las funciones anteriores, en esta se crea una fuerza puntual en la posición de la rueda. En el caso de la aceleración, en la línea 157 se aplica la $\boldsymbol{F_{aceleración}}$ en el sentido del vehículo "*carDir*" y en el caso del freno se aplica "*brakeForce*" en la dirección opuesta a la velocidad de las ruedas individualmente. Este código también será modificado más adelante para implementar nuevas funcionalidades.

En cuanto a "*forwardIsPressed*" y "*backwardIsPressed*", estos son parámetros que hacen referencia a la activación de las teclas de control y forman parte de la entrada de datos o *input*.

4.2.4 Entrada

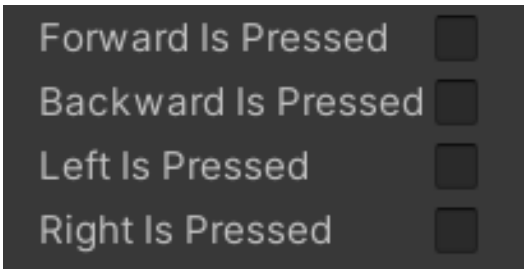

Figura 44 Campos de entrada del control

Como se ha mencionado anteriormente, uno de los objetivos de este proyecto es mantener cierta simplicidad, comenzando por mantener en cuatro la cantidad de botones que será necesario pulsar para manejar el vehículo. Para que el coche pueda ser controlado hará falta que el código incluya una función que para cada instante de tiempo compruebe que teclas están siendo presionadas y luego ejecute el resto de las funciones correspondientes. Usaremos las teclas "W" "A" "S" "D" y las flechas de dirección como controles para acelerar, frenar y girar a cada lado que se muestran en la figura 44 de forma que el usuario puede usar el que prefiera.

```
void ProcessInput()
{
    forwardIsPressed = Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow);
    backwardIsPressed = Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow);
    leftIsPressed = Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow);
    rightIsPressed = Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow);
}
```

Figura 45 Función "*ProcessInput()*"

El código para lograr que se procesen las teclas apretadas es simple. Este funciona mediante el método "*Input.GetKey*" como aparece en la figura 45 y comprueba iterativamente que teclas están siendo presionadas.

4.2.5 Giro

Para completar el control del vehículo necesitamos que este gire cuando se presionen las teclas de los lados. Como hemos programado las fuerzas para cada eje, este paso no será muy complejo. La fuerza lateral que evita que las ruedas se deslicen lateralmente será la que se encargará de que el vehículo pueda rotar. Bastará con programar que se añada un ángulo de rotación a la rueda cuando se pulsa un botón para girar, y la función anterior hará el resto ya que, al cambiar la orientación de la rueda, cambiará la dirección de las fuerzas generadas en esa rueda y la rueda al intentar no deslizar provocará una rotación en el coche igual que sucedería en la vida real.

Las opciones que el usuario tendrá para controlar la dirección son tres, girar a la izquierda, a la derecha o no girar, en lugar de tener un rango de movimiento como podría tener empleando dispositivos de control más complejos que nos alejarían de la simplicidad que buscamos con el primer objetivo específico. Con esto nos surge un problema, y es que en velocidades bajas necesitaremos que las ruedas tengan un ángulo de rotación mayor para girar en curvas cerradas y lentas mientras que en altas velocidades querremos lo contrario, ángulos de giro pequeños para poder corregir la dirección en una recta, por ejemplo.

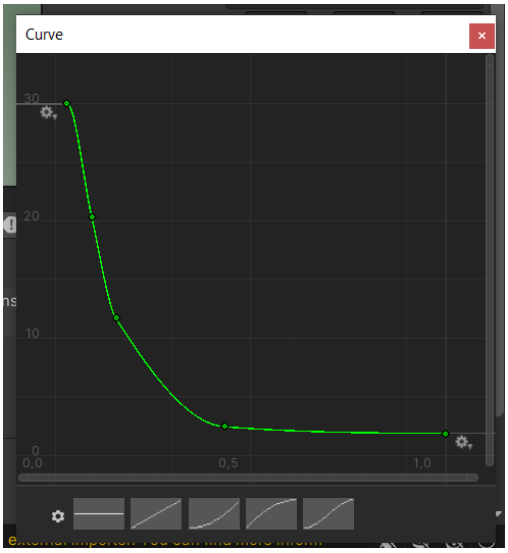

Figura 46 Curva de animación del Angulo de rotación de las ruedas al girar

Para compensar esta falta de libertad en el control y conseguir el vehículo pueda ser controlado en diferentes entornos, aplicaremos de nuevo una curva de animación. En la figura 46 lo que podemos observar es que cuando la velocidad es baja (parte izquierda) las ruedas rotarán treinta grados en la dirección deseada, y a medida la velocidad se va acercando a la velocidad máxima del vehículo, el ángulo de rotación se va reduciendo.

$$\propto_{rotacion\_ruedas} = f_{rota}\left(\frac{v}{v_{max}}\right) \qquad (7)$$

De la misma forma que se ha expresado la fuerza del motor al acelerar expresando la curva de animación del torque como una función, se puede expresar el valor del ángulo de rotación empleando la función $f_{rota}(x)$ donde x representa lo cerca que el vehículo se encuentra de su velocidad máxima, dando como resultado un valor entre 2º y 30º. (7) Estos valores los he elegido tras *testear* el control usando diferentes configuraciones hasta que la intensidad con la que el vehículo giraba era suficiente.

```
1 reference
void CalculateSteering(Transform tireTransform)
{
    currentRotationAngle = dynamicWheelRotation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed);

    if (leftIsPressed && !rightIsPressed && (tireTransform.name == "FLTire" || tireTransform.name == "FRTire"))
    {
        tireTransform.localRotation = Quaternion.Euler(0f, -currentRotationAngle, 0f);
    }
    else if (rightIsPressed && !leftIsPressed && (tireTransform.name == "FLTire" || tireTransform.name == "FRTire"))
    {
        tireTransform.localRotation = Quaternion.Euler(0f, currentRotationAngle, 0f);
    }
    else
    {
        tireTransform.localRotation = Quaternion.identity;
    }
}
```

Figura 47 Función "*CalculateSteering()*"

Este proceso lo lleva a cabo la función "*CalulateSteering*". Se evalúa la curva de animación para obtener el valor del ángulo de rotación de las ruedas que toma el nombre de "*currentRotationAngle*". Una vez obtenido el ángulo, se comprueba si se están presionando las teclas de giro, y en caso afirmativo, se aplica una rotación a las ruedas delanteras mediante el método "*localRotation*". Si no se están presionando las teclas de giro o se están presionando las

dos direcciones simultáneamente, la rotación de la rueda vuelve a su estado inicial como aparece en la figura 47.

4.2.6 Agarre

El tercer objetivo específico del proyecto consiste en añadirle al código la capacidad de simular algunos efectos físicos para completar el juego arcade con un poco más de realismo.

Los vehículos en el mundo real no tienen un agarre perfecto y absoluto, ni si quiera un Fórmula uno con gomas blandas y calientes, aunque pueda parecerlo. Un vehículo que tuviera un agarre absoluto, al tomar una curva a alta velocidad daría la sensación de que va sobre raíles de tren ya que su movimiento sería todo el rato hacia adelante, sin tener ningún tipo de deslizamiento lateral. Además, si la velocidad es suficiente, lo que conseguiremos es que el coche vuelque ya que por un lado las ruedas a las que se distribuye todo el peso durante el giro estarían tan pegadas al suelo que las otras se levantarían del suelo, dependiendo de la posición del centro de masas del vehículo.

