Universidad Europea

UNIVERSIDAD EUROPEA DE MADRID

FACULTAD DE DISEÑO Y TECNOLOGÍAS CREATIVAS

GRADO EN DISEÑO DE VIDEOJUEGOS

PROYECTO DE FIN DE GRADO

Desarrollo e Implementación de un Sistema de Físicas en Red utilizando Nvidia PhysX y Netcode for GameObjects sobre Unity 6

David Tabernero Mateo

Dirigido por

Luis Gutiérrez Tamurejo

CURSO 2024-2025

TÍTULO: Desarrollo e Implementación de un Sistema de Físicas en Red utilizando Nvidia PhysX y Netcode for GameObjects sobre Unity 6

AUTOR: David Tabernero Mateo

TITULACIÓN: Grado en Diseño de Videojuegos

DIRECTOR/ES DEL PROYECTO: Luis Gutiérrez Tamurejo

FECHA: Junio de 2025

RESUMEN

Este trabajo de fin de grado aborda la implementación del sistema de físicas Nvidia PhysX incluída en el motor de Unity en su versión 6 para su integración en entornos y videojuegos en red en tiempo real utilizando el paquete de herramientas oficial Netcode for GameObjects disponible en los repositorios oficiales de Unity.

El estudio comienza con un análisis profundo de la evolución de los sistemas de red en videojuegos desde sus primeras implementaciones hasta las tecnologías más actuales, así como las técnicas más comunes presentes en los juegos recientes.

Posteriormente, se lleva a cabo otro análisis sobre el funcionamiento de las herramientas citadas de manera individual para proponer una implementación que permita su funcionamiento conjunto de manera eficiente y de propósito general aplicable a distintos tipos de proyectos de videojuegos.

Finalmente se desarrolla un prototipo de videojuego de carreras en el que se implementa el sistema de físicas en red propuesto demostrando su viabilidad y funcionamiento en un entorno de videojuegos real.

Palabras clave: *Sistema de físicas, Nvidia PhysX, Netcode, Netcode for GameObjects, Carreras.*

ABSTRACT

This final degree project addresses the implementation of the Nvidia PhysX physics system in Unity 6 for its integration into real-time networked environments and video games using the official Netcode for GameObjects toolset available in Unity's official repositories.

The study begins with an in-depth analysis of the evolution of networking systems in video games, from their earliest implementations to the most recent technologies, as well as the most common techniques used in modern games.

Subsequently, another analysis is conducted on the individual functionality of the mentioned tools to propose an implementation that enables their joint operation in an efficient and general-purpose manner, applicable to various types of projects.

Finally, a racing video game prototype is developed, incorporating the proposed networked physics system to demonstrate its viability and proper functionality in a real game development environment.

Keywords: *Physics System, Nvidia PhysX, Netcode, Netcode for GameObjects, Racing.*

Agradecimientos

A mi **hermana**, por haber sido mi mayor apoyo y la persona a la que más quiero en el mundo. A mis **padres**, por su amor incondicional y por haberme brindado siempre su apoyo. A mi **familia**, por haber estado a mi lado desde siempre. A todos mis **amigos**, por haber estado aguantándome durante tantos años. Al ***«caballero lobo del mediodía»***, por ser un gran referente para mi tanto a nivel personal como profesional. A mi ***socia responsable***, *meow meow meow meow meow meow meow meow meow meow*. Y a mis **profesores**, por ser mis mentores a lo largo de toda mi trayectoria.

Infinitas gracias a todos.

Comentarios

Para quien esté leyendo esto:

Disfruta cada día como si fuera el último, nunca pierdas la ilusión de hacer cosas nuevas, aprender y divertirte con la alegría de un niño, pues son muchos los que mueren a los 25 y no son enterrados hasta los 75.

> *"If you carry your childhood with you, you never become older."*
>
> —Abraham Sutzkever

TABLA RESUMEN

	DATOS
Nombre y apellidos:	David Tabernero Mateo
Título del proyecto:	Desarrollo e Implementación de un Sistema de Físicas en Red utilizando Nvidia PhysX y Netcode for GameObjects sobre Unity 6
Directores del proyecto:	Luis Gutiérrez Tamurejo
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto: (esta entrada se puede marcar junto a la siguiente)	SÍ
El proyecto ha consistido en el desarrollo de una investigación o innovación: (esta entrada se puede marcar junto a la anterior)	SÍ
Objetivo general del proyecto:	El desarrollo e implementación de un sistema general de físicas en red en tiempo real utilizando el framework de Unity.

Índice

RESUMEN **3**
ABSTRACT **4**
Agradecimientos 5
Comentarios 5
TABLA RESUMEN **6**
Índice 7
Capítulo 1. INTRODUCCIÓN **15**
1.1 Contexto y justificación 15
1.2 Planteamiento del problema 16
Nota 16
1.3 Objetivos del proyecto 17
Objetivos generales 17
Objetivos específicos 17
Objetivos opcionales 17
Capítulo 2. ANTECEDENTES **18**
2.1 Contexto 18
2.1.1 Videojuegos 18
2.1.1.1 Los inicios (1940-1970) 18
2.1.1.2 La era de los arcades y las consolas (1970-1980) 19
2.1.1.3 La crisis de los videojuegos (1980-1990) 20
2.1.1.4 La era 3D y el auge de los juegos en CD (1990-2000) 21
2.1.1.5 La era moderna (2000-2010) 22
2.1.1.6 La era actual (2010-presente) 22
2.1.2 Diseño de videojuegos 24
El enfoque Mecánicas-Dinámicas-Estéticas (Mechanics-Dynamics-Aesthetics o MDA) 24
2.1.3 Simulación de físicas en videojuegos 25
Las librerías de simulación de físicas en videojuegos 26
Nota 27
2.1.4 Videojuegos de carreras con físicas 27
2.1.4.1 Tracción: motor y transmisión 27
2.1.4.2 Suspensión: muelle y amortiguador 29
2.1.4.3 Dirección: llantas y neumáticos 30
2.1.5 Videojuegos en red 32
2.1.5.1 Arquitectura cliente-servidor (server-client architecture) 34

2.1.5.2 Predicción del lado del cliente (client-side prediction)..... 34
2.1.5.3 Compensación de Latencia (Lag Compensation)..... 35
2.1.5.4 Interpolación y Extrapolación..... 36
2.1.5.5 Nota de consideraciones en el diseño del juego.....37
2.1.6 Videojuegos con físicas en red.....38
2.1.6.1 Sincronización de estado en física de red..... 38
2.1.6.2 Optimización del envío de estados..... 38
2.1.6.3 Control de ancho de banda y adaptabilidad..... 38
2.1.6.4 Manejo del jitter en la recepción..... 39
2.1.6.5 Suavizado visual para ocultar errores.....39
2.1.7 Anexo: Físicas deterministas en red (Deterministic lockstep).....40
2.1.7.1 Determinismo en la simulación.....40
2.1.7.2 Envío de entradas en red..... 40
2.1.7.3 Buffer de retraso de reproducción (playout delay buffer)..... 41
2.1.7.4 Limitaciones del TCP..... 41
2.1.7.5 Alternativa con UDP..... 41
2.2 Referentes..... 42
2.2.1 Rocket League.....42
Motor de física..... 42
Sistema de vehículos.....43
Red y predicción de entradas.....44
2.2.2 Wipeout Pulse..... 45
Contenido.....45
Jugabilidad..... 46
2.2.3 Trailmakers.....47
Construcciones y física..... 47
Modos de juego..... 48
Multijugador..... 48
Comunidad.....48
2.2.4 TrackMania Nations Forever.....49
Contenido.....49
Jugabilidad..... 50
Multijugador online..... 50
Capítulo 3. DESARROLLO DEL PROYECTO.....51
3.1 Planificación del proyecto..... 51
3.2 Descripción de la solución, metodologías y herramientas empleadas..... 53
3.2.1 Contexto del proyecto..... 53
3.2.2 Problemas presentes..... 55

3.2.2.1 Falta de una solución oficial integrada....56
3.2.2.2 Limitaciones en la sincronización de rigidbodies en red....56
3.2.2.3 Dilema entre autoridad del servidor y autoridad del cliente....56
3.2.2.4 Ausencia de mecanismos oficiales de predicción de cliente y compensación de lag....57
3.2.2.5 Consumo de ancho de banda innecesario....58
3.2.2.6 Frecuencia de actualización de físicas....58
3.2.3 Desarrollo de la solución....59
3.2.3.1 Prototipo de Protovolkers....59
3.2.3.2 Sistema de físicas en red: corpus y anima....66
3.2.3.3 Integración en el proyecto....72
3.2.4 Resultado final....74
3.3 Recursos requeridos....76
3.3.1 Hardware....76
Nota....76
3.3.2 Software....77
3.4 Viabilidad e implementación....79
3.4.1 Unity Engine....79
3.4.2 Unity Cloud....80
Nota....80
3.4.3 GitHub....81
3.4.4 Steam (Steamworks)....82
3.5 Resultados del proyecto y análisis....83
3.5.1 Analíticas....83
3.5.2 Comparativa....85
Capítulo 4. CONCLUSIONES....87
4.1 Conclusiones del trabajo....87
4.2 Conclusiones personales....87
Capítulo 5. FUTURAS LÍNEAS DE TRABAJO....88
5.1 El producto....88
5.1.1 Generalización....88
5.1.2 Optimización....88
5.1.3 Distribución....88
5.2 El proyecto....90
5.2.1 Refactorización....90
5.2.2 Finalización....90
5.2.3 Distribución....90
5.3 El estudio....92

5.3.1 Dragon Resonance.....92
5.3.2 Makoro.....92
5.3.3 Presencia en internet.....94
5.3.3.1 Página web principal.....94
5.3.3.2 Redes sociales como Makoro.....94
5.3.3.3 Redes sociales como estudio.....95
Capítulo 6. BIBLIOGRAFÍA.....96
Capítulo 7. ANEXOS.....100

Índice de Figuras

Figura 1. Cathode-Ray Tube Amusement Device....18
Figura 2. Space Invaders....19
Figura 3. Sega Genesis....20
Figura 4. The Legend of Zelda: Ocarina of Time....21
Figura 5. World of Warcraft....22
Figura 6. Steam 20th Anniversary....23
Figura 7. Fórmula de la fuerza de un motor....27
Figura 8. Curva de fuerza de torsión de un motor respecto a sus RPM....28
Figura 9. Fórmula de la fuerza de torsión de la transmisión....28
Figura 10. Curvas de las fuerzas de torsión de una caja de cambios de 6 marchas....29
Figura 11. Fórmula de Hooke....29
Figura 12. Diagrama y fórmula de Ackermann aplicada al ángulo de giro....30
Figura 13. Curva de deslizamiento de los neumáticos de Pacejka....31
Figura 14. QuakeWorld....32
Figura 15. Dune II....32
Figura 16. Need for Speed Road Challenge....33
Figura 17. Diagrama simplificado de una arquitectura cliente-servidor....34
Figura 18. Diagrama simplificado de ejemplo de latencia, predicción y corrección....35
Figura 19. Diagrama de secuencia de la transmisión de una acción en red....36
Figura 20. Diferencia entre interpolación y extrapolación....37
Figura 21. Rocket League....42
Figura 22. Diferencia entre las fuerzas ejercidas sobre una rueda de vehículo....43
Figura 23. Jared Cone exponiendo el sistema de red de Rocket League....44
Figura 24. Wipeout Pulse....45
Figura 25. Shipedit....46
Figura 26. Trailmakers....47
Figura 27. Workshop de Trailmakers....48
Figura 28. TrackMania Nations Forever....49
Figura 29. TMNF-X (TrackMania Nations Forever Exchange)....50
Figura 30. Diagrama Gantt de tareas en Notion....51
Figura 31. Cartel promocional de Protovolkers en 2024....53
Figuras 32 y 33. Capturas del desarrollo de Protovolkers en 2022 y 2024....53
Figura 34. Diagrama de la primera adaptación del flujo de escenas a un juego en red de Protovolkers....54
Figuras 35 y 36. Steffen Itterheim explicando el problema de la transmisión de físicas en red....55

Figura 37. Vehículo en red controlado por otro cliente mediante NetworkTransform.....56
Figura 38. Diferencia de posición del vehículo en el cliente respecto al servidor..... 57
Figuras 39 y 40. Configuración de tickrate de físicas a 50 *tick*/s respecto a una configuración liviana a 30 *tick*/s..... 58
Figura 41. Implementación del sistema de suspensión vehicular..... 59
Figuras 42 y 43. Implementación del sistema de agarre de las ruedas..... 60
Figura 44. Implementación del sistema de giro independiente por rueda.....61
Figura 45. Descripción del sistema de giro de Ackermann..... 61
Figura 46. Diagrama de clases de la primera versión del prototipo de Protovolkers.....62
Figura 47. Diagrama simplificado de Modelo-Vista-Controlador..... 62
Figura 48. Diagrama de clases de la segunda versión del prototipo de Protovolkers.....63
Figura 49. Diagrama de clases de la versión final del prototipo de Protovolkers..... 64
Figura 50. Diagrama de flujo de escenas de la primera versión del prototipo de Protovolkers.. 65
Figura 51. Diagrama de flujo de escenas de la futura versión de Protovolkers..... 66
Figuras 52 y 53. Jerarquía del vehículo en red con la separación de cuerpo para la simulación local, y alma para la simulación en remoto..... 67
Figura 54. El método de actualización de las físicas efectúa actualizaciones según si la instancia del vehículo es propietaria, observadora, o gestionada por el servidor.....67
Figura 55. Los métodos para enviar el estado del vehículo al servidor, actualizarlo desde el servidor, o solicitar a todos los clientes que actualicen los estados de los contrincantes.....68
Figura 56. Los métodos para sincronizar los estados del *anima* tanto del servidor como de los clientes en métodos independientes..... 68
Figura 57. Comando de resincronización para igualar *corpus* a *anima*..... 69
Figura 58. Visualización de cuerpo y alma parcialmente solapados.....69
Figura 59. Sistema de comprobación para la reconciliación.....69
Figura 60. Matriz de la configuración de físicas para las capas de *corpus* y *anima*..... 70
Figura 61. El componente Rigidbody del *anima* no hace uso de la gravedad ni colisiona con otras capas.....70
Figura 62. Implementación del sistema de físicas en red..... 71
Figura 63. Implementación del sistema de físicas en red en los vehículos.....71
Figura 64. Implementación del flujo de juego básico del prototipo..... 72
Figura 65. Optimización y depuración de los sistemas de control del vehículo para su funcionamiento en red..... 73
Figura 66. Diseño y automatización de un circuito básico en Blender..... 73
Figura 67. Importación automatizada del circuito en Unity..... 74
Figura 68. Prueba de funcionamiento de un único vehículo con físicas en red.....74
Figura 69. Prueba de funcionamiento completa del prototipo de Protovolkers..... 75
Figura 70. Captura del resumen de consumo de ancho de banda para el proyecto Protovolkers en la web de Unity Cloud.....75
Figura 71. Condiciones de los planes de Unity según la situación del desarrollador..... 79

Figura 72. Lista de precios de los planes ofrecidos por Unity.....79
Figura 73. Plan de precios para el servicio Relay, parte de Unity Cloud..... 80
Figura 74. Plan de precios para el servicio Lobby, parte de Unity Cloud..... 81
Figuras 75 y 76. Resumen del consumo de disco y ancho de banda por repositorio de GitHub y gestor de paquetes de datos y precios..... 81
Figuras 77 y 78. Entrada dedicada al pago del Steam Direct Fee para la publicación de productos en Steam y su devolución.....82
Figura 79. Administrador de tareas de Windows mostrando los consumos de Unity (en Editor) durante una partida multijugador..... 83
Figuras 80 y 81. Administrador de tareas de Windows mostrando el consumo de memoria y ancho de banda de Unity durante una partida multijugador..... 83
Figura 82. Profiler de Unity durante la ejecución de una partida multijugador..... 84
Figuras 83 y 84. Contador y gráfica de consumo de ancho de banda global a través del servicio Relay..... 84
Figuras 85 y 86. Gráficas de consumo de ancho de banda por CCU y espacios asignados a través del servicio Relay..... 85
Figura 87. Repositorio de Unity DevKit Package de Dragon Resonance..... 89
Figura 88. Repositorio de Unity ToolSet Package de Dragon Resonance.....89
Figura 89. Set de iconos de Protovolkers para múltiples resoluciones de pantalla..... 90
Figura 90. Página de Protovolkers en Steam.....91
Figura 91. Página de Protovolkers en itch.io.....91
Figura 92. Logotipo principal de Dragon Resonance..... 92
Figuras 93 y 94. Icono de Makoro anterior y posterior a su creación.....92
Figura 95. Ficha del diseño de personaje de Makoro..... 93
Figura 96. Apartado de proyectos de la página web..... 94
Figuras 97, 98, 99 y 100. Presencia como Makoro en X, Bluesky, Mastodon e Instagram.....94
Figuras 101, 102, 103 y 104. Presencia como estudio en YouTube, Steam Developers, GitHub y GitLab..... 95

Índice de Tablas

Tabla 1. Desglose de tareas del proyecto y Trabajo de Fin de Grado.....51
Tabla 2. Diferencias entre la autoridad del servidor y la autoridad del cliente..... 56
Tabla 3. Ficha técnica del equipo anterior a la actualización.....76
Tabla 4. Ficha técnica del equipo posterior a la actualización.....76
Tabla 5. Ficha del software local empleado.....77
Tabla 6. Ficha de los lenguajes de programación empleados.....77
Tabla 7. Ficha de los servicios de software empleados..... 77
Tabla 8. Tabla de consumos de ancho de banda de videojuegos online conocidos..... 85

Capítulo 1. INTRODUCCIÓN

Se ha abordado la implementación del sistema de físicas Nvidia PhysX en el motor Unity 6, integrándolo en entornos y videojuegos en red en tiempo real mediante el uso del paquete oficial Netcode for GameObjects.