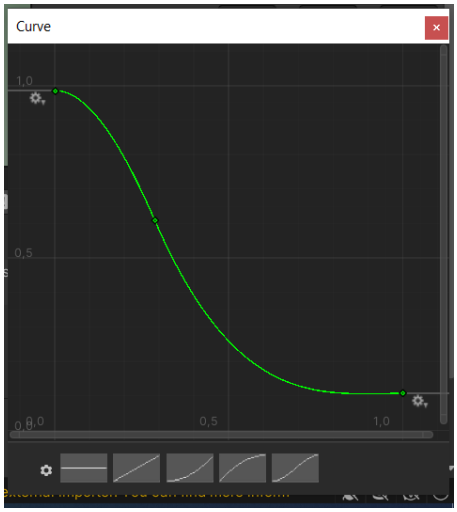

Figura 48 Curva de animación del agarre lateral en función de la velocidad lateral

Para evitar que eso suceda añadimos una curva que determinará un factor de agarre o "grip" que las ruedas tienen para cada instante en función de la velocidad lateral que experimentan, a mayor velocidad lateral, menor agarre. Un valor de agarre ideal sería igual a uno, mientras que si el valor fuera cero el agarre sería nulo. El valor del agarre que extraemos de aplicar la curva de animación de la figura 48 lo emplearemos en la función de agarre lateral explicada anteriormente para conseguir un movimiento más natural.

$$Factor\_agarre = f_{grip}\left(\frac{v_{lat}}{v_{max}}\right) \qquad \textbf{(8)}$$

El factor de agarre definido en la fórmula (8) se emplea en la función "*CalculateGrip*" de manera que si el valor de este factor de agarre es por ejemplo 0.8, la aceleración contraria a la velocidad lateral de la rueda que se genera para contrarrestarla anularía solo el 80% de esta velocidad en ese instante en lugar del 100% como definido anteriormente.

```
float CalculateGrip(Transform tireTransform, float carMass)
{
    Vector3 steeringDir = tireTransform.right;
    Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);

    float steeringVel = Vector3.Dot(steeringDir, tireWorldVel);
    float steeringVelGripFactor = lateralGripCurve.Evaluate(Mathf.Abs(steeringVel)/carTopSpeed);

    desiredVelChange = -steeringVel *  steeringVelGripFactor;

    float steeringAccel = desiredVelChange/ Time.fixedDeltaTime;

    carRB.AddForceAtPosition(steeringDir * steeringAccel * carMass /4, tireTransform.position);
    //Debug.DrawLine(tireTransform.position, tireTransform.position + steeringDir * steeringAccel
```

Figura 49 Función "*CalculateGrip()*" versión 2, con "*steeringVelGripFactor*"

En el código de la función "*CalculateGrip()*" que se muestra en la figura 49 se puede apreciar como la única diferencia entre este código y el mostrado anteriormente reside en la creación del parámetro "*steeringVelGripFactor*" el cual se calcula mediante la curva de animación. Luego al calcular "*desiredVelChange*" se tiene en cuenta el valor que tiene el factor de agarre, con lo que con eso se altera el comportamiento del vehículo añadiendo esta nueva funcionalidad.

4.2.7 Sobreviraje y subviraje

Ahora que tenemos un agarre variable se nos abre la posibilidad de añadir otro efecto más. En un vehículo de carreras real, cuando se toma una curva a una velocidad superior a la ideal pueden suceder dos cosas; que las ruedas delanteras pierdan agarre primero, o que lo hagan las traseras.

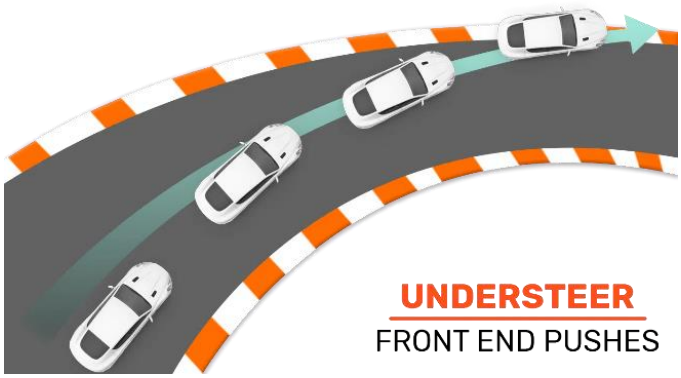

Figura 50 Ilustración de subviraje [46]

Se conoce como subviraje el fenómeno que se da cuando las ruedas delanteras pierden agarre antes. Lo que está sucediendo es que a pesar de que el conductor está girando el volante y las ruedas delanteras, estas terminan por deslizarse y no logran contrarrestar la energía cinética del vehículo ni consiguen hacer que este rote lo que lleva al coche inevitablemente a irse recto y salirse de la pista. Este fenómeno se puede dar por una variedad de motivos, como tener el centro de masa cerca de las ruedas delanteras, tener tracción delantera o por frenar demasiado poco o demasiado tarde. En la figura 50 se puede apreciar una ilustración con el movimiento descrito. [46]

Figura 51 Ilustración de sobreviraje [46]

Cuando por otro lado, son las ruedas traseras las que pierden agarre antes, lo que está sucediendo es que al girar el volante las ruedas delanteras sí siguen la trayectoria marcada por el piloto, pero las ruedas traseras se deslizan lateralmente y tratan de "adelantar" a las ruedas delanteras, ya sea por conservación de la energía cinética de la parte trasera del vehículo que quiere avanzar o porque se está dando gas mientras las ruedas no están adheridas al suelo. Este fenómeno se conoce como sobreviraje y se puede dar por una variedad de motivos, como tener el centro de masa cerca de las ruedas traseras, tener tracción trasera o por acelerar demasiado o demasiado pronto. En la figura 51 se puede apreciar una ilustración con el movimiento descrito.

En las competencias de conducción reales, los pilotos tienen preferencias personales a la hora de conducir un vehículo que sufre uno de los dos efectos como afirma Oscar Piastri, piloto de Fórmula 1 [47] en el vídeo que aparece en la figura 52. Estos pilotos se enfrentan entre ellos para ver quien es capaz de ser el más rápido posible mientras lidian con estos efectos, pilotando en muchas ocasiones al límite del sobreviraje o subviraje y son las limitaciones físicas del vehículo las que frenan a los pilotos.

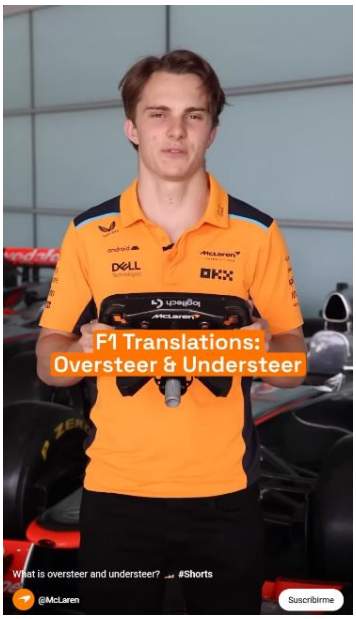

Figura 52 Youtube shorts de McLaren con Oscar Piastri [47]

En el proyecto que nos ocupa, como las ruedas pierden agarre con el aumento de la velocidad lateral, se logra replicar el efecto del subviraje ya que, si se pretende ir demasiado rápido por una

curva, las ruedas delanteras, que serán generalmente las que más velocidad lateral experimenten, debido a la curva de animación mostrada anteriormente perderán más agarre que las traseras. Para conseguir que el usuario pueda perder el control de las ruedas traseras al acelerar replicando así el sobreviraje vamos a alterar el código referente a la aceleración explicado anteriormente.

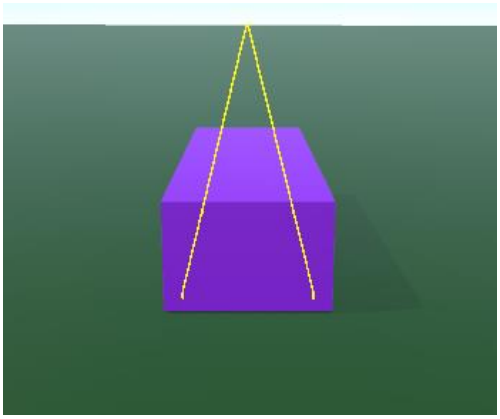

Figura 53 Dirección de las fuerzas de aceleración en línea recta

Se ha mencionado que al presionar la tecla de acelerar se generaría una fuerza hacia adelante. Eso da como resultado algo similar a lo que se observa en la figura 53 donde se muestra la dirección de esta fuerza. Esta fuerza es generada por la línea de código que se muestra en la figura 54 que se ha mencionado anteriormente.

```
if (forwardIsPressed && !backwardIsPressed && (tireTransform.name == "RLTire" || tireTransform.name == "RRTire"))
{
    carRB.AddForceAtPosition(carDir * availableTorque, tireTransform.position);
}
```

Figura 54 Línea de código de la función "*CalculateAcceleration()*"

Sin embargo, ahora vamos a introducir un cambio que añadirá un grado más de dificultad y realismo al proyecto. De la misma forma que en un Fórmula1 real, si se acelera al girar cuando las ruedas traseras no tienen agarre estas patinan y se desplazan hacia los lados resultando en un trompo, se va a programar que la dirección de la fuerza de aceleración que generan las ruedas traseras dependa también del nivel de agarre. Esto daría como resultado algo como lo que se observa en la figura 55.

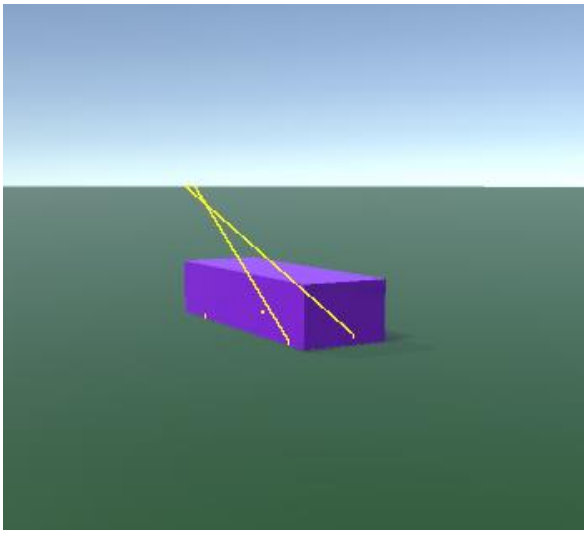

Figura 55 Dirección de las fuerzas de aceleración en curva

Para lograrlo, se programa que la fuerza de aceleración no vaya direccionada siempre hacia adelante, sino que esta dirección pueda variar; Cuando hay agarre completo, la fuerza de aceleración será hacia adelante, según la orientación del coche, pero, cuando no lo haya, la dirección de esta fuerza será la misma que la dirección de la velocidad de las ruedas.

Cuando en el código que se emplean direcciones, estas siempre son en forma de vectores unitarios tridimensionales. En este contexto, entendamos la dirección a la que "apunta" el coche como el vector unitario $\widehat{Car}$ y la dirección de la velocidad de las ruedas como el vector unitario $\widehat{Tirev}$. Para añadir este efecto se calcula la dirección en la que se aplica la aceleración $\hat{a}$ en base al *Factor_agarre* de la fórmula (8) de la siguiente forma. (9)

$$\hat{a} = \widehat{Car} * Factor\_agarre + \widehat{Tirev} * (1 - Factor\_agarre) \quad (9)$$

Esta fórmula solo tendrá relevancia cuando el jugador gire, ya que cuando no esté girando $\widehat{Car}$ y $\widehat{Tirev}$ coincidirán. Al implementar esta función en el código como se muestra en la figura 56 empleamos el vector "*accelerationDir*" y esta dirección es la que se usa como argumento para determinar la dirección de la fuerza generada por el método "*AddForceAtPosition*" en la línea 159.

```
146    float CalculateAcceleration(Transform tireTransform, float carMass, float steeringVelGripFactor)
147    {
148        Vector3 carDir = tireTransform.forward;
149        Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);
150
151        float carSpeed = Vector3.Dot(carDir, tireWorldVel);
152        float normalizedSpeed = Mathf.Clamp01(Mathf.Abs(carSpeed)/carTopSpeed);
153        float availableTorque = powerCurve.Evaluate(normalizedSpeed) * accelInput;
154
155        Vector3 accelerationDir = carDir*steeringVelGripFactor+tireWorldVel.normalized*(1-steeringVelGripFactor);
156
157        if (forwardIsPressed && !backwardIsPressed && (tireTransform.name == "RLTire" || tireTransform.name == "RRTire"))
158        {
159            carRB.AddForceAtPosition(accelerationDir * availableTorque, tireTransform.position);
160
161        }
162        else if (backwardIsPressed && !forwardIsPressed)
163        {
164            carRB.AddForceAtPosition(-tireWorldVel * breakForce, tireTransform.position);
```

Figura 56 Función "*CalculateAcceleration()*" con efecto sobreviraje

Es decir que, si se está girando, hay poco agarre y se acelera, las ruedas traseras generarán la fuerza hacia la misma dirección en la que ya se estaban moviendo y en lugar de ayudar a que el vehículo se controle fácilmente, se logrará añadir una limitación para el usuario, el cual tendrá que ir más despacio por el riesgo de perder el control de las ruedas traseras y realizar un trompo como casi le sucede a Max Verstappen en la figura 57.

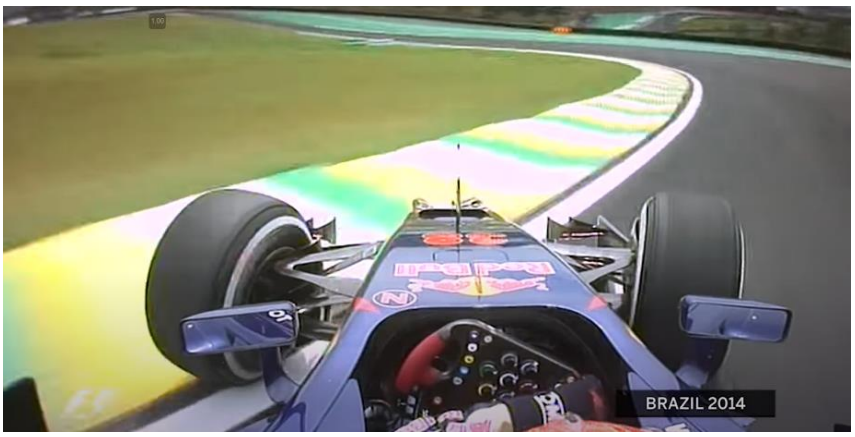

Figura 57 Momento de sobreviraje de Max Verstappen en el gran premio de Brasil de 2014 [48]

4.2.8 Degradación de neumáticos

Para finalizar el desarrollo del movimiento del vehículo añadimos otro efecto físico más propio de juegos de simulación a nuestro proyecto. La degradación de los neumáticos es un proceso que sucede siempre que se pilota un vehículo con ruedas de goma, pero tiene una especial influencia en los vehículos de carreras, ya que estos emplean gomas especiales diseñadas para conseguir el máximo agarre adaptándose a las características del vehículo y el circuito, sacrificando la durabilidad de las mismas.

Figura 58 Neumáticos con alto nivel de degradación [49]

En la fórmula uno podemos ver como a medida que los pilotos dan vueltas con el mismo neumático este va perdiendo la capacidad de transmitir la energía entre el asfalto y el vehículo, resultando en una reducción del agarre y por lo tanto en una menor aceleración y capacidad de realizar curvas rápidamente. En la figura 58 se muestra el aspecto que pueden tener unas ruedas que han sufrido degradación. Además, la degradación de los neumáticos por su uso no es lineal, sino que depende del circuito y como se conduzca. Si el circuito cuenta con muchas curvas de

alta velocidad a la derecha, por ejemplo, tenderán a desgastarse más las ruedas izquierdas, que son las que están soportando más rato la presión y el peso del coche. Si el circuito cuenta con frenadas intensas, tenderán a desgastarse antes las gomas delanteras por el mismo motivo.

Para simular este efecto en nuestro proyecto se ha añadido un nuevo parámetro. Este parámetro almacenará el estado de degradación de los neumáticos, almacenando un número que va desde uno hasta cero. Para que las cuatro ruedas no se desgasten a la vez, el código cuenta con un parámetro de degradación por cada rueda. La implementación de estos nuevos parámetros se realiza sobre la función "*CalculateGrip*". Estos harán que sea menor el agarre cuanto mayor es el desgaste de los neumáticos.

```
105     float CalculateGrip(Transform tireTransform, float carMass)
106     {
107         Vector3 steeringDir = tireTransform.right;
108         Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);
109 
110         float steeringVel = Vector3.Dot(steeringDir, tireWorldVel);
111         float steeringVelGripFactor = lateralGripCurve.Evaluate(Mathf.Abs(steeringVel)/carTopSpeed);
112 
113         if (tireTransform.name == "FRTire")
114         {
115             desiredVelChange = -steeringVel * FRTirewear * steeringVelGripFactor;
116             FRTirewear = FRTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
117         }
118         if (tireTransform.name == "FLTire")
119         {
120             desiredVelChange = -steeringVel * FLTirewear * steeringVelGripFactor;
121             FLTirewear = FLTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
122         }
123         if (tireTransform.name == "RLTire")
124         {
125             desiredVelChange = -steeringVel * RLTirewear * steeringVelGripFactor;
126             RLTirewear = RLTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
127         }
128         if (tireTransform.name == "RRTire")
129         {
130             desiredVelChange = -steeringVel * RRTirewear * steeringVelGripFactor;
131             RRTirewear = RRTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
132         }
133 
134         float steeringAccel = desiredVelChange/ Time.fixedDeltaTime;
135 
136         carRB.AddForceAtPosition(steeringDir * steeringAccel * carMass /4, tireTransform.position);
137         //Debug.DrawLine(tireTransform.position, tireTransform.position + steeringDir * steeringAccel * carMass /4, Color.yellow);
138 
```

Figura 59 Función *"CalculateGrip()"* versión 3, con desgaste de ruedas

Como se puede observar en la figura 59, el cambio que se realiza sobre la función "*CalculateGrip*" es la adición del parámetro de desgaste de rueda "*RRTirewear*" donde la "RR" hace referencia a la rueda que identifica, en ese caso "*rear right*" o trasera derecha. Este parámetro se emplea al calcular "*desiredVelChange*" en la línea 119, y como al igual que "*steeringVelGripFactor*" también tiene un valor entre 0 y 1, ambos contribuyen a reducir el agarre de las ruedas al girar. En el código se aprecian cuatro sentencias "*if*()" las cuales comprueban sobre qué neumático se está iterando en cada caso para garantizar que la función se aplica correctamente.