Para ello se ha realizado un análisis exhaustivo de la evolución de los sistemas de red en videojuegos desde sus primeras implementaciones hasta las tecnologías más avanzadas utilizadas en la actualidad. Asimismo se han estudiado las técnicas más comunes en los videojuegos modernos para gestionar la sincronización de físicas en entornos multijugador.

Posteriormente se han examinado las herramientas involucradas y se ha desarrollado una variante que permite su integración adaptable a distintos tipos de proyectos de videojuegos. Como prueba de ello, se ha desarrollado un prototipo funcional de un videojuego de carreras integrando este mismo sistema de físicas en red.

1.1 Contexto y justificación

A lo largo de los años los videojuegos han experimentado una evolución constante tanto en la complejidad como en las técnicas y mecánicas empleadas en su desarrollo y funcionalidad. Como consecuencia de estos avances, han surgido librerías especializadas en la simulación de físicas pseudorrealistas, permitiendo a las empresas desarrolladoras integrar comportamientos de físicas más realistas en sus videojuegos.

A raíz de estas librerías no solo se busca mejorar la experiencia visual, sino también se ha visto el surgimiento de videojuegos que las toman como el núcleo de la jugabilidad y la mecánica principal con la que interaccionar y sobre la que funcionar.

Paralelamente la tecnología de networking ha evolucionado de manera exponencial, aumentando tanto la tasa de transferencia de datos como la velocidad de transmisión de los mismos. Esto, sumado a lo anterior, ha permitido el desarrollo y la evolución de videojuegos con componentes de red donde la comunicación entre jugadores ocurre de manera cada vez más habitual.

Sin embargo, la naturaleza de los videojuegos en red varía enormemente según su nivel de exigencia en cuanto a sincronización y latencia: mientras que algunos títulos pueden tolerar tiempos de espera más amplios y una menor frecuencia de actualización, otros requieren la transmisión de datos en tiempo real para garantizar una experiencia agradable para el jugador.

La intersección entre estas dos áreas ha dado lugar a un desafío técnico especialmente complejo: **la implementación de físicas en red en tiempo real**. Esto implica la sincronización de manera eficiente de cálculos físicos entre múltiples jugadores sin comprometer la jugabilidad ni la estabilidad de la red.

1.2 Planteamiento del problema

Partiendo de este contexto, surge el siguiente reto técnico al que es necesario dar solución: En el caso particular de Unity el motor de físicas empleado por defecto es Nvidia PhysX como ya se ha mencionado. Si bien este motor es una herramienta excepcional debido a su precisión, un realismo bastante creíble, y buen rendimiento, así como ser gratuita y de código abierto, presenta de la misma manera una particularidad que complica su uso en entornos multijugador específicos: **no es un sistema de físicas determinista**. Esto significa que pequeñas variaciones en los cálculos pueden generar resultados distintos en cada dispositivo, lo que dificulta la sincronización en un entorno de red.

Por otro lado, la transmisión de datos en red introduce un factor de latencia inevitable que afecta directamente la fluidez del juego y a la experiencia del jugador, incluso con conexiones de alta velocidad.

Ambas partes en conjunto afectan directamente a un tercer elemento de la ecuación: la autoridad dentro de la red. Dependiendo de quién controle la simulación de las físicas, surgen dos escenarios distintos:

A. Si la autoridad la tiene el servidor, por un lado el juego es menos vulnerable a trampas y manipulaciones por parte de sus jugadores, ya que todas las interacciones pasan por un control centralizado considerado “neutral”. Por otro lado, esta configuración introduce un retraso en la respuesta de las físicas, ya que cada acción debe viajar hasta el servidor y volver antes de reflejarse en el juego.
B. Si la autoridad recae en cada usuario, la ejecución de las físicas de juego es instantánea y más fluida ya que las interacciones físicas se procesan de manera local sin depender de la red. Sin embargo, esta arquitectura es altamente vulnerable a trampas, ya que los datos pueden ser alterados por los jugadores sin una validación externa.

Así surge la necesidad de investigar y desarrollar un sistema propio que logre sacar el mayor provecho de ambas vertientes intentando paliar sus defectos en la medida de lo posible.

Idealmente, y como objetivo adicional, el sistema debería poder ser desarrollado de manera general para poder ser implementado en múltiples tipos de proyecto de videojuego en red.

Nota

Este proyecto tiene un propósito innovador y no está respaldado ni impulsado por ningún tipo de empresa u organización. Todo el desarrollo ha sido ejecutado bajo mi entera responsabilidad.

1.3 Objetivos del proyecto

En resumen, y habiendo especificado todo lo anterior, los siguientes son los objetivos generales y específicos que se pretenden abordar con este trabajo.

Objetivos generales

- Estudiar y entender el funcionamiento de los videojuegos con mecánicas basadas en físicas.
- Estudiar y entender el funcionamiento de los videojuegos en red de última generación, así como sus diferentes implementaciones según el género.
- Estudiar y analizar las técnicas más habituales para el desarrollo de videojuegos con físicas en red.
- Plantear un sistema potencialmente funcional compatible con las librerías mencionadas y recurriendo a la menor cantidad de recursos y librerías ajenas a las mencionadas.

Objetivos específicos

- Crear un sistema de físicas en red funcional.
- Crear un sistema de compensación de la latencia a través de la predicción tanto de la posición como del movimiento de los elementos (principalmente jugadores, pero no necesariamente limitado a estos).

Objetivos opcionales

- Desarrollar y adaptar un sistema de funcionamiento de propósito general compatible con múltiples tipos de proyecto.
- Crear un prototipo de videojuego (en este caso, un juego de carreras) que implemente el sistema desarrollado de manera satisfactoria.

Capítulo 2. ANTECEDENTES

2.1 Contexto

2.1.1 Videojuegos

Los videojuegos son aplicaciones interactivas orientadas al entretenimiento que se ejecutan en dispositivos electrónicos como ordenadores, videoconsolas, móviles, etc. Consisten en entornos virtuales donde uno o varios jugadores interactúan con el sistema a través de unos controles siguiendo reglas y objetivos específicos.

No ha sido hasta las últimas generaciones cuando los videojuegos han logrado no solo adquirir una extraordinaria importancia como medio, sino formar parte de la vida de muchísimas personas a nivel social y cultural (Berens & Howard, 2008b).

La historia de los videojuegos está dividida en varias épocas marcadas por los avances tecnológicos que han ido permitiendo su evolución. En los siguientes puntos se verá un breve resumen de los momentos más notables (Donovan & Garriot, 2010).

2.1.1.1 Los inicios (1940-1970)

En esta época se observa el desarrollo de las primeras computadoras electrónicas y la capacidad de procesamiento básico.

Figura 1. *Cathode-Ray Tube Amusement Device*. Fuente: CitizenSide.

Las primeras invenciones que podrían considerarse "videojuegos" son aquellas que surgieron como experimentos académicos y militares. La figura 1 muestra el *Dispositivo de Entretenimiento de Tubos de Rayos Catódicos* o "*Cathode-Ray Tube Amusement Device*", un extraño dispositivo creado en 1947 capaz de generar rayos luminosos en una pequeña pantalla circular con juegos sencillos como un pequeño simulador de disparos espacial o un laberinto (Tester, 2023).

Fue en 1958 cuando William Higinbotham desarrolló *Tenis para Dos* o "*Tennis for Two*", un sencillo juego que simulaba un partido de tenis, y que sería el que más tarde se conocería como *Pong* en la versión comercializada por Atari.

En 1962, Steve Russell creó "Spacewar!" para el ordenador DEC PDP-1, y que se convirtió en el primer videojuego ampliamente distribuido en el mercado.

2.1.1.2 La era de los arcades y las consolas (1970-1980)

Los avances tecnológicos en esta época estuvieron marcados por mejoras en los circuitos integrados y la aparición de los microprocesadores.

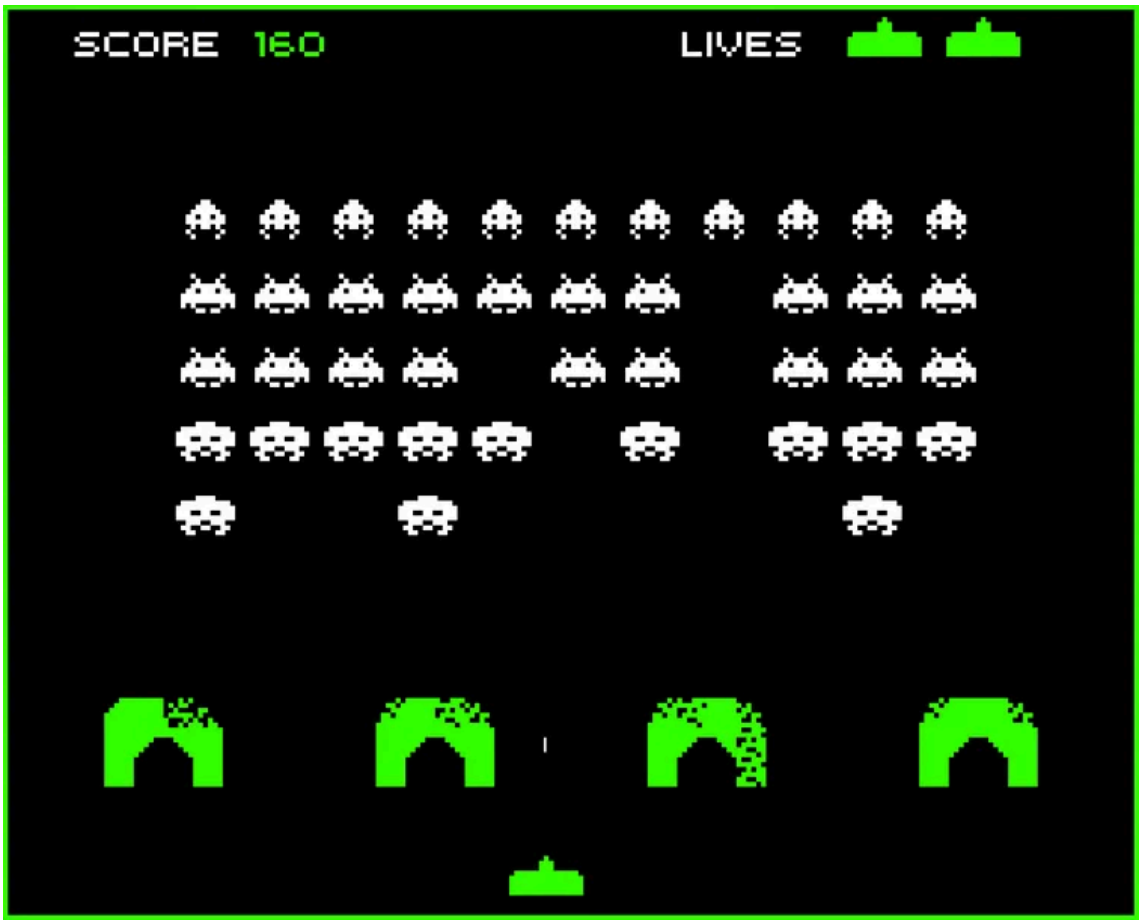


Figura 2. *Space Invaders*. Fuente: Smithsonian.

En 1972, Atari lanzó "*Pong*", el primer videojuego comercial considerado exitoso, y que marcó el inicio de la popularización de los videojuegos. A partir de este momento, las máquinas arcade comenzaron a llenar los salones recreativos, convirtiéndose en un fenómeno social (Kent, 2001).

Posteriormente, en 1977, Atari lanzó la consola *Atari 2600*, revolucionando el mercado al permitir que los jugadores disfrutaran de diferentes títulos desde sus propias casas gracias a su revolucionario sistema de cartuchos intercambiables (Nobes, 2024).

Más tarde, en 1978, el archiconocido videojuego "*Space Invaders*" se convirtió en un referente de los juegos de disparos y ayudó a consolidar la industria junto con "*Pac-Man*", lanzado en 1980. Ambos sentaron un precedente y definieron la cultura pop de la época.

2.1.1.3 La crisis de los videojuegos (1980-1990)

Durante esta etapa, los avances tecnológicos en la industria del videojuego estuvieron marcados por mejoras tanto en materia de gráficos y sonido, así como la introducción de procesadores aún más potentes.

Figura 3. *Sega Genesis*. Fuente: Lifewire.

En 1983, la industria sufrió una crisis debido a la saturación del mercado y la baja cantidad y calidad de algunos videojuegos, lo que provocó una pérdida del interés por parte del mercado (Padden, 2021).

En 1985, Nintendo logró revitalizar el sector con el lanzamiento de la NES (Nintendo Entertainment System). Esta consola, junto con títulos icónicos como "*Super Mario Bros.*", sentó las bases para una nueva era de los videojuegos estableciendo estándares de calidad más altos.

En los años siguientes, la competencia entre Sega y Nintendo impulsó nuevas innovaciones en el sector, como consolas como la *Sega Genesis* y la *Super Nintendo*, las cuales introdujeron gráficos de 16 bits.

2.1.1.4 La era 3D y el auge de los juegos en CD (1990-2000)

Durante este tiempo, la industria experimentó una transformación significativa con la transición de los gráficos 2D a 3D de la mano de una mayor capacidad de procesamiento de gráficos, así como la expansión del uso de CDs como formato de almacenamiento.

Figura 4. *The Legend of Zelda: Ocarina of Time*. Fuente: Zeldapedia.