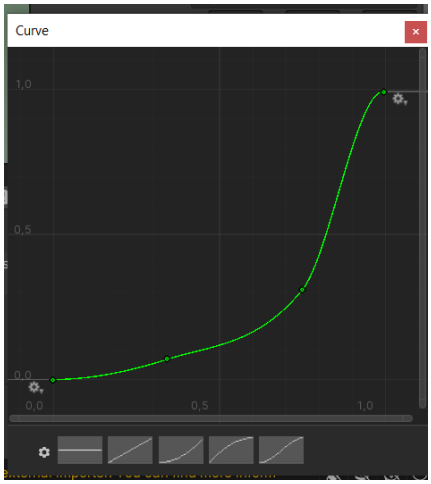

Figura 60 Curva de animación de degradación de neumáticos en función de la velocidad

Con ese cambio ya se altera el agarre de las ruedas en función del desgaste de las mismas, pero también hay que programar cuándo y cómo va a aumentar el desgaste de las ruedas. Como he dicho, se suelen desgastar al girar a gran velocidad y al frenar. Se ha empleado la curva de animación de la figura 42 para determinar cuánto debería desgastarse. Cuanta más velocidad registra el vehículo, más nos acercamos al límite de la derecha y el desgaste será mayor. A velocidades bajas, por mucho que gire o frene el desgaste será casi nulo ya que nos encontramos en el lado izquierdo de la gráfica. En el código de la figura 59 se recurre a esta curva en la línea 120 para calcular en qué medida se desgasta el neumático. Sigue la siguiente lógica.

$$\boldsymbol{Deg = f_{deg}\left(\frac{v}{v_{max}}\right) * \frac{|desired\_vel\_change|}{3000}} \qquad \textbf{(10)}$$

Mediante esa ecuación (10) se calcula la degradación de cada neumático de forma que es relativo a la velocidad y directamente proporcional al agarre lateral de este neumático. El número 3000 es una constante que se ha elegido tras probar varias opciones y seleccionar la que da un mejor funcionamiento final para que la degradación sea progresiva. La misma función de agarre que genera fuerzas laterales también implementa ahora la funcionalidad de reducir el estado de las ruedas de forma que el desgaste de cada rueda sea directamente proporcional a la fuerza lateral que genera. Con esto se logra que la rueda que más contribuye al agarre del vehículo porque es en la que se genera más fuerza lateral será también la que se desgastará a un mayor ritmo.

$$\boldsymbol{Deg = f_{deg}\left(\frac{v}{v_{max}}\right) * \frac{1}{5000}} \qquad \textbf{(11)}$$

Por otro lado, en la función "*CalculateAcceleration*", la cual se encarga de hacer que el vehículo acelere y frene, se ha añadido el código con la ecuación (11) cuando se está frenando. De esta forma se consigue un desgaste del neumático relativa a la velocidad que tiene el coche en ese instante. El número 5000, de nuevo, es una constante que se emplea para cambiar la magnitud de la degradación. Este código se puede observar en la figura 60.

```
else if (backwardIsPressed && !forwardIsPressed)
{
    carRB.AddForceAtPosition(-tireWorldVel * breakForce, tireTransform.position);
    FLTirewear = FLTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)/5000;
    FRTirewear = FRTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)/5000;
    RLTirewear = RLTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)/5000;
    RRTirewear = RRTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)/5000;

}
```

Figura 61 Función "*CalculateAcceleration()*" con degradación al frenar

4.2.9 Modelo 3D del vehículo

Figura 62 Modelo 3D de un Ferrari SF70H [50]

Finalmente será necesario que el objeto que estamos programando para que se comporte como un vehículo realmente se vea como uno. Para ello vamos a usar el modelo 3D de un formula 1 de Ferrari inspirado en el modelo sf70h que se puede ver en la figura 62. Otra opción podría haber sido crear manualmente el modelo 3D del vehículo que vamos a usar, pero se ha optado por descargar y emplear un modelo gratuito de internet ya que el elemento principal del desarrollo de este proyecto es el código referente al movimiento del vehículo y no el apartado artístico o estético, el cual hubiera requerido también de bastante tiempo dedicado.

4.3 La cámara

En cuanto al movimiento de la cámara, se ha optado por una solución tradicional en el género de juegos de conducción arcade. Se trata de la cámara de "persecución" que seguirá en todo momento al vehículo dando al usuario una perspectiva con la información de la ubicación del vehículo, así como del espacio que lo rodea.

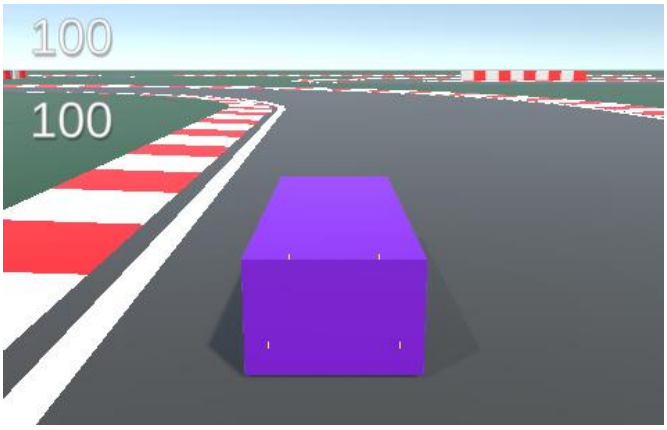

Figura 63 Captura del prototipo con cámara de persecución

Para lograr este efecto se ha empleado el objeto cámara de Unity al cual se le ha implementado un script. En este código se define el movimiento que la cámara realizará, el cual consiste en

situarse detrás y arriba del vehículo y mantenerse en la misma posición relativa del vehículo mientras que enfoca al mismo como se observa en la figura 63. El código empleado es muy similar al del primer prototipo. Este código es el que se muestra en la figura 64.

```
0 references
public class CameraFollow : MonoBehaviour
{
    2 references
    public Transform target;
    1 reference
    public Vector3 offset = new Vector3(0f, 5f, -10f);

    1 reference
    public float smooth = 8f;

    0 references
    private void LateUpdate()
    {
        Vector3 targetPosition = target.TransformPoint(offset);

        transform.position = Vector3.Lerp(transform.position, targetPosition, smooth * Time.deltaTime);
        transform.LookAt(target);
    }
}
```

Figura 64 Código de la cámara

4.4 El circuito

Por supuesto, para que un juego de coches pueda ser usado requiere de al menos un circuito. Para esto se ha optado por emplear un asset con el nombre de “Modular Lowpoly Track Roads FREE” [51] que incluye unos pocos segmentos de circuito a partir de los cuales se puede construir uno completo.

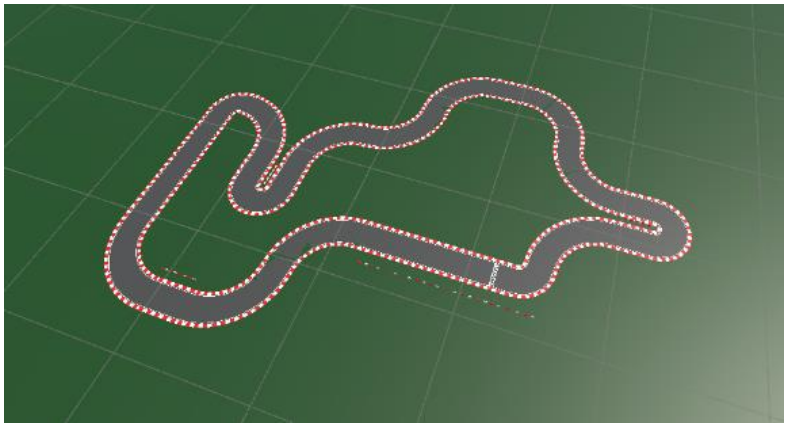

Figura 65 Prototipo del circuito creado

Para realizar esto se han colocado estos segmentos formando un circuito como el que se aprecia en la figura 65 y luego se les ha aplicado un reescalado ya que las dimensiones originales de estos objetos del asset son demasiado pequeños para el tamaño del vehículo. Además, se ha añadido sobre la línea de meta un objeto sin componente que determine su apariencia ya que será invisible y realizará la función de temporizador de vuelta.