En 1994, Sony ingresó al mercado con la *PlayStation*. Dos años después, en 1996, Nintendo lanzó la *Nintendo 64*, que destacó por su potencia gráfica y mecánicas innovadoras.

Entre los títulos más emblemáticos de este último se encuentra "*The Legend of Zelda: Ocarina of Time*" en 1998, considerado uno de los mejores videojuegos de la historia por su diseño (Luckerson, 2018).

Durante este periodo, algunos juegos marcaron hitos en la evolución del medio. "*Final Fantasy VII*" sorprendió por su ambicioso apartado narrativo y cinemáticas impactantes, así como "*Tomb Raider*" presentó a *Lara Croft*, uno de los personajes más icónicos de la industria.

2.1.1.5 La era moderna (2000-2010)

Durante esta etapa, el videojuego avanzó con la llegada de la conexión e integración con los servicios de internet, unos gráficos en alta definición y el uso de hardware de procesamiento multinúcleo.

Figura 5. *World of Warcraft*. Fuente: Blizzard.

En 2001, Microsoft ingresó al mercado con la *Xbox*, consolidándose como un nuevo competidor en la industria hasta la actualidad. Mientras tanto, la PlayStation 2 se convirtió en la que sería la consola más vendida de la historia, gracias a su amplio catálogo de juegos, potencia y su capacidad para reproducir DVD, entre otras características.

Durante este período, los juegos en línea cobraron gran relevancia, con títulos como "*World of Warcraft*", que popularizaron el modelo de juego multijugador masivo en línea (MMO).

Los años siguientes vieron el lanzamiento de consolas que marcaron un punto de inflexión. La *Xbox 360* de parte de *Microsoft*, la *PlayStation 3* creada por *Sony*, y la *Wii* del lado de *Nintendo*, no solo mejoraron el apartado gráfico e integraron características como la conectividad en línea, sino que también introdujeron controles innovadores, como sensores de movimiento o captación de interacción por cámara, que permitió nuevas formas de interacción, y mecánicas y diseños de videojuegos.

2.1.1.6 La era actual (2010-presente)

Desde entonces y hasta la actualidad, se han ido viendo los últimos avances tecnológicos que van marcando los videojuegos de nuestros días: la realidad virtual, los gráficos en 4K, la

implementación de inteligencia artificial avanzada y el auge del streaming de juegos, entre otras muchas. Todo ello ha venido acompañado naturalmente de grandes avances en términos de potencia de computación.

Figura 6. *Steam 20th Anniversary*. Fuente: Steam.

En el apartado de las videoconsolas, la *PlayStation 4* y *PlayStation 5*, la *Xbox One* y *Xbox Series*, y la *Nintendo Switch* han dominado el mercado en los últimos años, ofreciendo un catálogo variado y las mejoras equivalentes en términos de potencia y conectividad.

Paralelamente, los juegos móviles han alcanzado audiencias masivas, con títulos como *Pokémon GO* por parte de Nintendo, que combinó la realidad aumentada con la geolocalización para ofrecer una experiencia de juego única (Muñoz, 2019).

La popularización de plataformas de distribución digital como *Steam*, *GOG* o *Epic Games Store* han revolucionado la forma en que los jugadores acceden a los videojuegos, facilitando su compra y descarga a través de internet, sin necesidad de copias físicas.

En la última década, se ha presenciado el lanzamiento de grandes títulos que se han convertido en referentes en la industria. Entre los juegos de los desarrolladores AAA más destacados se encuentran "*The Witcher 3: Wild Hunt*" (2015), "*The Legend of Zelda: Breath of the Wild*" (2017), "*Red Dead Redemption 2*" (2018), "*Elden Ring*" (2022) o "*Baldur's Gate 3*" (2023). Mientras que en el ámbito independiente, destacan "*Hollow Knight*" (2017), "*Celeste*" (2018), "*Hades*" (2020) o "*Inscryption*" (2021).

2.1.2 Diseño de videojuegos

Un videojuego como tal, es un proyecto que puede conllevar años de desarrollo por parte de grandes equipos de perfiles muy dispares: programadores, artistas, modeladores, músicos, etc. Todos estos cargos son absolutamente cruciales para garantizar una experiencia atractiva y divertida para los jugadores, por ello es necesario que detrás de proyectos como estos haya un trabajo de diseño que los tenga en cuenta.

Un buen diseño abarca desde mecánicas intuitivas y una jugabilidad fluida hasta una narrativa envolvente y una progresión bien estructurada. Además, influye en la inmersión, la accesibilidad y la rejugabilidad, asegurando que el juego no solo sea divertido, sino también satisfactorio a largo plazo.

Adicionalmente, las decisiones creativas por parte de los diseñadores están grandemente influenciadas por todos los juegos que han jugado anteriormente (Brown, 2023).

Un grupo de diseñadores colaboraron en la creación de un artículo en el que desglosan una metodología efectiva para la investigación y el diseño de videojuegos. Esta metodología se basa tanto en sus propias experiencias como en el trabajo de otros tantos diseñadores que les precedieron. En los siguientes párrafos se desarrolla la propuesta.

El enfoque Mecánicas-Dinámicas-Estéticas (*Mechanics-Dynamics-Aesthetics* o *MDA*)

El artículo “MDA: A Formal Approach to Game Design and Game Research” introduce un enfoque formal a la hora de analizar y comprender los videojuegos existentes con el objetivo de proporcionar ciertas herramientas para facilitar ciertas decisiones en el diseño de videojuegos futuros. Este modelo fue creado por Hunicke, LeBlanc y Zubek (finalizado en 2004), y mostrado durante el taller *Game Design and Tuning Workshop* durante la *Game Developers Conference* de San José entre 2001 y 2004.

Las siglas MDA corresponden a tres conceptos principales:

- **Mecánicas (*Mechanics*)**: Son el conjunto de todas las reglas que definen el funcionamiento del videojuego a nivel técnico y estructural. Algunos ejemplos de esto pueden ser los turnos de un juego de cartas, la munición de un juego de disparos, o el movimiento del balón en un juego de deporte.
  Ajustar las mecánicas puede mejorar la experiencia de juego. Siguiendo el ejemplo de Monopoly, se podrían introducir mecánicas como impuestos progresivos o subsidios para jugadores con menos dinero, con el objetivo de mantener la competencia durante más tiempo.
  El ajuste de mecánicas es crucial para mejorar la experiencia del jugador. Se pueden realizar pruebas iterativas para equilibrar la dificultad, la progresión y la satisfacción del jugador. De la misma manera, esto implica evaluar cómo los cambios efectuados afectan a la experiencia global y actuar en consecuencia.

- **Dinámicas (*Dynamics*)**: Son el conjunto de comportamientos emergentes que suceden cuando los jugadores interactúan con las mecánicas y con el entorno en tiempo real. Algunos ejemplos de esto incluyen la presión del tiempo en un juego contrarreloj, el compañerismo en un juego cooperativo, o la expresión en una pantalla de personalización de vehículos.
  Las dinámicas pueden ser modeladas para predecir sus efectos en la jugabilidad. Siguiendo el ejemplo del Monopoly, los jugadores ricos se vuelven cada vez más poderosos mientras los pobres se ven penalizados, lo que puede hacer que la partida pierda tensión o interés en etapas avanzadas.
- **Estéticas (*Aesthetics*)**: Son el conjunto de experiencias y respuestas emocionales que los jugadores experimentan cuando son expuestos a las anteriores.
  Uno de los desafíos más recurrentes en el estudio de videojuegos es la falta de un lenguaje claro para describir el "divertimento" o la "jugabilidad" pues son inmensurables.
  El modelo MDA propone una clasificación de objetivos estéticos que ayudan a definir la experiencia deseada: **Sensación**, placer sensorial (gráficos, banda sonora); **fantasía**, escapismo y construcción de mundos imaginarios; **narrativa**, desarrollo de una historia significativa; **desafío**, superación de obstáculos y pruebas de habilidad; **compañerismo**, juegos diseñados para la interacción social; **descubrimiento**, exploración y aprendizaje dentro del juego; **expresión**, creatividad y personalización del jugador; o **sumisión**, juegos como pasatiempo relajante o escapismo.
  Ejemplos de esto son por ejemplo Quake, basado en el desafío y la competencia; The Sims, que apuesta por la narrativa y la expresión; o Final Fantasy, se apoya en la fantasía, la narrativa y el descubrimiento.

El MDA plantea también que los videojuegos son algo parecido a “artefactos” más que medios de comunicación como lo es el cine por ejemplo, pues el contenido de un videojuego no es solo lo que se muestra por pantalla (si la tuviera), sino también el comportamiento que emerge de la interacción entre sus varios sistemas y el jugador.

Como aspecto final, este modelo diferencia la perspectiva entre diseñadores y jugadores: Desde el punto de vista del diseñador, las mecánicas generan dinámicas que conducen a experiencias estéticas; mientras que desde el punto de vista del jugador, la experiencia estética es lo primero que es percibido, e influye en cómo se interpretan las dinámicas y su interacción con las mecánicas.

2.1.3 Simulación de físicas en videojuegos

Desde hace unos años, los videojuegos han contado con librerías específicamente desarrolladas para permitir y facilitar la simulación pseudorrealista de físicas apoyadas en elementos de la mecánica clásica tales como las leyes de Newton y la gravedad, para modelar movimientos y fuerzas en entornos digitales.

Se ha tomado como ejemplo el libro *Teaching Computational Thinking: An Integrative Approach for Middle and High School Learning*, por Neumann, Dion y Snapp (2021) donde se detalla la aplicación de estos conceptos en videojuegos.

En resumen, se enseña el uso de la física en videojuegos de manera práctica, cómo los parámetros son modificables por parte del desarrollador (o el alumno, en este caso) desde un enfoque práctico.

Las librerías de simulación de físicas en videojuegos

El uso de este tipo de librerías ha ido progresando junto con los mismos videojuegos que las utilizaban. La simulación de físicas ha sido un elemento clave en la evolución del realismo y la interactividad en estos entornos, y la existencia de este tipo de herramientas han permitido a los desarrolladores integrar mecánicas físicas avanzadas en sus proyectos sin necesidad de programarlas desde cero.

En los primeros videojuegos, la simulación de físicas era extremadamente básica y, en muchos casos, inexistente. Los títulos de la década del 1970 como "*Pong*" (1972) o "*Donkey Kong*" (1981), implementaron movimientos predefinidos y colisiones simplificadas mediante detección de intersecciones geométricas, pues estos son cálculos sencillos y efectivos.

Uno de los primeros intentos de aplicar una simulación más avanzada fue "*Asteroids*" (1979), que utilizó una aproximación a la física newtoniana tanto para el movimiento de la nave y la destrucción de los asteroides. Sin embargo, estas físicas eran programadas de forma específica para cada juego, sin una librería genérica reutilizable.

A medida que el hardware mejoró, la necesidad de sistemas más sofisticados llevó al desarrollo de motores físicos dedicados. Juegos como "*Doom*" (1993) y "*Quake*" (1996) implementaron sistemas de colisiones en 3D y desplazamientos más realistas, aunque seguían siendo soluciones hechas a medida.

El primer gran avance en librerías físicas reutilizables llegó con *Havok*, lanzado en el año 2000. Este motor, utilizado en títulos como Half-Life 2 (2004) y Halo 2 (2004), permitía simulaciones de colisiones, cuerpos rígidos y *ragdolls* de manera eficiente, marcando el inicio de la integración masiva de motores físicos en la industria.

Actualmente estas las principales librerías de motores de físicas utilizadas en los videojuegos:

- **PhysX**: Adquirido por Nvidia. Motor por defecto en Unity y Unreal Engine 4. Introdujo aceleración por hardware mediante tarjetas gráficas especializadas.
- **Havok**: Adquirido por Microsoft (*Havok to Join Microsoft*, 2015) y utilizado en la mayoría de los juegos AAA.
- **Bullet**: Librería de código abierto integrada de manera nativa en Godot. Utilizada en juegos y algunas aplicaciones sencillas de simulación.
- **Jolt**: Librería nueva de código abierto gratuita utilizada en algunos juegos, camino de reemplazar el motor de físicas de Godot prometiendo estar a la altura de sus competidores (Pomeyrol, 2025).

- **Chaos**: El nuevo motor de Unreal Engine 5 en colaboración con Intel (*Unreal Engine New Chaos Physics System Screams With In-Depth Intel*, 2019) que promete una calidad a la altura de las superproducciones de Hollywood.

Nota

Como ya se ha mencionado anteriormente, este trabajo se centrará de manera exclusiva en el motor Nvidia PhysX integrado por defecto en Unity.

2.1.4 Videojuegos de carreras con físicas

En este apartado se presentan ejemplos específicos de videojuegos que ilustran algunas demostraciones prácticas de algunos de los principios que se verán. En él se aborda la teoría detrás del desarrollo de juegos de carreras basados en físicas, explicando las principales mecánicas que afectan al movimiento de los vehículos.

Todo este apartado está basado en la documentación proporcionada por los recursos "*Car Physics for Games*" (Monster, 2003), "*Development of a car physics engine for games*" (Srisuchat, 2012), y "*Vehicle acceleration and maximum speed modeling and simulation*" (X-Engineer, n.d.).

2.1.4.1 Tracción: motor y transmisión

El motor de un vehículo genera torsión (*torque, T*), la cual es transmitida a las ruedas para generar movimiento. La relación entre esta torsión aplicada y la fuerza ejercida en las ruedas se expresa de la siguiente manera:

$$F_t = \frac{T_e \cdot i_x \cdot i_0 \cdot \eta_d}{r_{wd}} \quad (2)$$

where:

T_e [Nm] – engine torque
i_x [-] – transmission gear ratio
i_0 [-] – final drive ratio
η_d [-] – driveline efficiency
r_{wd} [m] – dynamic wheel radius

Figura 7. *Fórmula de la fuerza de un motor*.

El motor no entrega directamente una fuerza lineal al uso, sino una fuerza de torsión, que depende del régimen de revoluciones por minuto (RPM).

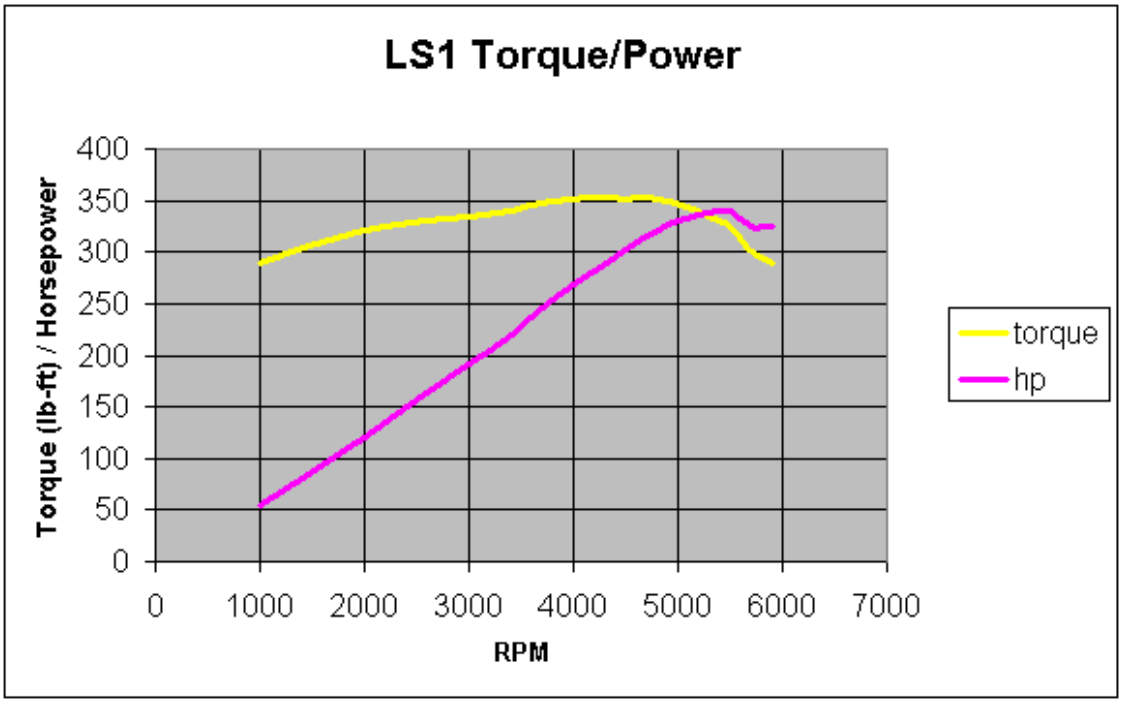


Figura 8. *Curva de fuerza de torsión de un motor respecto a sus RPM*.