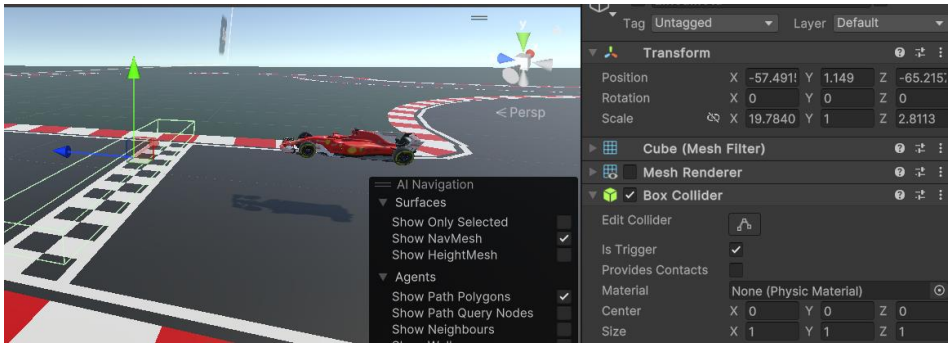

Figura 66 Objeto de línea de meta

Este objeto cuenta con un componente de script y un componente de colisión el cual tiene la opción "*Is Trigger*" activa como aparece en la figura 66. De esta forma en el código se puede emplear el método "*onTriggerEnter*" para que partes del código se ejecuten cuando otro objeto colisiona con este o lo atraviesa. El script es un contador de tiempo de vuelta que se activa cuando el usuario lo atraviesa con el vehículo y almacena el tiempo de la vuelta vigente, y al pasar de nuevo por la línea de meta almacena el tiempo en que se ha realizado la vuelta y comienza a contar de nuevo.

```
0 references
void Update(){
    if (hasStartedLap){

        elapsedTime = Time.time - startTime;
        //Debug.Log(elapsedTime);
    }
}
0 references
private void OnTriggerEnter(Collider other){
    if (hasStartedLap){
        lastLapTime = elapsedTime;
    }
    hasStartedLap = true;
    startTime = Time.time;
}
```

Figura 67 Script del contador de línea de meta

Para realizar esto se emplea un código muy simple el cual aparece en la figura 67. [52]

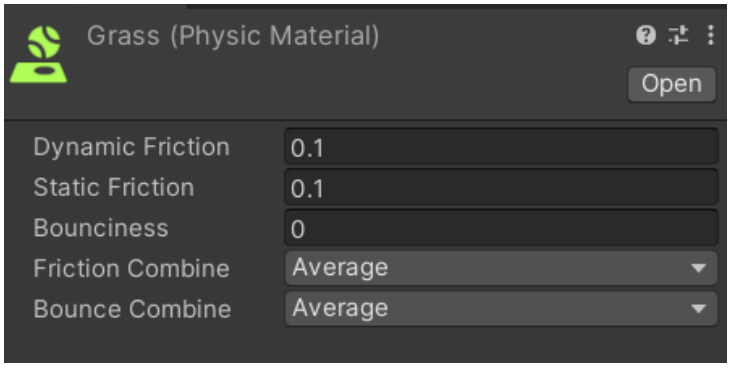

Figura 68 Parámetros de Unity del material de físicas del césped

Para añadir más complejidad al circuito, y seguir con la línea de desarrollo de alterar el agarre del vehículo, se ha añadido una variabilidad del material de físicas del suelo. El script del vehículo será capaz de leer el valor de la fricción dinámica que se observa en la figura 68 del objeto que

tiene debajo, y este valor se aplicará en la función del agarre de una forma similar a la implementación del desgaste de los neumáticos visto anteriormente. El objetivo es penalizar al jugador que en su intento por ser rápido excede los límites de la pista, y como consecuencia las ruedas pierden el agarre lo que a veces resultara en la pérdida total del vehículo y, casi siempre, en una pérdida de tiempo de la vuelta.

```
1 reference
float CalculateGrip(Transform tireTransform, float carMass)
{
    Vector3 steeringDir = tireTransform.right;
    Vector3 tireWorldVel = carRB.GetPointVelocity(tireTransform.position);

    float steeringVel = Vector3.Dot(steeringDir, tireWorldVel);
    float steeringVelGripFactor = lateralGripCurve.Evaluate(Mathf.Abs(steeringVel)/carTopSpeed);

    if (tireTransform.name == "FRTire")
    {
        desiredVelChange = -steeringVel * FRTirewear * steeringVelGripFactor * surfaceGrip;
        FRTirewear = FRTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
    }
    if (tireTransform.name == "FLTire")
    {
        desiredVelChange = -steeringVel * FLTirewear * steeringVelGripFactor * surfaceGrip;
        FLTirewear = FLTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
    }
    if (tireTransform.name == "RLTire")
    {
        desiredVelChange = -steeringVel * RLTirewear * steeringVelGripFactor * surfaceGrip;
        RLTirewear = RLTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
    }
    if (tireTransform.name == "RRTire")
    {
        desiredVelChange = -steeringVel * RRTirewear * steeringVelGripFactor * surfaceGrip;
        RRTirewear = RRTirewear-tireDegradation.Evaluate(Mathf.Abs(carSpeed)/carTopSpeed)*Mathf.Abs(desiredVelChange)/3000;
    }

    float steeringAccel = desiredVelChange/ Time.fixedDeltaTime;

    carRB.AddForceAtPosition(steeringDir * steeringAccel * carMass /4, tireTransform.position);
```

Figura 69 Código de la función "*CalculateGrip()*" versión 4, con "*surfaceGrip*"

Para programar esto se ha empleado el código de la figura 69, que vuelve a ser nuevamente la función "*CalculateGrip*" pero ahora al calcular el parámetro "*desiredVelChange*" como en la línea 119 también se tiene en cuenta el valor de "*surfaceGrip*".

```
    float carMass = carRB.mass;

    foreach (Transform tireTransform in tireTransforms) //Para cada Transform de cada rueda
    {
        RaycastHit hit;
        if (Physics.Raycast(tireTransform.position, -tireTransform.up, out hit, groundCollisionDist) && (tireTransform.name == "FLT
        {

            surfaceGrip = hit.collider.material.dynamicFriction;
            ProcessInput();
            CalculateSuspension(tireTransform);
            CalculateSteering(tireTransform);
            steeringVelGripFactor = CalculateGrip(tireTransform, carMass);
            carSpeed = CalculateAcceleration(tireTransform, carMass, steeringVelGripFactor);
        }
    }
```

Figura 70 script donde se accede a "*surfaceGrip*"

Para obtener el valor del "*surfaceGrip*" correspondiente a la superficie que se está recorriendo en cada momento se añade una línea en el primer bucle que hemos visto, el de la figura 70 y la línea 59 lo que hace es almacenar el valor de la fricción del material que se encuentra directamente bajo la rueda mediante el "*Raycast*" mencionado

4.5 La interfaz de usuario

Figura 71 Apariencia de la interfaz gráfica de usuario

Por mucho que se añadan mecánicas al juego, el usuario necesita ser capaz de entender qué está sucediendo y una buena manera de hacer llegar información al usuario es a través de una interfaz gráfica como la de la figura 71 que aporte la información justa y necesaria que el usuario necesita para poder disfrutar de una experiencia completa.

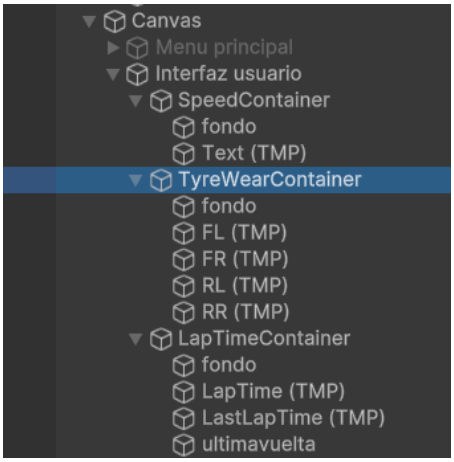

Figura 72 Objeto "canvas" con la interfaz gráfica

Mediante un objeto de “Canvas” como el de la figura 72 con el que Unity cuenta, se pueden construir fácilmente interfaces gráficas con objetos de texto llamados “TextMeshPro”. además, a estos objetos de texto se les puede añadir también un script que recoja el valor de una variable de otro script y luego cambiar el texto que se muestra en pantalla con ese valor.

```
0 references
public class speedometer : MonoBehaviour
{
    2 references
    private TMPro.TextMeshProUGUI speedText;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        speedText = GetComponent<TMPro.TextMeshProUGUI>();
    }


    0 references
    void Update()
    {
        CarMovement_F1 carMovement = FindAnyObjectByType<CarMovement_F1>();
        if (carMovement != null)
        {
            speedText.text = carMovement.carSpeed.ToString("F0");
        }
    }
}
```

Figura 73 Código del velocímetro de la interfaz

Es de esta forma que se ha añadido a la interfaz el estado de la degradación de las cuatro ruedas en la parte superior izquierda, así como un velocímetro en la zona inferior derecha y el temporizador de la vuelta actual arriba a la izquierda como aparece en la figura 73. Los valores de la degradación de las ruedas y de la velocidad se toman del script del movimiento del coche mientras que el del tiempo de vuelta actual y el ultimo se toman del script que está siendo ejecutado por el objeto en la línea de meta el cual hace la función de temporizador.