La cantidad de torsión que puede entregar un motor en función de las RPM se representa mediante una curva de potencia de torsión, la cual varía según el diseño del motor.

La torsión del motor es transmitida a las ruedas a través de la caja de cambios y el diferencial, los cuales actúan como multiplicadores de ésta. La fuerza que finalmente se aplica a las ruedas se puede calcular con la siguiente ecuación:

$$T_t = i_x \cdot i_0 \cdot T_e \cdot \eta_d \quad (13)$$

where:

T_e [Nm] – engine torque
i_x [-] – transmission gear ratio
i_0 [-] – final drive ratio
η_d [-] – driveline efficiency

Figura 9. *Fórmula de la fuerza de torsión de la transmisión*.

La transmisión introduce pérdidas de energía, principalmente en forma de calor, reduciendo la eficiencia global del sistema.

La fuerza de torsión entregada a las ruedas es inversamente proporcional a la velocidad tangencial de las mismas, o similarmente, a la velocidad del vehículo: En marchas bajas, la relación de transmisión es alta, lo que genera mayor torsión pero menor velocidad; en marchas altas, ocurre lo contrario, menos torsión pero mayor velocidad.

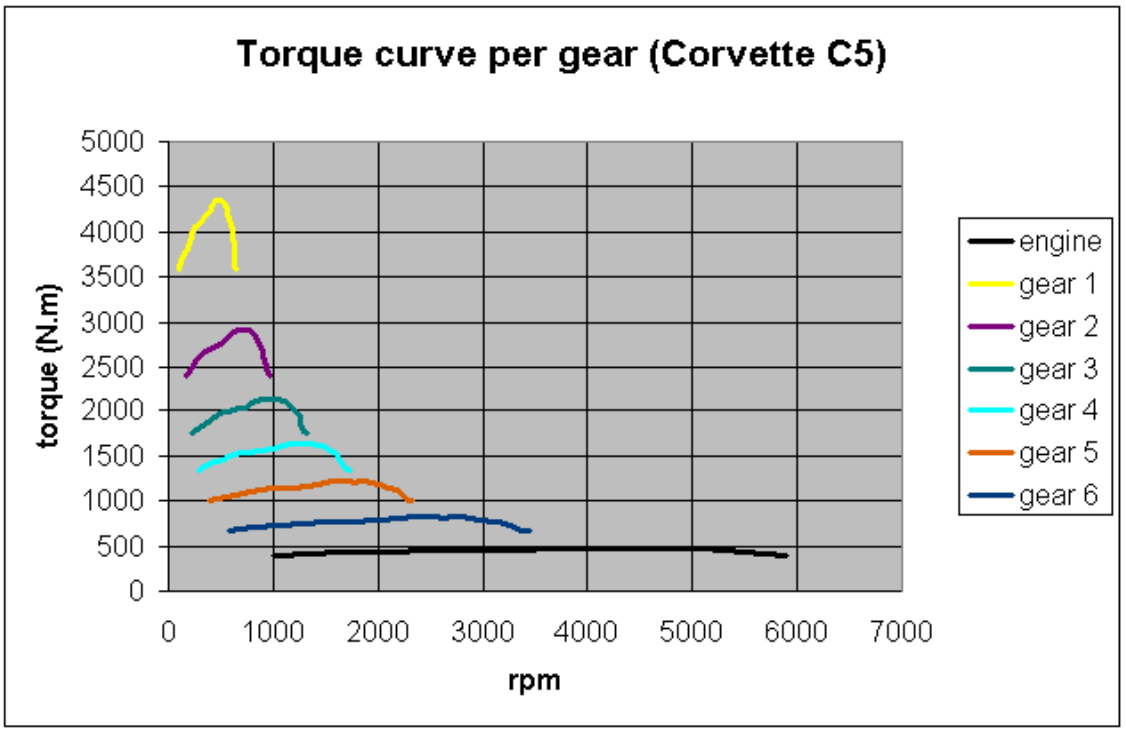


Figura 10. *Curvas de las fuerzas de torsión de una caja de cambios de 6 marchas*.

Si el motor está conectado a las ruedas mediante una transmisión sin deslizamiento (sin embrague accionado), las revoluciones del motor dependen directamente de la velocidad de las ruedas y de la relación de transmisión en ese momento.

Para maximizar la aceleración sin perder tracción, la torsión aplicada debe mantenerse justo por debajo del límite de adherencia de los neumáticos. Si la fuerza de tracción excede la capacidad de agarre, las ruedas derrapan, reduciendo la eficiencia de la aceleración.

Según el grado de realismo deseado para el proyecto, es posible omitir algunos de estos principios menos influyentes y reducir la cantidad y complejidad de los cálculos dentro del videojuego.

2.1.4.2 Suspensión: muelle y amortiguador

El sistema de suspensión de un vehículo se puede desarrollar de manera simplificada utilizando resortes que conectan cada rueda con el chasis del vehículo. Estos resortes generan fuerzas iguales y opuestas en los objetos conectados, siguiendo la Ley de Hooke, según la ecuación:

$$Fs = Ks\ (L - r) \qquad (7\text{-}6)$$

Where Fs is the spring force, Ks is the spring constant, L is the stretched or compressed length of the spring, and r is the rest length of the spring.

Figura 11. *Fórmula de Hooke*.

La suspensión también incluye un amortiguador capaz de frenar el efecto rebote del muelle para evitar su actuación de manera infinita, similar al comportamiento de los vehículos de la vida real.

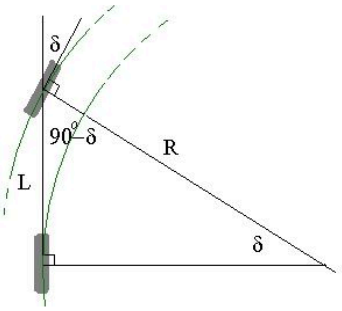
Existen otros sistemas y parámetros, como el ángulo de inclinación de las ruedas respecto a la carretera, con la capacidad de ir ajustándose dinámicamente según la aceleración y frenado. Se puede definir un *ángulo de caída* máximo y calcular el valor actual en función de la transferencia de peso longitudinal del vehículo para desarrollar este sistema.

De la misma manera que en el primer subapartado, sistemas como este último podrían ser omisibles si no se consideran estrictamente necesarios para la evolución del presente proyecto.

2.1.4.3 Dirección: llantas y neumáticos

El sistema de dirección simulado más conocido en términos de realismo es la geometría de dirección de Ackermann. Es un sistema de dirección utilizado principalmente en vehículos para asegurar que las ruedas delanteras sigan trayectorias curvas con radios diferentes, evitando el deslizamiento y mejorando la estabilidad en las curvas.

Consiste en colocar las ruedas delanteras de forma que, al girar, la rueda interior (más cercana al centro de la curva) gire con un ángulo mayor que la rueda exterior. Esto es deseable debido a que, al tomar una curva, la rueda interior recorre un radio de giro más pequeño que la exterior.



The distance between front and rear axle is known at the wheel base and denoted as L. The radius of the circle that the car describes (to be precise the circle that the front wheel describes) is called R. The diagram shows a triangle with one vertex in the circle centre and one at the centre of each wheel. The angle at the rear wheel is 90 degrees per definition. The angle at the front wheel is 90 degrees minus *delta*. This means the angle at the circle centre is also *delta* (the sum of the angles of a triangle is always 180 degrees). The sine of this angle is the wheel base divided by the circle radius, therefore:

$$\sin(\delta) = \frac{L}{R} \Leftrightarrow R = \frac{L}{\sin(\delta)}$$

Note that if the steering angle is zero, then the circle radius is infinite, i.e. we're driving in a straight line.

Figura 12. *Diagrama y fórmula de Ackermann aplicada al ángulo de giro*.

La implementación del sistema, en la vida real, se logra a través de una barra de dirección que conecta ambas ruedas delanteras, junto con un conjunto de brazos y pivotes que ajustan el ángulo de cada rueda según la posición del volante (Burnhill, 2009). Esta disposición permite que las ruedas mantengan una alineación óptima durante el giro, mejorando la tracción, la eficiencia y reduciendo el desgaste de los neumáticos.

La fricción de los neumáticos crea subviraje y sobreviraje, fenómenos clave en la conducción de vehículos en videojuegos.

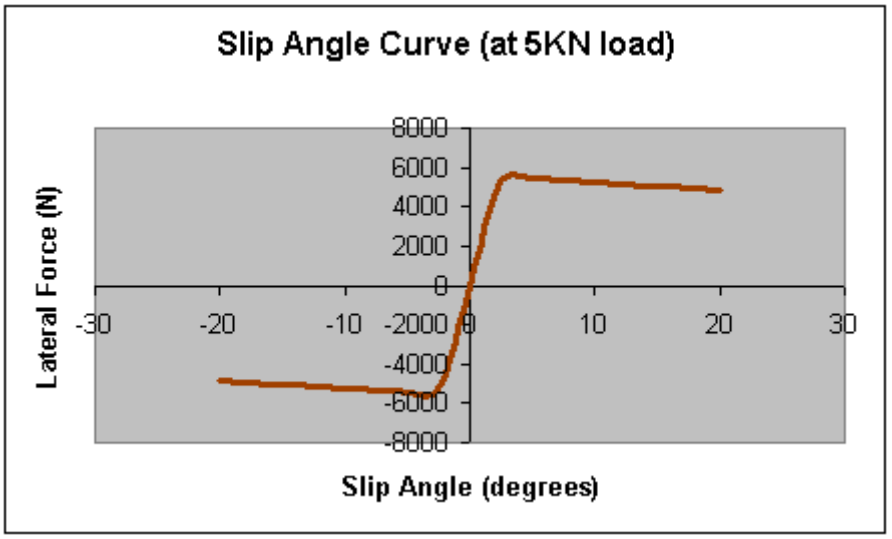


Figura 13. *Curva de deslizamiento de los neumáticos de Pacejka*.

Para calcular la respuesta del vehículo en giros a alta velocidad, se pueden utilizar modelos físicos como la *Fórmula Mágica de Pacejka*, que describe la relación entre el ángulo de deslizamiento y la fuerza lateral del neumático con alta precisión (López et al., 2012).

2.1.5 Videojuegos en red

Este es un breve extracto de la historia de los videojuegos en red basado en el libro "*Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*" (Armitage et al., 2006).

Figura 14. *QuakeWorld*. Fuente: Alchetron.

La evolución del networking en juegos tuvo sus comienzos con "*QuakeWorld*", que fue el primer juego documentado que introdujo la predicción de posicionamiento de los jugadores en el cliente para compensar la latencia, permitiendo partidas más fluidas en equipos con conexiones de distintas calidades y velocidades. Ello ayudó a que este mismo juego impulsara el auge del multijugador competitivo y los torneos en línea.

Figura 15. *Dune II*. Fuente: IMDB.

Los MMORPGs surgieron con "*Ultima Online*" (1995) inmediatamente seguido por "*EverQuest*" (1999999), ambos popularizaron el género con más medio millón de jugadores. Estos juegos introdujeron modelos de suscripción y mundos persistentes.

En el género RTS, "*Dune II*" (1992) estableció las bases, pero no fue hasta "*Warcraft*" (1994) que se llevó las mecánicas multijugador en línea al siguiente nivel. *Blizzard* perfeccionó esto con una secuela del mismo y, más tarde, con la integración de su plataforma online "*Battle.net*" con su servicio de emparejamiento en línea, optimizando la escalabilidad del juego en red.

Como ejemplo último, en el género de carreras, títulos como "*Need for Speed*" introdujeron sistemas de progresión y juego en red con perfiles persistentes, permitiendo competir y mejorar vehículos a lo largo del tiempo.

Figura 16. *Need for Speed Road Challenge*. Fuente: YouTube.

Valve publicó en su *wiki* el artículo "*Server In-game Protocol Design and Optimization*" (Bernier, 2001), donde se explican las técnicas utilizadas en el diseño de juegos multijugador, como la compensación de la latencia y la optimización de la experiencia de juego.

Aunque el artículo se enfoca principalmente en el desarrollo de juegos de acción en primera persona, con especial atención a "*Counter-Strike*", uno de los títulos más importantes de la compañía, las técnicas descritas pueden aplicarse a otros tipos de proyectos, incluido el actual.

2.1.5.1 Arquitectura cliente-servidor (server-client architecture)

Un gran porcentaje de los juegos en línea utilizan una arquitectura cliente-servidor. En este modelo, el servidor es la autoridad central que ejecuta la lógica principal del juego y lo controla, mientras que los clientes (generalmente, los jugadores) envían sus entradas (movimientos, disparos, etc.) a este.

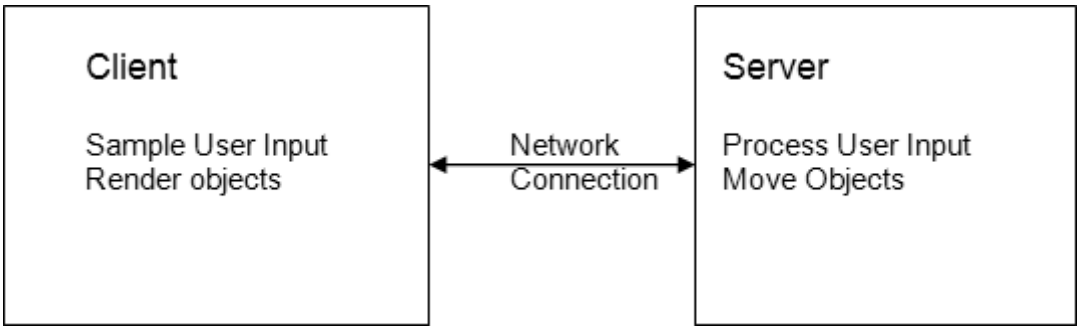


Figura 17. *Diagrama simplificado de una arquitectura cliente-servidor*. Fuente: Valve.

El servidor procesa la información percibida por parte de los clientes (las entradas), actualiza el estado del juego y envía la información actualizada de vuelta a los clientes para que actúen en consecuencia.

Este enfoque garantiza que el estado del juego sea relativamente consistente para todos los jugadores, pero introduce un retraso o "latencia" entre cada acción del jugador y la respuesta por parte del servidor.

Por poner un ejemplo, si un jugador tiene una latencia alta (por ejemplo, de 500 milisegundos), cualquier acción que realice tardará como mínimo ese medio segundo en ser enviada, procesada por el servidor, respondida de nuevo al cliente, procesada por este último, y finalmente ser reflejada en su pantalla.

2.1.5.2 Predicción del lado del cliente (client-side prediction)

Para mitigar el efecto de la latencia, se utiliza la técnica de la predicción del lado del cliente. Esta técnica permite que el cliente simule localmente el resultado de las acciones del jugador antes de recibir la confirmación del servidor.