4.6 Menú principal

Figura 74 Menú principal

Con la finalidad de terminar de empaquetar bien este prototipo de juego se ha realizado un esquemático menú principal como el de la figura el cual cuenta solo con la opción de reproducir el juego o de cerrarlo por completo. A pesar de que la funcionalidad del menú principal es limitada por el momento, este abre las puertas a la opción de mejorarlo en el futuro.

```
public class menu_principal : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    public void PlayGame (){
        SceneManager.LoadScene(1);
    }
        0 references
        public void QuitGame (){
        Debug.Log("QUIT");
        Application.Quit();

    }
}
```

Figura 75 Script del menú principal

Para desarrollarlo se han seguido los pasos relatados en el vídeo [53]. Primero es necesario contar con las dos escenas entre las que se alternaran. Una escena que contiene el menú y otra escena que contiene el juego. En la escena del menú hay un objeto de "*canvas*" como el que se ha empleado para realizar la interfaz gráfica, pero en esta ocasión con elementos botones en lugar de texto. se ha añadido un script al objeto "*canvas*" como el que se muestra en la figura 75. En el script se definen dos funciones las cuales corresponden a cada botón del menú. Como se muestra en la figura 76, los botones pueden ejecutar funciones en el apartado "*OnlClick*()" desde el propio editor de Unity, y de esta manera se consigue añadirle funcionalidad a los botones del menú principal.

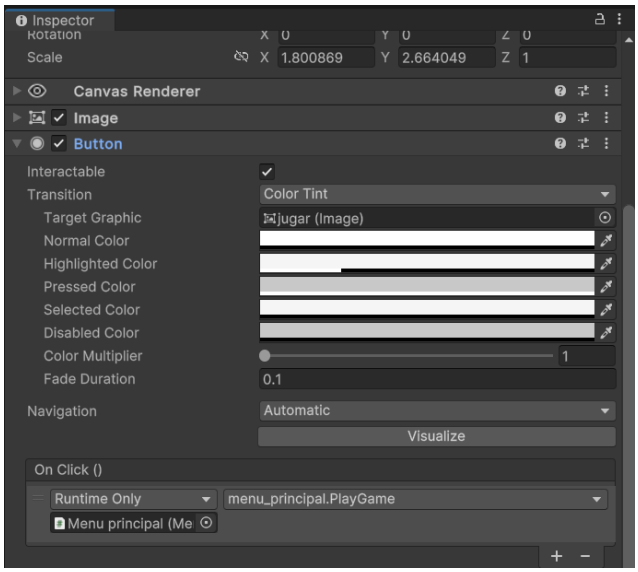

Figura 76 Función "*OnClick()*" del objeto *Button*

Capítulo 5. Conclusiones y propuestas de trabajo futuro

5.1 Conclusiones del proyecto

En este punto del desarrollo se puede concluir que es posible realizar un juego de coches con controles simples y estilo arcade que implemente elementos de la jugabilidad propias de títulos de simulación de conducción. Si bien es cierto que solo se ha abordado la superficie del profundo iceberg que son los simuladores de conducción, se ha conseguido satisfacer el objetivo principal que motiva el desarrollo del proyecto que consiste en crear un entorno de interacción simple y centrado en la jugabilidad, pero adoptando elementos físicos que aporten realismo, así como un circuito y una interfaz.

Valorando el trabajo realizado observo que ha sido precisamente por tratar de mantener las físicas del vehículo simples, descomponiendo las fuerzas que involucran a cada uno de los ejes para tratarlas por separado hemos podido desarrollarlas y mejorarlas implementando nuevas funcionalidades, cosa que si hubiéramos empleado asset de control de vehículos ya desarrollados hubiera resultado complicado. Además de habernos permitido añadir las funciones que se han comentado, esta lógica a la hora de desarrollar el control del vehículo abre las puertas a que en el futuro pueda ser mejorado implementando nuevas soluciones.

No obstante, resulta evidente con ver el resultado del proyecto que son muchos los campos en los que se podría seguir trabajando para mejorar el producto final. A lo largo del desarrollo se ha puesto énfasis en completar la funcionalidad del control del vehículo para asegurar que el prototipo cuenta con los requisitos mínimos para considerarlo funcional, sin embargo, tanto en la funcionalidad como en la experiencia del usuario son muchos los puntos que se podrían seguir desarrollando. A continuación, se exponen una serie de propuestas como puntos a mejorar.

5.2 Propuestas

5.2.1 Propuestas de jugabilidad

El principal punto por mejorar consiste en realizar *tests* suficientes sobre el control para poder cambiar tanto los parámetros que definen el control como el código y las curvas de animación con el objetivo de lograr que el vehículo se controle mejor. En un juego de carreras, el movimiento del vehículo es el lenguaje a través del cual el usuario se relaciona con el entorno del juego, y por lo tanto, es primordial que el control sea cómodo, divertido y predecible. Además, del control del vehículo dependerá que el juego resultante sea divertido o no por lo que es un elemento primordial tanto en términos de usabilidad como de la experiencia del usuario. A través de toda experiencia interactiva se busca transmitir algo en el usuario, y son las físicas las que tienen que estar ajustadas de una manera correcta para transmitir emociones como la emoción de ir rápido.

La experiencia de juego también podría ser más completa e inmersiva si tomar cada curva lo más rápido posible no es el único objetivo. En algunos simuladores de conducción, la forma que uno tiene de conducir repercute no solo en el tiempo de vuelta, sino también en el desgaste de las piezas. Es posible mejorar el sistema de movimiento actual para añadir la posibilidad de que la forma más rápida de tomar alguna curva no siempre sea la mejor porque de esa forma degradas más los neumáticos o gastas más combustible, mientras que la forma óptima de gestionarlo sería tomar la curva más lentamente y compensar ese tiempo perdido en las siguientes vueltas ya que se cuenta con unos neumáticos en mejor estado. Esta gestión de neumáticos y combustible junto con la adición de paradas en boxes para cambiar neumáticos podría añadir nuevas capas de profundidad a la experiencia, creando la posibilidad de diseñar y aplicar estrategias distintas.

Em la figura 48 se puede observar la ventana de selección de estrategia de “F1 22” en la cual el jugador tiene la posibilidad de decidir el tipo de neumático con el que quiere comenzar la carrera, el neumático que usara tras la parada en boxes y la cantidad de combustible que llevará.

Figura 77 Captura de la ventana de selección de estrategia de f1 22 [54]

5.2.2 Propuestas de controles

Uno de los objetivos de este proyecto consistía en mantener los controles lo más simples posibles con el objetivo de facilitar el desarrollo y reducir la complejidad de este. Sin embargo, eliminar esta restricción podría contribuir a ampliar mucho las posibilidades en cuanto a lo que es posible hacer con el vehículo.

La forma más sencilla y económicamente accesible para el usuario sería emplear controladores de Xbox como entrada. Aunque sí que sería más complejo que el sistema de controles empleado actualmente, usando un paquete de control de entrada de datos, y tras configurar las teclas que van a ser usadas, la mayor parte del cambio sería cambiar el código para que funcione como funcionaban las teclas del teclado. Una vez logrado eso, se podría comenzar a expandir el manejo del vehículo empleando las nuevas funcionalidades que el controlador de Xbox puede aportar. Por ejemplo, este mando cuenta con dos joysticks y dos gatillos que, en lugar de tener solo dos estados, uno para cuando es presionado y otro para cuando no lo es, tienen todo un rango de posibilidades entremedias. Lo recomendable sería enlazar el presionado de los gatillos a la aceleración y el freno y, uno de los ejes de un gatillo al giro. Al permitir al usuario controlar el vehículo con un mayor detalle, también se abre la posibilidad de aumentar la dificultad exigiendo que el control sea más preciso.

Por ejemplo, puede programarse que la forma más eficiente de frenar sea presionar fuerte el freno primero y luego ir soltándolo para que las ruedas no se bloqueen como hacen los pilotos de Fórmula 1 reales. Al acelerar se puede mejorar el sobreviraje implementado anteriormente con la posibilidad del usuario de dar gas poco a poco para contrarrestarlo. La combinación de estas técnicas en el circuito cuando se afronta una curva lenta que requiere frenar es la forma óptima en la vida real y se conoce como “trail braking”. [55] La figura 49 muestra la telemetría de un vehículo que frena intensamente en un inicio y mas suave a medida se adentra en la curva y del acelerador que abre el gas gradualmente.

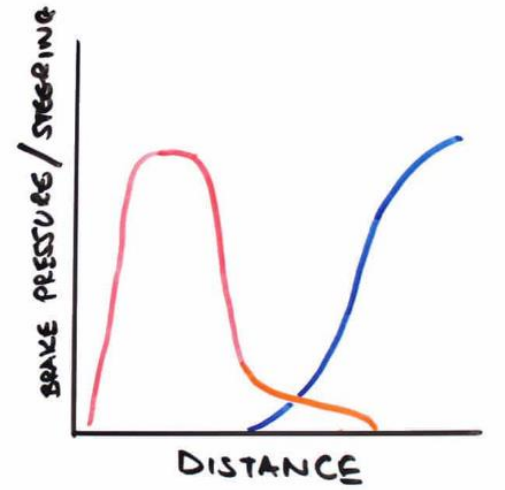

Figura 78 Boceto de la telemetría describiendo el "Trail braking" [55]

En cuanto a formas de controlar experiencias interactivas de conducción, es importante destacar las ventajas que podría aportar la posibilidad de controlar el juego con pedales y un volante con "force-feedback" o respuesta de fuerza como el que se muestra en la figura 50. Por un lado, permitiría al usuario aplicar una cantidad concreta de aceleración, freno o de giro en las ruedas, pero, además los volantes motorizados no son solo dispositivos de entrada de información sino también de salida. El jugador siente desde sus manos las fuerzas generadas por el volante, las cuales transmiten información como cuanta fuerza está siendo ejercida sobre las ruedas.