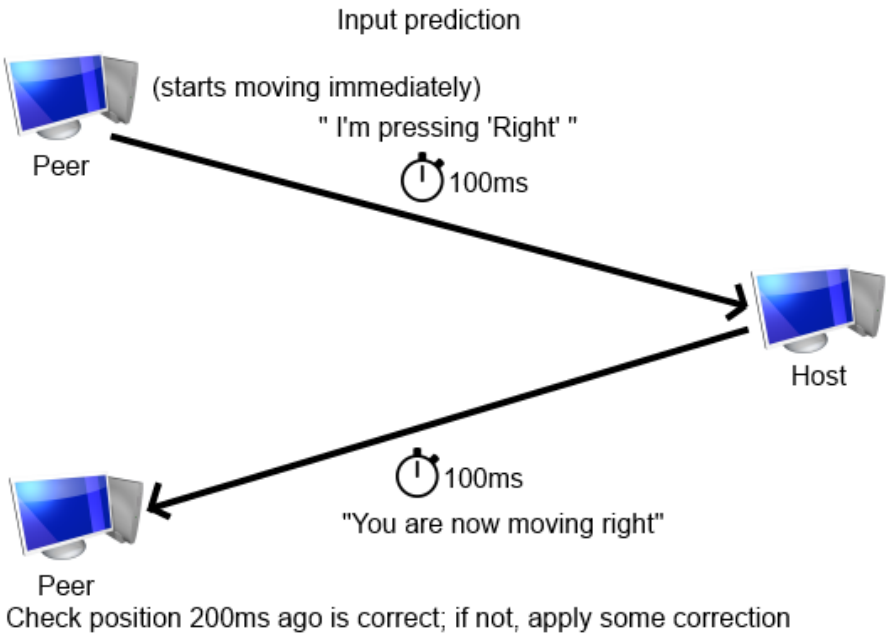


Figura 18. *Diagrama simplificado de ejemplo de latencia, predicción y corrección*. Fuente: Construct.net.

Por ejemplo, si un jugador se mueve hacia adelante, el cliente asume que el servidor aceptará ese movimiento y actualiza la posición del jugador de inmediato en la pantalla. Cuando el servidor finalmente responde, el cliente ajusta la posición si es necesario (este reajuste se conoce también como "corrección" o "compensación").

La principal ventaja es que el juego se siente más responsivo, ya que el jugador ve el resultado de sus acciones de inmediato. Por otro lado, si la predicción del cliente no coincide con la del servidor, se pueden producir correcciones bruscas (conocidas como "saltos" o "*warps*") en la posición de los jugadores, lo que puede ser incómodo, frustrante y perjudicial para la inmersión.

En juegos como "*Half-Life*", se comparte el mismo código de movimiento entre el cliente y el servidor para minimizar las discrepancias en la predicción. Esto asegura que tanto el cliente como el servidor simulen los movimientos de la misma manera.

2.1.5.3 Compensación de Latencia (Lag Compensation)

La compensación de latencia es una técnica que permite al servidor tener en cuenta la latencia de cada jugador al procesar sus respectivas acciones. Cuando un jugador realiza una acción, el servidor "retrocede en el tiempo" al momento en que el jugador realizó la acción y verifica si fue efectivo en ese preciso momento.

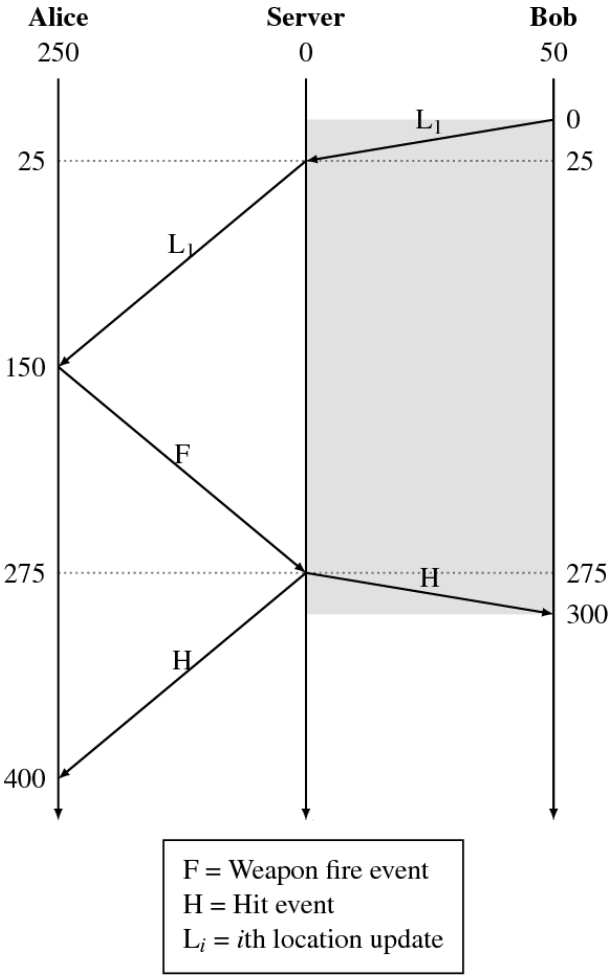


Figura 19. *Diagrama de secuencia de la transmisión de una acción en red*. Fuente: Semantic Scholar.

Esto significa que, aunque otro jugador se haya movido desde entonces, la acción se calculará en función de su posición en el pasado (en el momento que fue realizado).

La principal ventaja es que los jugadores no necesitan "adelantar" sus acciones para compensar la latencia. La desventaja es que la diferencia de latencias puede provocar falsos positivos en situaciones más extremas (al borde de colisiones).

2.1.5.4 Interpolación y Extrapolación

Unas técnicas no exclusivas (aunque sí recurrentes) de los juegos en red, son tanto la interpolación como la extrapolación de la posición de los objetos.

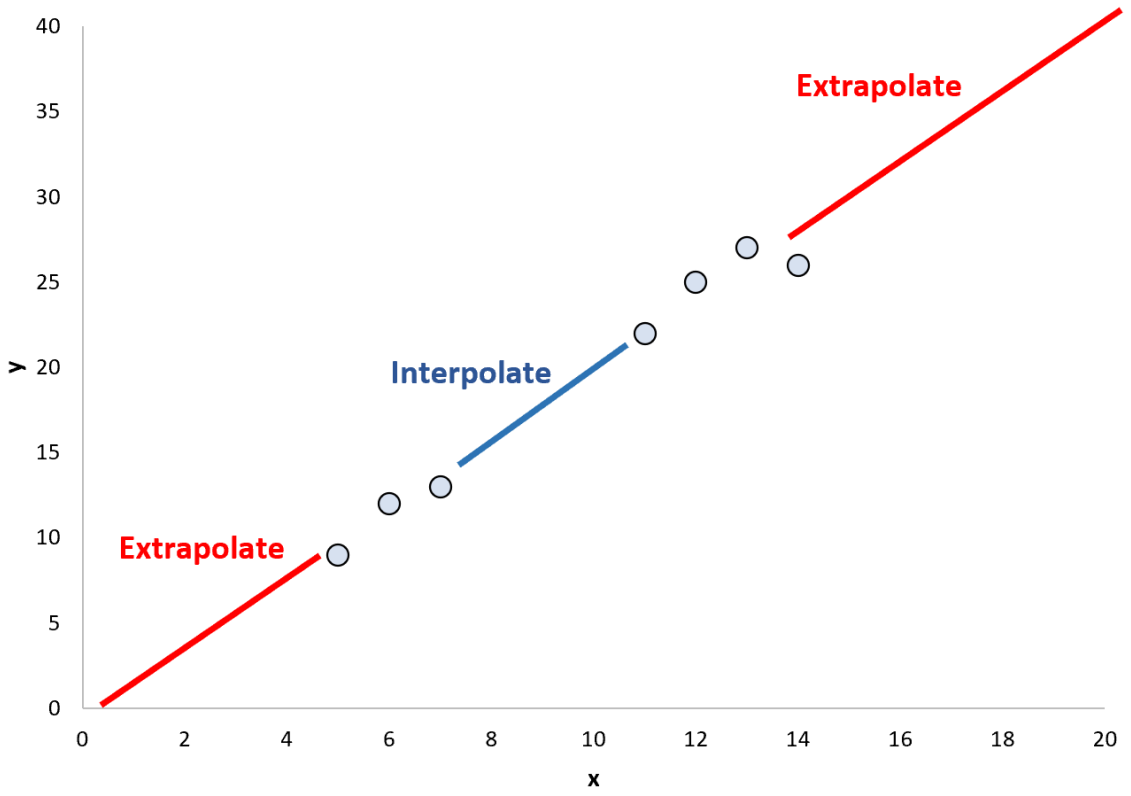


Figura 20. *Diferencia entre interpolación y extrapolación*. Fuente: Statology.

La interpolación es el suavizado de movimiento entre los objetos entre las posiciones recibidas, describiendo visualmente una trayectoria suave, que como consecuencia introduce un pequeño retraso respecto a su posición lógica.

En contrapartida, la extrapolación es la predicción de la posición futura de un objeto basándose en su última velocidad y dirección conocidas. Como aspecto negativo, esta técnica es menos precisa, ya que los movimientos no siempre son predecibles y pueden darse casos de saltos visuales cuando se producen cambios bruscos en la velocidad.

2.1.5.5 Nota de consideraciones en el diseño del juego

No todas las técnicas son necesarias para todos los proyectos, es necesario entender tanto el funcionamiento de estas en profundidad como el diseño del juego a desarrollar para saber elegir los elementos más convenientes según cada caso.

2.1.6 Videojuegos con físicas en red

2.1.6.1 Sincronización de estado en física de red

La sincronización del estado de un objeto es una estrategia para la implementación de física en red basado en la ejecución de la misma simulación en ambos extremos de la conexión y enviar tanto el estado como las entradas de los mismos.

A diferencia de otras técnicas como el *Deterministic Lockstep* (este método se detalla en una sección dedicada más adelante) donde solo se envían las entradas y se espera que ambas simulaciones produzcan los mismos resultados, en la sincronización de estado se envían actualizaciones periódicas de posición, orientación, velocidad lineal y velocidad angular (Fiedler, 2005).

Esto permite que la simulación sea resiliente ante pérdidas de paquetes, aunque introduce potenciales problemas como las divergencias y saltos en los objetos debido a extrapolaciones incorrectas.

2.1.6.2 Optimización del envío de estados

Es recomendable tomar ciertas medidas para optimizar (generalmente reducir) el uso de ancho de banda y mejorar los tiempos de respuesta del juego (Fiedler, 2010):

- No enviar actualizaciones de estado para todos los objetos en cada paquete, sino solo para un subconjunto seleccionado estratégicamente. Para ello es conveniente crear un esquema de priorización de objetos de red, anteponiendo los controles de los jugadores por encima de otros objetos del escenario o menos importantes.
- Evitar transmitir datos innecesarios, como velocidades nulas cuando los objetos están en reposo. Además, los paquetes incluyen un número de secuencia para identificar y evitar enviar pérdidas o duplicados de los datos.
- Enviar múltiples entradas redundantes para reducir el impacto de la pérdida de paquetes y favorecer su compensación.

2.1.6.3 Control de ancho de banda y adaptabilidad

El ancho de banda disponible no siempre es constante, por lo que es importante poder ajustarlo dinámicamente.

Para ello, se analiza la congestión de la red y se reduce la tasa de actualización y la calidad de la sincronización de manera controlada. Cuando la red mejora, se vuelve a aumentar la cantidad de datos enviados.

2.1.6.4 Manejo del jitter en la recepción

Debido a la naturaleza de las redes, los paquetes no siempre llegan en el orden correcto ni a intervalos regulares.

Para evitar efectos visuales abruptos como consecuencia de ello, se utiliza un “buffer del jitter”, encargado de almacenar la información de los paquetes temporalmente antes de aplicarlos a la simulación en el momento correcto. De esta manera, se garantiza que las actualizaciones de estado lleguen de manera uniforme y sin desincronización entre los objetos (Fiedler, 2015).

2.1.6.5 Suavizado visual para ocultar errores

Dado que los saltos en la simulación son inevitables debido a los condicionantes de la red, es muy habitual (y hasta recomendable) compensarlo mediante correcciones visuales.

En lugar de alterar la simulación o la lógica subyacente, es posible calcular una simulación de los desplazamientos en posición y orientación que se reduzca gradualmente en cada cuadro hasta llegar a cero para la compensación total.

La idea detrás de estos sistemas es permitir que los objetos en pantalla se muevan suavemente incluso cuando internamente la simulación experimenta cambios bruscos.

2.1.7 Anexo: Físicas deterministas en red (Deterministic lockstep)

Este método es una técnica empleada para la sincronización efectiva de simulaciones físicas en red mediante el envío exclusivo de las entradas del usuario en lugar del estado completo del sistema.

Esto reduce significativamente el uso del ancho de banda, ya que la cantidad de datos transmitidos depende del tamaño de la entrada y no del número de objetos en la simulación. Sin embargo, su implementación es bastante más compleja que el otro método, ya que requiere que la simulación sea completamente determinista, algo difícil de lograr debido a las variaciones en el comportamiento de los cálculos de operaciones de coma flotante entre diferentes motores de físicas, compiladores, sistemas operativos y arquitecturas de hardware.

Este anexo está basado en la documentación provista por Glenn Fiedler en el artículo "*Deterministic Lockstep*" (2014), posteriormente revisado en su conferencia "*Physics for Game Programmers: Networking for Physics Programmers*" (2015).

2.1.7.1 Determinismo en la simulación

El determinismo en el caso de la simulación significa que, dados tanto un estado inicial y unas entradas, la simulación debe producir exactamente los mismos resultados en cada ejecución, sin variaciones.

Cualquier pequeña diferencia en el orden de procesamiento, decimal por ínfimo que sea, o cambio de cualquier índole, puede generar divergencias progresivas entre simulaciones, lo que provoca su desincronización.

Los motores de física más empleados (Nvidia PhysX, Chaos, Bullet) no son deterministas en su salida, esto incluye el caso de Unity por defecto; no por ello existen otros motores (Havok, Jolt) que sí cuentan con esta característica.

Existen técnicas para garantizar cierto grado de determinismo en un simulador de físicas no determinista, pero se puede fijar la semilla del generador de números aleatorios antes de cada simulación.

NOTA: En el caso de Jolt, el determinismo es opcional mediante una configuración para una compilación específica (*Jolt Physics*, n.d.).

2.1.7.2 Envío de entradas en red

En una simulación basada en entradas, no es necesario enviar todo el estado del teclado o del periférico de entrada correspondiente, sino únicamente el de aquellos botones que afectan directamente a la simulación.

Para ello, las entradas se representan en una estructura de datos y se envían junto con un identificador del fotograma. La simulación remota solo puede avanzar cuando recibe las

entradas correspondientes a un fotograma específico. Si estos no llegaran a tiempo, la simulación debe esperar.

2.1.7.3 Buffer de retraso de reproducción (playout delay buffer)

Dado que los datos enviados a través de la red pueden llegar con fluctuaciones en el tiempo (*jitter*), es necesario usar la técnica del *Playout Delay Buffer*, un mecanismo que almacena temporalmente los paquetes para asegurar una entrega constante.

Este enfoque es similar a cómo los servicios de streaming, como YouTube o Netflix, almacenan una pequeña cantidad de video adelantada al tiempo de reproducción para evitar interrupciones a través de una descarga progresiva.

La principal contra de este sistema es la introducción de una pequeña latencia en la respuesta por parte del propio juego, pero ayuda a suavizar la llegada de paquetes. Para funcionar correctamente y asegurar una experiencia de juego agradable debe elegirse adecuadamente: Un tamaño de *buffer* demasiado pequeño puede causar interrupciones en la simulación si los paquetes se retrasaran, mientras que un *buffer* excesivamente grande aumenta la latencia total del sistema.

2.1.7.4 Limitaciones del TCP

El protocolo TCP garantiza la entrega de datos en orden y sin pérdida pero introduce problemas en redes con alta latencia y alto índice de pérdida de paquetes.

En escenarios con frecuente pérdida de paquetes, TCP debe retransmitir los datos y esperar su recepción, lo que genera pausas en la simulación y efectos visuales como saltos en los objetos de red. Esto se intensifica todavía más si la red tiene una latencia media o alta.

2.1.7.5 Alternativa con UDP

La propuesta para evitar los problemas del sistema basado en TCP, es el uso de UDP con un protocolo personalizado para el proyecto: En lugar de retransmitir paquetes perdidos, se pueden enviar redundancias de entradas con cada paquete, asegurando que todas las entradas lleguen eventualmente sin necesidad de esperar confirmaciones.

Dado que las entradas de usuario son pequeñas en tamaño, es viable enviar varias actualizaciones de entradas en cada paquete sin consumir un mayor ancho de banda.

Bajo este enfoque, el receptor envía confirmaciones (en redes, *acks*) de las entradas recibidas, permitiendo al emisor descartar aquellas que ya han sido confirmadas por el receptor. Esto mantiene el número de entradas en tránsito bajo control, minimizando aún más el consumo de red.

2.2 Referentes

2.2.1 Rocket League

Figura 21. *Rocket League*. Fuente propia.

Rocket League es un videojuego de deporte desarrollado y publicado por *Psyonix* en 2015 para PC y *PlayStation 4*.

Combina el fútbol con vehículos impulsados por cohetes, donde los jugadores controlan coches para golpear un balón gigante e intentar anotar goles en la portería del equipo contrario. El juego se caracteriza por ser fácil de controlar y frenético. Cuenta con modos de juego tanto para un jugador como multijugador en línea.

Jared Cone, ingeniero principal de gameplay en Psyonix, ofreció una visión detallada sobre los aspectos técnicos detrás de Rocket League en su charla "*It IS rocket science! The Physics of Rocket League detailed*" en la *Game Developers Conference* de 2018. Cone aborda tres áreas principales: el motor de física, el sistema de vehículos y la red.

Motor de física

El videojuego utiliza el motor de físicas *Bullet*, una elección que considera fundamental para el éxito técnico del juego debido a su filosofía de código abierto, lo que permitió al equipo modificar y ajustar el motor para adaptarlo a las necesidades específicas del juego.

Aunque Rocket League se desarrolló en Unreal Engine 3, el equipo decidió no reemplazar el motor de física nativo, sino integrar *Bullet* como un motor adicional para manejar la física del gameplay.

Una de las decisiones clave fue utilizar una tasa de actualización de las físicas o *tickrate* fijo de 120 Hz, lo que significa que el juego actualiza los cálculos de la física 120 veces por segundo. Esto garantiza una simulación física suficientemente determinista, lo que es crucial tanto para la jugabilidad como para la red.

Como apunte, Cone admite que, en retrospectiva, habría explorado otras soluciones para manejar las penetraciones en colisiones, ya que el alto *tickrate* aumenta directamente el costo computacional del juego.

Sistema de vehículos

El diseño de los vehículos en Rocket League tenía objetivos claros: debían ser rápidos, responsivos y estables. El juego no busca ser realista, pero la física debe ser consistente y controlable para que los jugadores sientan que tienen un control preciso sobre sus coches.

Figura 22. *Diferencia entre las fuerzas ejercidas sobre una rueda de vehículo*.
Fuente: Inges Aerospace-Mechanical Engineering.

Para simplificar el sistema de vehículos, se tomaron las siguientes decisiones sobre su diseño:

- Eliminación de la transmisión: En lugar de simular cambios de marcha, se proporcionó a los diseñadores una curva de aceleración que podían ajustar directamente.
- Masa constante: Se desactivó el cálculo automático de la masa basado en la densidad y el tamaño, lo que permitió mantener la masa constante y evitar cambios inesperados en el manejo.

- Fricción simplificada: Se eliminó la fricción longitudinal (la que hace que las ruedas giren) y se simplificó la fricción lateral para lograr un equilibrio entre un manejo preciso y la capacidad de deslizamiento.

Adicionalmente, Cone destaca la importancia de aplicar las fuerzas de fricción a la altura del centro de masa para evitar que los coches vuelquen al girar bruscamente. Esto se logra mediante la aplicación de fuerzas auxiliares que ayudan a estabilizar los vehículos cuando pierden el control.

Red y predicción de entradas

Uno de los mayores desafíos técnicos de Rocket League es hacer que el juego sea competitivo en línea, ya que el juego requiere una sincronización precisa entre los clientes y el servidor para manejar colisiones con objetos en movimiento, como la pelota.

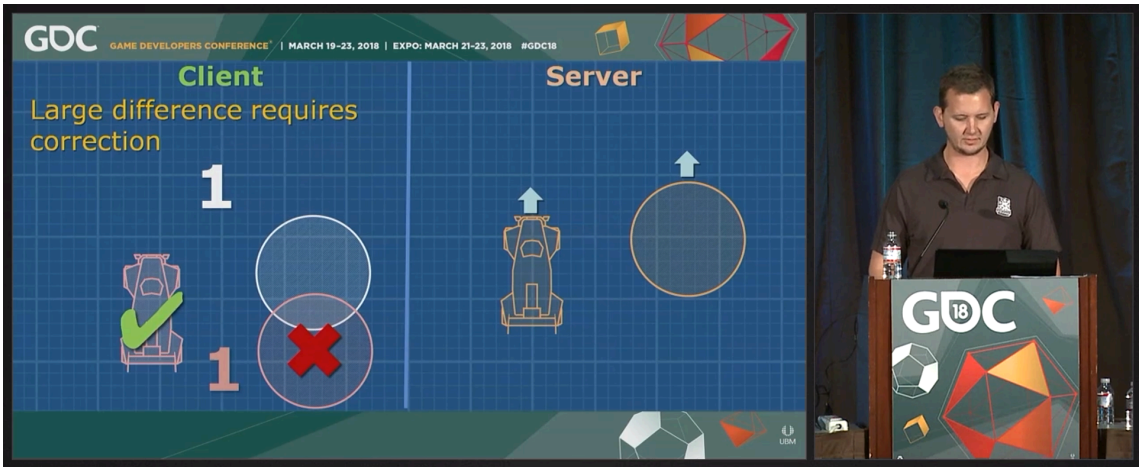


Figura 23. *Jared Cone exponiendo el sistema de red de Rocket League*. Fuente: GDC.

El sistema de red de Rocket League se basa en dos conceptos principales:

- *Buffer* de entrada: En lugar de procesar las entradas de los jugadores tan pronto como llegan al servidor, estas se almacenan en un *buffer* y se consumen a un ritmo constante.
- Predicción completa: Los clientes predicen el movimiento de su propio vehículo, el de los demás jugadores y el del balón. Cuando el servidor envía una actualización, el cliente compara su predicción con la información del servidor y, si es necesario, retrocede en el tiempo y vuelve a simular las físicas para corregir las discrepancias.

Esto permite que los jugadores con alta latencia no arruinen la experiencia para los demás, ya que no se utiliza compensación de lag (*lag compensation*). Como contrapartida, las correcciones pueden ser costosas en términos de rendimiento, como agregado al coste ya mencionado.

2.2.2 Wipeout Pulse

Figura 24. *Wipeout Pulse*. Fuente: TodoJuegos.com.

Wipeout Pulse es un videojuego de carreras de naves futuristas desarrollado por *Studio Liverpool* y publicado por *Sony Computer Entertainment* en 2007 para la *PlayStation Portable*.

Como parte de la franquicia *Wipeout*, el juego mantiene el enfoque de sus anteriores entregas en la velocidad extrema y las naves antigravedad, además de introducir nuevas mecánicas, potenciadores, circuitos y banda sonora.

Este juego ha sido desglosado a partir del análisis "*WIPEOUT PULSE - Last, best & final form*" (The Review Den, 2023) para tratar de abarcar sus aspectos más importantes.

Contenido

Teniendo en cuenta que el juego salió para la PlayStation Portable, una videoconsola por lo general poco potente, cabe destacar una mejora en la calidad visual respecto a sus antecesores (ejemplo en la Figura 24): un mayor nivel de detalle, efectos visuales, iluminación, interfaz dinámica, y todo ello ejecutado a una tasa de refresco de pantalla de 60 Hz sin alteraciones por parte del usuario.

También como continuación de la estela de la marca, se retomaron algunos de los elementos más icónicos conocidos para los seguidores de la saga: 8 escuderías con hasta 3 naves cada una, 12 pistas recorribles en ambos sentidos (en el paquete de expansión añaden alguna más, pero solo estuvo disponible por tiempo limitado en territorio europeo), una gran cantidad de música renovada, y todo un nuevo modo campaña organizado en cuadrícula para dar mayor flexibilidad al jugador sobre cómo progresar en las fases.

Respecto al apartado sonoro, hubo ciertas mejoras por la parte de los sonidos: no solo se mantuvieron o mejoraron los sonidos del antecesor juego, también se agregaron efectos de retroalimentación para ciertas acciones como los aceleradores, para una mejor indicación; por

el lado de la música, la inclusión de artistas archiconocidos en el género de la electrónica como Aphex Twin y Kraftwerk fue grandemente aclamada, además de permitir personalizar la música a escuchar durante el juego directamente desde la carpeta de canciones con el contenido que el usuario tenga almacenada en la propia consola.

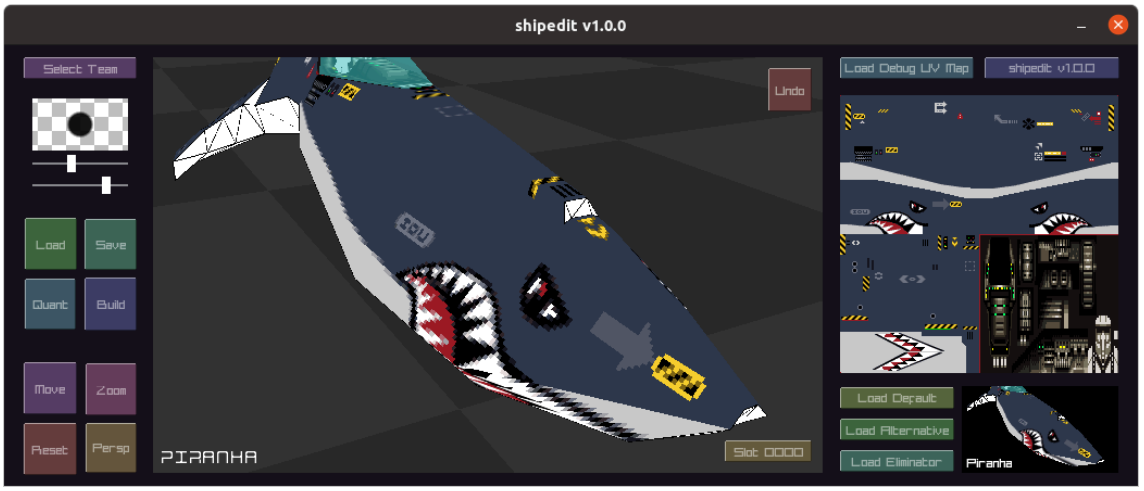


Figura 25. *Shipedit*. Fuente: thp.

Algo único en este videojuego fue la posibilidad de personalizar la apariencia o *skin* de las naves a través de software externo: durante los primeros años, el propio estudio propició una página web oficial donde la gente podía crear sus naves personalizadas directamente desde sus navegadores web; más tarde, tras la caída de esta web y el abandono del juego, la comunidad creó sus propios programas (como Shipedit, en la Figura 25) para continuar aprovechando esta característica.

Jugabilidad

En el análisis se destaca un comportamiento equiparable una mezcla equilibrada entre “Wipeout Fusion” (2002) y su manejo “sobre raíles”, y “Wipeout Pure” (2005) y su nuevo sistema de físicas más complejas de manejar.

Esto sumado a una nueva tirada de circuitos mucho mejor diseñados para aprovechar el sistema antigravitatorio, ofreciendo mayor variedad de zonas intrincadas y bucles, también recibió elogios por ser excepcionalmente divertido de jugar.

2.2.3 Trailmakers

Figura 26. *Trailmakers*. Fuente propia.

Trailmakers es un videojuego de construcción y exploración desarrollado y publicado por *Flashbulb Games* en 2019 para PC, *PlayStation 4* y *Xbox One*.

El juego permite a los jugadores diseñar y ensamblar vehículos modulares utilizando un sistema de construcción basado en piezas similares al sistema LEGO, y cuyos diseños van desde simples autos y aviones hasta complejas máquinas personalizadas (Flashbulb Games, 2020).

Construcciones y física

El juego se presenta en su anuncio oficial como una experiencia accesible para exploradores, corredores e inventores, siendo la mecánica principal la construcción de vehículos, que se realiza de manera guiada e intuitiva mediante un sistema de arrastrar y soltar piezas. Esto permite a los jugadores crear vehículos funcionales de forma rápida y sencilla.

El juego pretende basarse en física realista, lo que significa que los vehículos se comportan de manera creíble según su diseño. Esto añade un nivel de profundidad al gameplay, ya que los jugadores deben considerar aspectos como el peso, la aerodinámica y la estabilidad al construir sus creaciones.

A través de las contínuas actualizaciones el juego ha ido contando con un abanico de piezas con diferentes funcionalidades: articulaciones, armas, pesos, etc. que elevan todavía más la experiencia.

Modos de juego

El juego cuenta con varios modos de caja de arena o *sandbox* para que los jugadores puedan disfrutar de sus construcciones sin restricciones, en amplios y variados mundos llenos de caminos, obstáculos, agua o vegetación, entre otras muchas atracciones.

Sin embargo, otros modos muy interesantes son las campañas: escenarios diseñados para que los jugadores vayan avanzando y viendo los diferentes bloques de manera progresiva, enseñándole a usarlos para lograr una serie de objetivos definidos en forma de misiones.

Adicionalmente, es posible competir en diversos modos de carrera y contrarreloj, también apoyados por el componente crucial que es la construcción de vehículos.

Multijugador

Trailmakers soporta hasta 8 jugadores en la mayoría de sus modos de juego, lo que permite:

- Cooperar en cualquiera de las campañas principales y compartir el progreso entre los jugadores.
- Participar en torneos de robots de batalla en las arenas dedicadas.
- Competir en las diversas carreras disponibles en las zonas marcadas.
- Jugar libremente en los mapas *sandbox* para crear sus propias aventuras, competiciones, pruebas, etc.

Comunidad

El juego se integra con la *Workshop* de *Steam*, donde los jugadores pueden subir sus creaciones para compartir con otros jugadores de la comunidad. Esto además fomenta la creatividad y permite enseñar a otros jugadores menos diestros las mecánicas que esconde el juego.