Figura 79 Logitech g29 [56]

A medida se le fuera añadiendo profundidad al control nos estaríamos alejando de lo que se entiende como juego de conducción arcade, pero la capacidad de controlarlo con volante y pedales sí que implica un importante acercamiento al género de simulación de conducción.

Por último, otra función de control a añadir podría ser la posibilidad de subir y bajar de marcha. Esto podría tener un efecto en la cantidad de potencia de la que el vehículo dispone según la marcha y las revoluciones, y además afectaría al consumo de combustible

5.2.3 Propuestas de contenido

No obstante, la propuesta más grande de mejora del proyecto tiene que ser la adición de nuevo contenido. Lo que se ha desarrollado en este proyecto se puede calificar de demo técnica, ya que el foco se ha puesto en el desarrollo de las físicas relacionadas con el control del vehículo. Para poder considerarlo un juego completo haría falta definir un reto al cual un jugador se enfrente a través de los controles diseñados.

La implementación de distintos modos de juego sería el primer paso a seguir a la hora de convertir este proyecto en un juego real. Tanto si se trata de construir un entorno multijugador en línea, como una experiencia para un solo jugador, es importante definir cuál es el objetivo para que el

jugador sepa como perseguirlo. Para un jugador es posible crear carreras contra otros coches controlados con inteligencia artificial y modo contrarreloj para mejorar sus mejores tiempos. Para jugar en línea es posible crear modo contrarreloj donde cuenten la duración de la partida para conseguir el mejor tiempo en una vuelta posible sin poder colisionar entre sí, un modo de carrera rápida donde los participantes simplemente disputan una carrera de corta duración y modo gran premio, el cual podrá contar con práctica, clasificación y una carrera de mayor duración donde la estrategia de paradas en boxes ganaría protagonismo. Un planteamiento más arcade sería añadir ayudas o bonificaciones para los jugadores que se queden rezagados ya que así se logra reducir la frustración que pueda llegar a causar, aunque se sacrifica el lado más competitivo que suele vincularse al deporte de motor. En la figura 51 podemos ver una captura de "Mario Kart 7" en un menu que cuenta con diversos modos de juego disponibles.

Figura 80 Captura de los modos de juego de "Mario Kart 7" [57]

Otra opción es añadir contenido en forma de nuevos vehículos. En cuanto a los vehículos que se pueden usar, una posibilidad es añadir aquellos vehículos que suelan ser usados en carreras reales como formula dos, formula e, GT o "hypercars" adaptando el código del control para que este se asemeje más al del tipo de vehículo. Esto se podría hacer añadiendo al código métodos para gestionar la distribución del peso en el vehículo, el efecto aerodinámico de la carrocería y el funcionamiento de los motores híbridos entre otras cosas. Otra posibilidad es permitir al usuario añadir modificaciones al vehículo de tal manera que, ante unas opciones disponibles, cada uno construya una versión del mismo vehículo que se adapte más a su estilo de conducción o que el usuario considere que es más rápido.

Otro aspecto en el que se podría ampliar el contenido es en relación con la cantidad y variedad de circuitos. Se pueden crear diversos circuitos para que los jugadores puedan competir entre si ya sea en carrera o en modo contrarreloj. Además, siguiendo la idea del juego "Trackmania: Nations Forever" también puede resultar interesante la creación de pistas como niveles, ordenados de menor a mayor dificultad donde el jugador se enfrente a un tiempo mínimo que ha de superar para pasar al siguiente nivel como los que se observan en la figura 52.

Figura 81 Captura de la selección de niveles de "Trackmania: Nations Forever" [58]

Para finalizar, otro aspecto de contenido que se podría desarrollar es una inteligencia artificial que sea capaz de controlar el vehículo. Diferentes títulos han desarrollado este concepto de forma distinta. Por un lado, tenemos juegos como “F1 22” donde para aumentar la dificultad de esta inteligencia artificial lo que hacen es reducir el peso de sus vehículos para que cuenten con ayudas extra y ser así más rápidas. Sin embargo, el modelo de IA más avanzada que se ha creado en un juego de conducción es el que tiene “iRacing” el cual podemos observar en la figura 53 donde a partir de la información que captura de carreras online con jugadores reales, genera modelos de distintos niveles de conducción que emulan muy fielmente el estilo de conducción de la gente, incluso copiando sus errores. Con esto sería posible hacer que el jugador se enfrente por horas contra la IA sin necesidad de conexión a internet.

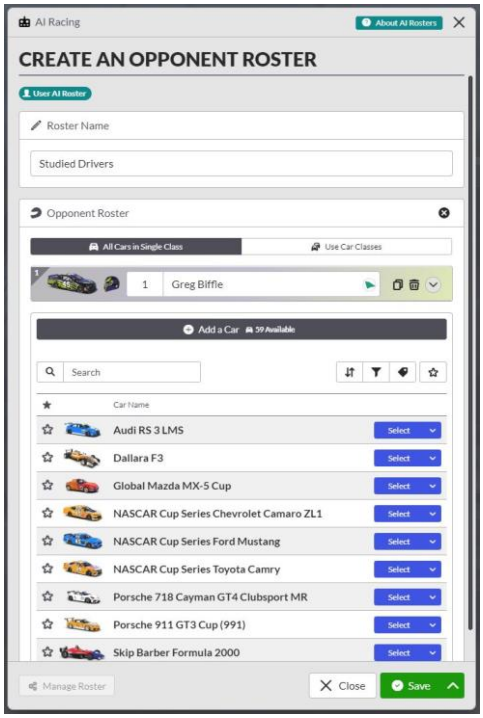

Figura 82 Ventana de personalización de los rivales AI en "iRacing" [59]

Capítulo 6. Bibliografía

[1] Colaboradores de Wikipedia, «Interacción persona-computadora», Wikipedia, la enciclopedia libre, may 2023, [En línea]. Disponible en: https://es.wikipedia.org/wiki/Interacci%C3%B3n_persona-computadora

[2] «Disciplinas involucradas en la interacción humano computadora». https://1library.co/article/disciplinas-involucradas-en-la-interacci%C3%B3n-humano-computadora.y8r7d95q

[3] K. A. Kendall y J. E. Kendall, «Systems analysis and design», en CRC Press eBooks, 2015, pp. p. doi: 10.1201/b18362-20. http://cotana.informatica.edu.bo/downloads/ld-Analisis%20y%20Diseno%20de%20Sistemas_Kendall-8va.pdf

[4] «¿Qué es la interacción humano-computadora (HCI)? - Definición de Techopedia - Desarrollo 2023», *Icy Science*, 2023. https://es.theastrologypage.com/human-computer-interaction

[5] «Los videojuegos están en auge y se espera que la industria siga creciendo.», *Foro Económico Mundial*, 9 de septiembre de 2022. https://es.weforum.org/agenda/2022/09/el-juego-esta-en-auge-y-se-espera-que-siga-creciendo-este-grafico-le-dice-todo-lo-que-necesita-saber/

[6] «Anuario de la industria del videojuego en España 2021 (AEVI)», *spainaudiovisualhub.mineco.gob.es*, 21 de abril de 2022. https://spainaudiovisualhub.mineco.gob.es/es/actualidad/anuario-de-la-industria-del-videojuego-en-espana-2021--aevi-

[7] Colaboradores de Wikipedia, «Motion Simulator», *Wikipedia*, may 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Motion_simulator

[8] Louise, «Driving Simulator Motion Systems 101 - Cruden», *Cruden*, 22 de junio de 2022. https://www.cruden.com/2020/04/22/driving-simulator-motion-systems-101/

[9] Admin, «¿Qué es el force feedback en un volante?», *Volantes.pro*, ene. 2021, [En línea]. Disponible en: https://www.volantes.pro/blog/force-feedback/

[10] J. Melús, «El enorme potencial de los simuladores de conducción», *ITCL*, ene. 2023, [En línea]. Disponible en: https://itcl.es/blog/el-enorme-potencial-de-los-simuladores-de-conduccion/

[11] Colaboradores de Wikipedia, «Flight simulator», *Wikipedia*, ago. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Flight_simulator

[12] Colaboradores de Wikipedia, «Racing game», *Wikipedia*, ago. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Racing_game