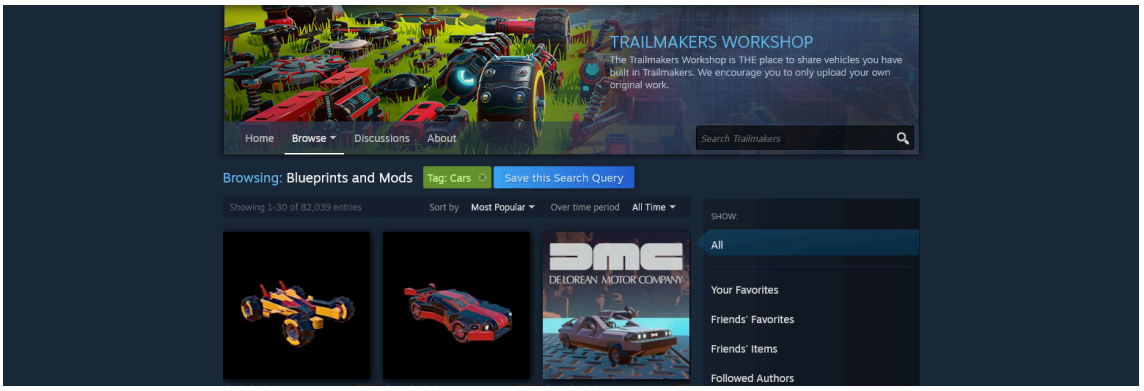


Figura 27. *Workshop de Trailmakers*. Fuente: Steam.

Además, los desarrolladores organizan desafíos semanales o *Weekly Challenges*, competiciones de construcción en las cuales los mejores diseños se muestran en transmisiones en vivo. Los constructores ganadores reciben un bloque especial de trofeo como recompensa como reconocimiento.

2.2.4 TrackMania Nations Forever

Figura 28. *TrackMania Nations Forever*. Fuente: YouTube.

TrackMania Nations Forever es un videojuego de carreras para PC desarrollado por *Nadeo* y publicado por *Ubisoft* y lanzado en 2008 como una versión mejorada y gratuita de TrackMania Nations, la anterior entrega de la saga.

Se centra en carreras a alta velocidad en circuitos desafiantes que juegan con las acrobacias y la gravedad, y con un sistema de conducción determinista basado en alta precisión y control (ColourShedProductions, 2016).

Contenido

El juego cuenta con un modo en solitario que presenta varias series de numerosos desafíos, estructurados en etapas donde los jugadores deben superar tiempos específicos para desbloquear nuevas pistas y obtener medallas de bronce, plata y oro según sus tiempos contrarreloj. Para acceder a las etapas finales, se requiere que el jugador alcance las medallas de oro en todas las pistas anteriores.

El diseño de los niveles no solo se enfoca en la funcionalidad, ya de por sí compleja y bien medida, sino que también incorpora ciertos elementos cosméticos como globos, rampas y distintos tipos de terreno, como hierba, tierra o incluso secciones bajo el agua, lo que aporta variedad visual y reduce la sensación de repetición, un hito en sí mismo dado que la práctica totalidad del juegos sucede en un único escenario de estadio.

Uno de los aspectos más destacados es el editor de pistas, el cual brinda la posibilidad de diseñar y compartir circuitos personalizados, impulsando la creatividad entre la comunidad de aficionados.

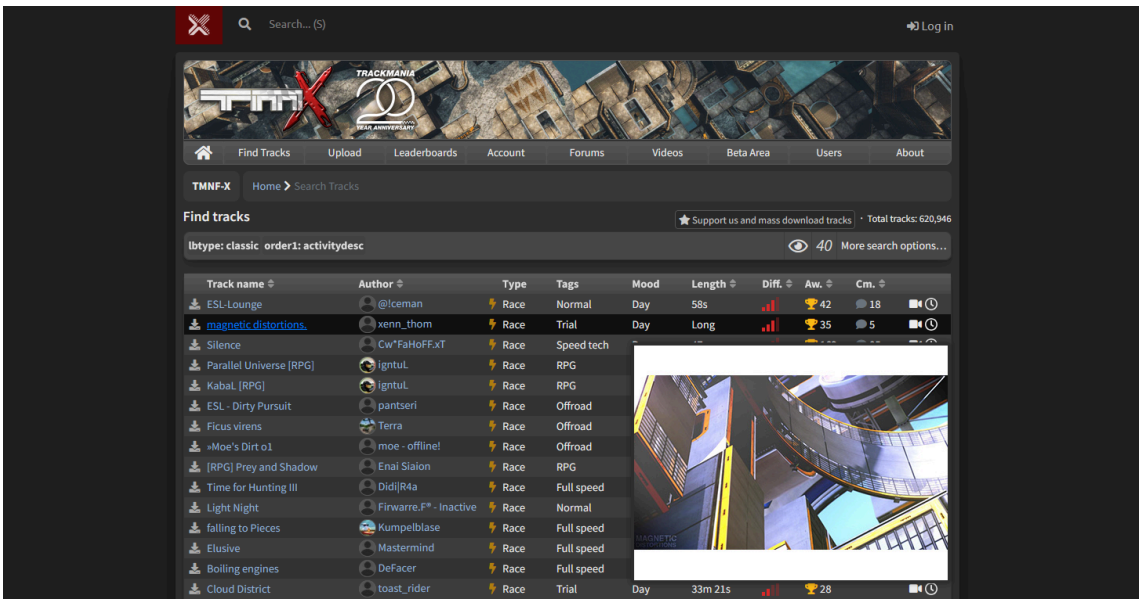


Figura 29. *TMNF-X (TrackMania Nations Forever Exchange)*. Fuente propia.

Plataformas web como *TrackMania Nations Forever Exchange* (Figura 29) es una pequeña muestra de lo volcada que ha sido capaz de estar la comunidad durante todos estos años, donde incluso a día de hoy se siguen subiendo y compartiendo circuitos.

Jugabilidad

Los gráficos del juego, a pesar de su reducido tamaño de almacenamiento (486.45 MiB de descarga y 740.96 MiB en disco según SteamDB), se caracterizan por su limpieza y fluidez manteniendo una tasa de 60 fotogramas por segundo en un ordenador de gama media-baja.

Se ha señalado la falta de variedad en los escenarios, ya que toda la acción transcurre en el mismo estadio sin gradas, y aunque se trate de un videojuego gratuito, su antecesor "*TrackMania United*" *ofrece* una mayor diversidad en este aspecto.

En cuanto a los controles, destacan por ser extremadamente sencillos e intuitivos, permitiendo incluso jugar con una sola mano. No obstante, y debido a la limitación en la mecánica de juego, los vehículos tienen cierta tendencia a volcarse, y el juego carece de un sistema oficial para restablecer su posición para obligar a reiniciar el circuito, lo que puede generar frustración entre algunos jugadores.

Multijugador online

El modo en línea se presenta como uno de los elementos más destacados del juego, siendo probablemente el factor más importante en su popularidad. Los servidores aleatorios funcionan como contrarrelojes en los que los jugadores de todo el mundo compiten por obtener el mejor tiempo. Adicionalmente hay rankings mundiales clasificados según el tamaño del territorio: comunidad, nacional, continental y mundial.

Capítulo 3. DESARROLLO DEL PROYECTO

3.1 Planificación del proyecto

A continuación se presenta un desglose de la planificación del proyecto así como del prototipo que lo acompaña, con una vista general del diagrama Gantt y la tabla de las tareas ejecutadas.

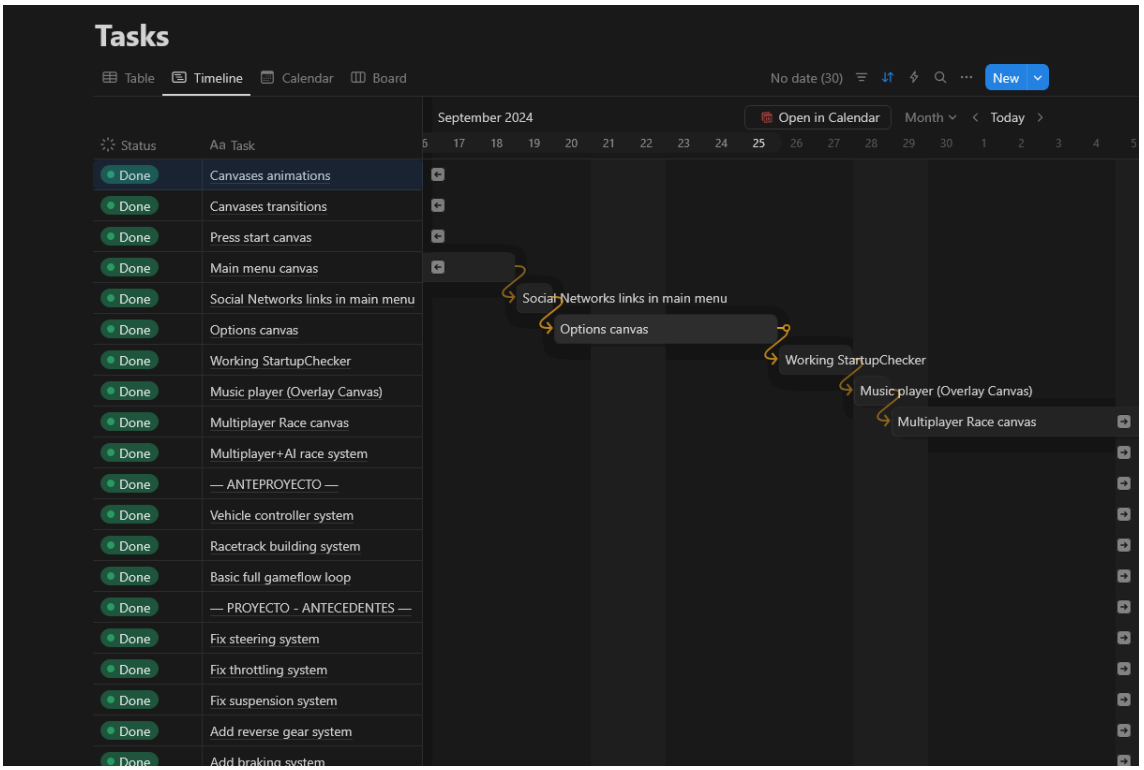


Figura 30. *Diagrama Gantt de tareas en Notion.*

Para la gestión y seguimiento del proyecto se ha utilizado la plataforma Notion mediante el uso de una base de datos tipo calendario y su correspondiente vista de diagrama de Gantt —esta es la vista más útil—, donde asignar las fechas, el orden de resolución y los estados a cada tarea.

Status	Task	Date
Done	Canvases animations	Sep. 2, 2024 → Sep. 3, 2024
Done	Canvases transitions	Sep. 4, 2024 → Sep. 5, 2024
Done	Press start canvas	Sep. 6, 2024 → Sep. 9, 2024
Done	Main menu canvas	Sep. 10, 2024 → Sep. 18, 2024
Done	Social Networks links in main menu	Sep. 19, 2024 → Sep. 19, 2024
Done	Options canvas	Sep. 20, 2024 → Sep. 25, 2024

Done	Working StartupChecker	Sep. 26, 2024 → Sep. 27, 2024
Done	Music player (Overlay Canvas)	Sep. 28, 2024 → Sep. 28, 2024
Done	Multiplayer Race canvas	Sep. 29, 2024 → Oct. 20, 2024
Done	Multiplayer+AI race system	Oct. 21, 2024 → Nov. 17, 2024
Done	— ANTEPROYECTO —	Nov. 18, 2024 → Nov. 24, 2024
Done	Vehicle controller system	Nov. 25, 2024 → Jan. 13, 2025
Done	Racetrack building system	Jan. 14, 2025 → Jan. 20, 2025
Done	Basic full gameflow loop	Jan. 21, 2025 → Feb. 16, 2025
Done	— ANTECEDENTES —	Feb. 17, 2025 → Mar. 9, 2025
Done	Fix steering system	Mar. 10, 2025 → Mar. 11, 2025
Done	Fix throttling system	Mar. 12, 2025 → Mar. 13, 2025
Done	Fix suspension system	Mar. 14, 2025 → Mar. 16, 2025
Done	Add reverse gear system	Mar. 17, 2025 → Mar. 17, 2025
Done	Add braking system	Mar. 18, 2025 → Mar. 20, 2025
Done	Fix and optimize networked ghost	Mar. 21, 2025 → Mar. 23, 2025
Done	Design Scene Document	Mar. 24, 2025 → Mar. 28, 2025
Done	Reimplement initial check system	Mar. 29, 2025 → Mar. 30, 2025
Done	Create 3D proxy vehicles	Mar. 31, 2025 → Apr. 1, 2025
Done	Create 3D proxy circuit	Apr. 2, 2025 → Apr. 4, 2025
Done	Customize GUI with basic assets	Apr. 5, 2025 → Apr. 11, 2025
Current	— PROYECTO Y MEMORIA —	Apr. 12, 2025 → Jun. 1, 2025

Tabla 1. *Desglose de tareas del proyecto y Trabajo de Fin de Grado.*

Esta tabla recoge de manera general todas las tareas realizadas a lo largo del proyecto y su prototipo. Cada entrada incluye el estado, el título de la tarea y el espacio de tiempo asignado a cada una.

3.2 Descripción de la solución, metodologías y herramientas empleadas

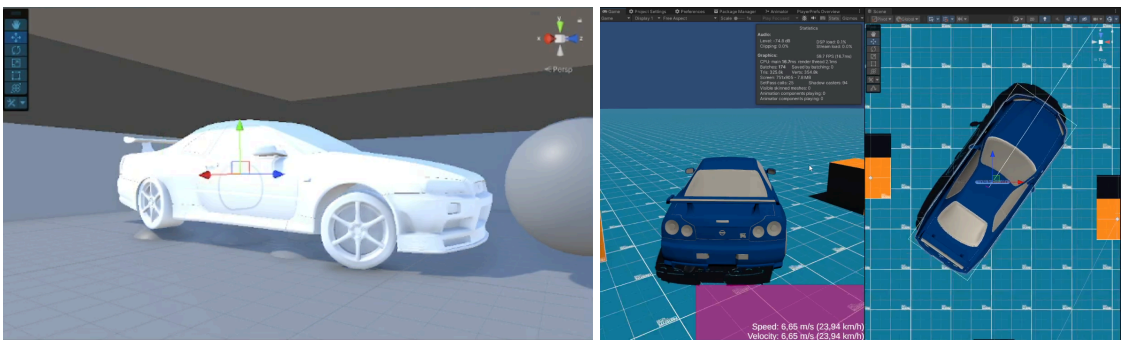
3.2.1 Contexto del proyecto

Figura 31. *Cartel promocional de Protovolkers en 2024.*

Si bien el Trabajo de Fin de Grado se centra específicamente en el estudio y desarrollo de un sistema de físicas en red, su origen está estrechamente ligado al proyecto ***Protovolkers***, comenzado en 2022.

Fue a raíz de la evolución de este proyecto personal, junto con la adición por parte de Unity de actualizar su motor y herramientas de desarrollo en red, así como la formación adquirida durante el curso, cuando surgió la oportunidad de enfocar el trabajo en la implementación de un sistema multijugador basado en físicas sincronizadas entre los clientes.

Este proyecto nació como una iniciativa personal de desarrollo de un videojuego de carreras de ambientación futurista, tomando como principales referentes títulos como Wipeout Pulse, Trackmania Nations Forever o Trailmakers, entre otros. En su planteamiento inicial, la intención era crear una experiencia de conducción para un único jugador.



Figuras 32 y 33. *Capturas del desarrollo de Protovolkers en 2022 —izquierda— y 2024 —derecha—.*

Con el paso del tiempo, y en paralelo al avance de las tecnologías implementadas por Unity en su inminente versión 6, así como la actualización de su paquete Netcode for GameObjects a su versión 2, y el aprendizaje adquirido en clases de redes y programación multijugador, abrieron la posibilidad de contemplar una sustancial mejora para el proyecto con la adición de un componente multijugador en línea en tiempo real.

Esta decisión implicó redefinir varios aspectos del diseño inicial, ya que la incorporación del componente de red, según su implementación, no solo afecta a la arquitectura del código, sino también al planteamiento de la jugabilidad, el flujo de la partida, y **la sincronización de físicas**.