[13] «Mechanical Yacht Race», *BoardGameGeek*. https://boardgamegeek.com/boardgame/108054/mechanical-yacht-race

[14] «Attract Mode: The rise and fall of Coin-Op arcade Games», *Google Books*. https://books.google.es/books?id=d6wCEAAAQBAJ&pg=PA18&redir_esc=y#v=onepage&q&f=false

[15] «1941 International Mutoscope Drive Mobile Drive-Mobile operated Driving Arcade game». http://www.pinrepair.com/arcade/mdrivey.htm

[16] «Classic Home Video Games, 1972-1984», *Google Books*. https://books.google.es/books?id=BzxTtml8Jq4C&pg=PA253&redir_esc=y#v=onepage&q&f=false

[17] VoxOdyssey, «Wipeout on the Magnavox Odyssey the first home game console», *Vox Odyssey*. https://voxodyssey.com/magnavox-odyssey/wipeout

[18] Colaboradores de Wikipedia, «Rally-X», *Wikipedia*, ago. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Rally-X

[19] Colaboradores de Wikipedia, «Turbo (video game)», *Wikipedia*, abr. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Turbo_(video_game)

[20] L. Reilly, «The top 10 most influential racing games ever - IGN», *IGN*, may 2017, [En línea]. Disponible en: https://www.ign.com/articles/2015/04/03/the-top-10-most-influential-racing-games-ever?page=2

[21] J. García, «Historia visual de los Juegos de Carreras», *IGN España*, 30 de marzo de 2015. [En línea]. Disponible en: https://es.ign.com/reportaje/92281/feature/historia-visual-de-los-juegos-de-carreras

[22] Colaboradores de Wikipedia, «Pole position», *Wikipedia*, jul. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Pole_Position

[23] GPLaps, «The birth of SiM Racing - REVS», *YouTube*. 4 de julio de 2021. [En línea]. Disponible en: https://www.youtube.com/watch?v=kvNeGH50nuQ

[24] Colaboradores de Wikipedia, «Out run», *Wikipedia*, jul. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Out_Run

[25] S. M. Wiki, «Super Mario Kart - Super Mario Wiki, The Mario Encyclopedia», *Super Mario Wiki*, ago. 2023, [En línea]. Disponible en: https://www.mariowiki.com/Super_Mario_Kart

[26] Colaboradores de Wikipedia, «Hard drivin'», *Wikipedia*, jul. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Hard_Drivin%27

[27] Colaboradores de Wikipedia, «Sim Racing», *Wikipedia*, ago. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Sim_racing

[28] G. Goble, «History of Papyrus Racing Games», *GameSpot*, 8 de junio de 2005. [En línea]. Disponible en: https://www.gamespot.com/articles/history-of-papyrus-racing-games/1100-6103365/

[29] Morfeus; admin@retrogames.cz, «Formula One Grand Prix (DOS) - online game | RetroGames.cz», *RetroGames.cz*. https://www.retrogames.cz/play_454-DOS.php

[30] T. Moore, «Top 5 best Mario Kart Wii courses: A definitive ranking - The News Wheel», *The News Wheel*, 13 de abril de 2017. https://thenewswheel.com/top-5-best-mario-kart-wii-courses-a-definitive-ranking/

[31] P. Stubbs, «Design Analysis of Everyday Thing: Nintendo Wii Remote», *philstubbswriter.files.wordpress.com*, 7 de febrero de 2013. https://philstubbswriter.files.wordpress.com/2017/01/design-analysis-wii.pdf

[32] A. Staff, «1998 - the beginning of a car obsession», *Automoblog*, 18 de febrero de 2022. [En línea]. Disponible en: https://www.automoblog.net/1998-beginning-car-obsession/

[33] «Gran Turismo 4 Review - IGN», *IGN*, 5 de marzo de 2021. https://www.ign.com/articles/2005/02/23/gran-turismo-4-review

[34] «Gran Turismo (2023) ⭐ 7.4 | Action, Adventure, Drama», *IMDb*, 25 de agosto de 2023. https://www.imdb.com/title/tt4495098/?ref_=tt_mv_desc

[35] Colaboradores de Wikipedia, «Mario Kart DS», *Wikipedia*, ago. 2023, [En línea]. Disponible en: https://en.wikipedia.org/wiki/Mario_Kart_DS#

[36] C. T. N. Wiki, «Stadium», *Nadeo Wiki*, [En línea]. Disponible en: https://nadeo.fandom.com/wiki/Stadium

[37] «F1 22: How to choose the perfect difficulty and assists | Strat Packer's blog», *Strat Packer's blog*, 4 de noviembre de 2022. https://stratpack.blog/2022/11/04/f1-22-how-to-choose-perfect-difficulty-assists

[38] «Assetto corsa on Steam». https://store.steampowered.com/app/244210/Assetto_Corsa/?gclid=Cj0KCQjw_5unBhCMARIsACZyzS0c4z1jAQMvJqii67yYDRtSyNKFAUb_-tEel7GZJjVdeHp_FmkO2F8aAsoPEALw_wcB

[39] Verstoppen, «Max Verstappen SIM Racing "It's not just a game"», *YouTube*. 13 de julio de 2023. [En línea]. Disponible en: https://www.youtube.com/watch?v=_1192lTMZr0

[40] «Unity Asset Store - The best assets for game making», *Unity Asset Store*. https://assetstore.unity.com/

[41] U. Technologies, «Unity - Manual: Unity User Manual 2022.3 (LTS)». https://docs.unity3d.com/Manual/index.html

[42] Angel, «Vehicle Physics Pro». https://vehiclephysics.com/

[43] Toyful Games, «Making custom car physics in unity (for very very valet)», *YouTube*. 8 de julio de 2022. [En línea]. Disponible en: https://www.youtube.com/watch?v=CdPYlj5uZeI

[44] Ingenierizando, «Movimiento Armónico Simple (MAS)», *Ingenierizando*, mar. 2023, [En línea]. Disponible en: https://www.ingenierizando.com/cinematica/movimiento-armonico-simple-mas/

[45] «Oscilaciones amortiguadas». http://www.sc.ehu.es/sbweb/fisica/oscilaciones/amortiguadas/amortiguadas.htm

[46] J. Davis, «Understanding oversteer and understeer», *Maxxis Tyres Australia*, ago. 2022, [En línea]. Disponible en: https://maxxistyres.com.au/understanding-oversteer-and-understeer/

[47] «Before you continue to YouTube». https://www.youtube.com/shorts/qHrgYF_Oc_U

[48] Hutthew., «Verstappen's Drifting Skillz | F1 Brazil 2014 [HD]», *YouTube*. 19 de marzo de 2017. [En línea]. Disponible en: https://www.youtube.com/watch?v=gUOcdVQW9mw

[49] A. Cooper, «Mercedes en el "peor extremo" del desgaste de neumáticos», *Motorsport*, 10 de agosto de 2020. [En línea]. Disponible en: https://lat.motorsport.com/f1/news/mercedes-peor-extremo-desgaste-neumaticos/4853927/

[50] «Ferrari Formula 1 Free 3D Model - .C4D .OBJ .FBX - Free3D». https://free3d.com/3d-model/ferrari-formula-1-72527.html

[51] «Modular Lowpoly Track Roads FREE | 3D Roadways | Unity Asset Store», *Unity Asset Store*, 14 de diciembre de 2021. https://assetstore.unity.com/packages/3d/environments/roadways/modular-lowpoly-track-roads-free-205188

[52] «Unity lap timer». https://www.homeandlearn.co.uk/games-programming/unity-lap-timer.html

[53] NightTime Developments, «Load Different Scenes/Levels using "Main Menu" tutorial | Unity 3D», *YouTube*. 14 de abril de 2022. [En línea]. Disponible en: https://www.youtube.com/watch?v=rcROe0lFYMY

[54] K. Ur, «F1 22 Strategy Guide - How to dominate every race in F1 22 - KeenGamer», *KeenGamer*, 16 de enero de 2023. https://www.keengamer.com/articles/guides/f1-22-strategy-guide-how-to-dominate-every-race-in-f1-22/

[55] Scott, «How to trail brake», *Driver61*, ene. 2022, [En línea]. Disponible en: https://driver61.com/uni/trail-braking/

[56] «Volantes y pedales Logitech G29 Driving Force». https://www.logitechg.com/es-es/products/driving/driving-force-racing-wheel.html

[57] S. M. Wiki, «Mario Kart 7 - Super Mario Wiki, The Mario Encyclopedia», *Super Mario Wiki*, ago. 2023, [En línea]. Disponible en: https://www.mariowiki.com/Mario_Kart_7

[58] «Green in 11:01.740 by Klagarn - TrackMania Nations Forever - Speedrun». https://www.speedrun.com/tmnf/runs/z038x19z

[59] iRacing.com Motorsport Simulations, «AI roster Management - iRacing.com», *iRacing.com*, 16 de agosto de 2023. https://www.iracing.com/ai-rosters/