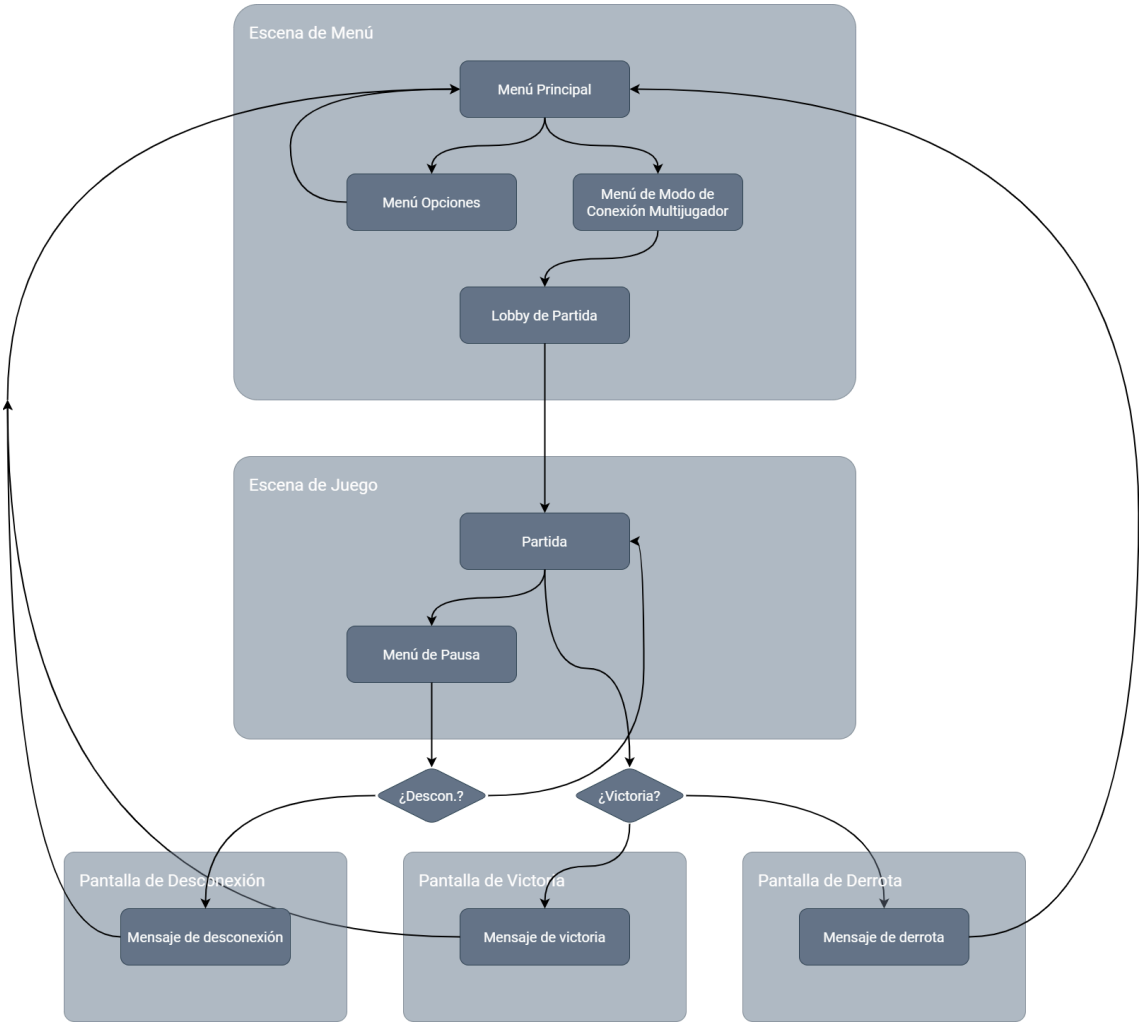


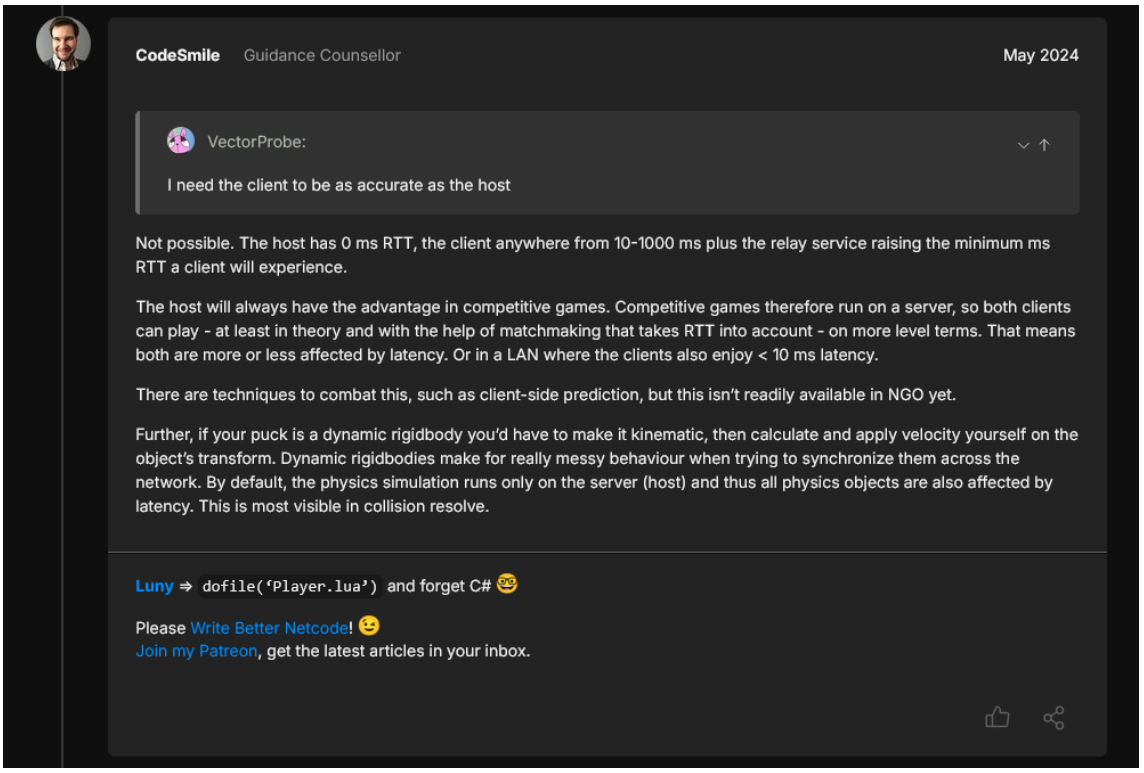
Figura 34. *Diagrama de la primera adaptación del flujo de escenas a un juego en red de Protovolkers.*

Naturalmente, es posible lograr una integración de mecánicas de red de maneras mucho metros intrusivas: sistemas de mensajería en tiempo real o presentes en el mundo (*Dark Souls*, etc.), tablas de clasificación mundial (*Muse Dash*, etc.), o incluso omitir la transmisión de físicas para funcionar a modo de vehículo fantasma (*Dirt 3*, etc.). Sin embargo, un sistema como este es de mucho más interés, y supone un reto técnico a la altura de un Trabajo de Fin de Grado.

3.2.2 Problemas presentes

El objetivo principal del proyecto es diseñar un sistema de multijugador en línea que permita a varios jugadores controlar sus personajes —o vehículos en el caso de *Protovolkers*— a través de un sistema de físicas pseudorrealistas en red.

Sin embargo, este objetivo presenta varios problemas técnicos, que se detallan a continuación.



Figuras 35 y 36. *Steffen Itterheim [CodeSmile] explicando el problema de la transmisión de físicas en red.* Fuente: Unity Discussions.

3.2.2.1 Falta de una solución oficial integrada

Unity, al igual que los principales motores de videojuegos actuales, no proporciona una solución oficial completa para la sincronización de físicas en red dentro de sus paquetes estándar. Aunque existen alternativas desarrolladas por terceros o paquetes experimentales, carecen de un soporte robusto y la integración que ofrece el motor en otras áreas.

3.2.2.2 Limitaciones en la sincronización de *rigidbodies* en red

La opción más próxima al objetivo dentro de Unity, siendo esta el uso del componente NetworkRigidbody, permite sincronizar objetos físicos mediante *rigidbodies*, pero presenta una limitación importante: la falta de transmisión de las colisiones.

Dicho de otra forma, los vehículos de los demás jugadores aparecen como objetos cinemáticos (*kinematic*), lo que condiciona al sistema de físicas a interpretar estos objetos como **estáticos a efectos prácticos**, lo que impide una interacción física real entre ellos.

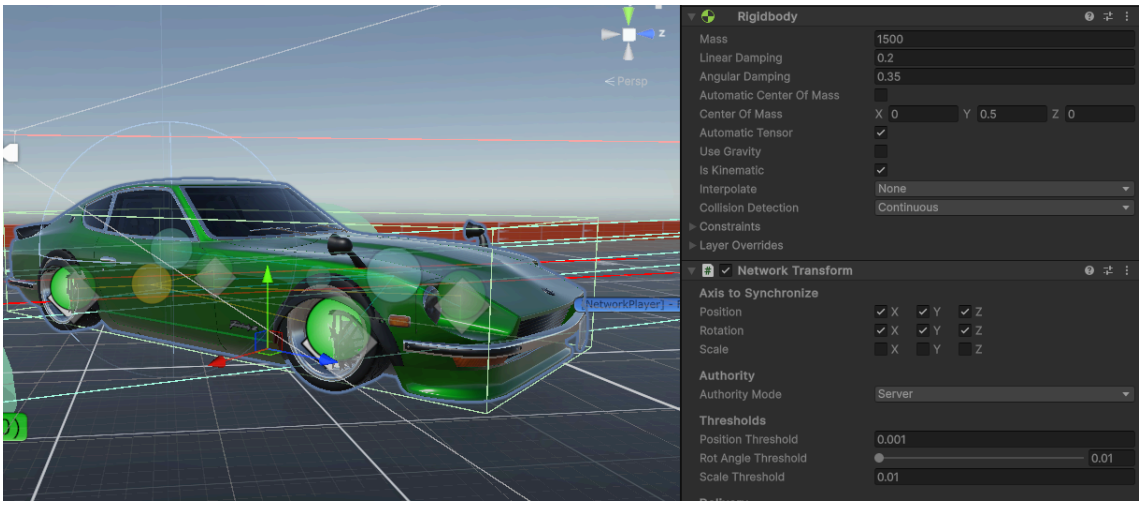


Figura 37. *Vehículo en red controlado por otro cliente mediante NetworkTransform*.

3.2.2.3 Dilema entre autoridad del servidor y autoridad del cliente

De entre las soluciones oficiales ofrecidas en los componentes de red, se permite una configuración de autoridad tanto del lado del servidor como del cliente, cada una con sus pros y sus contras.

A continuación se comparan las características de cada una de estas configuraciones.

	Autoridad del Servidor	**Autoridad del Cliente**
Seguridad	Alta Servidor capaz de evitar trampas.	Baja Clientes pueden manipular datos.

Sincronía	Segura Consistencia global garantizada entre los jugadores.	Arriesgada Posible desincronización entre jugadores.
Latencia	Mayor Las acciones deben ser gestionadas.	Mínima Control inmediato.
Procesamiento	Alta Carga mayor en el servidor.	Baja Repartida entre los clientes.
Complejidad	Alta Requiere predicción y reconciliación.	Menor Implementación inicial más simple.
Experiencia	Menos fluida El *lag* es inevitable.	Más fluída Experiencia receptiva inmediata.

Tabla 2. *Diferencias entre la autoridad del servidor y la autoridad del cliente*.

3.2.2.4 Ausencia de mecanismos oficiales de predicción de cliente y compensación de lag

Los componentes de red por defecto en Unity no incluyen sistemas avanzados de **compensación de latencia**, como **interpolación** o **extrapolación**. Esto resulta en desincronizaciones visibles o comportamientos erráticos en conexiones de calidad media o baja que dependen de desarrollos personalizados externos.

Figura 38. *Diferencia de posición del vehículo en el cliente (azul) respecto al servidor (fantasma)*.

En la imagen es posible visualizar la diferencia de posiciones de los jugadores del lado del servidor respecto al cliente a través de una conexión online. Si bien la transmisión es estable y de alta velocidad, cierto grado de lag es inevitable, y debe ser gestionado del lado del jugador para ofrecer una mejor experiencia de juego.

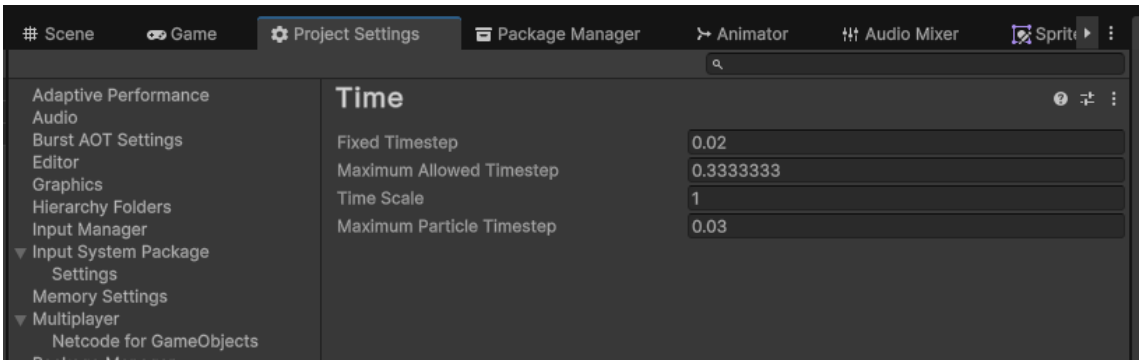
3.2.2.5 Consumo de ancho de banda innecesario

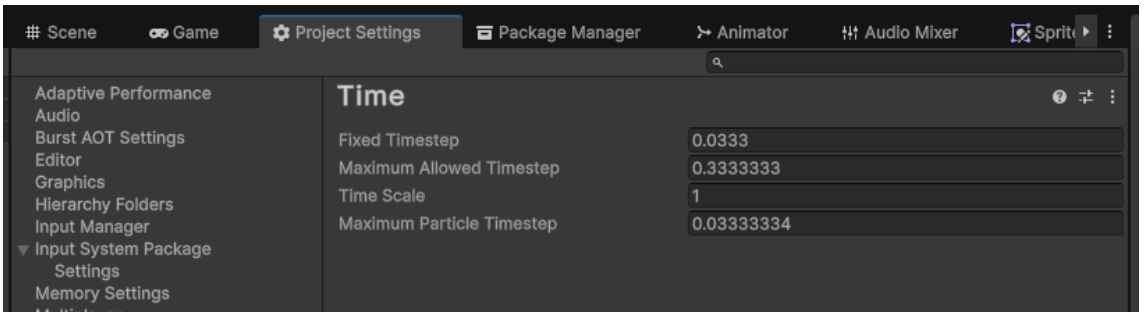
Aunque los componentes de red incluidos están razonablemente optimizados, una solución personalizada tiene la posibilidad de controlar la información transmitida y limitarla a lo estrictamente necesario.

Reducir los datos transmitidos, como la precisión excesiva en la posición o valores redundantes, podría mejorar la eficiencia del sistema y beneficiar a usuarios con conexiones más limitadas.

3.2.2.6 Frecuencia de actualización de físicas

Los componentes por defecto están pensados para funcionar correctamente con una frecuencia de actualización de 50 *ticks* de físicas por segundo, pues este es el valor por defecto.





Figuras 39 y 40. *Configuración de tickrate de físicas a 50 tick/s —por defecto, arriba— respecto a una configuración liviana a 30 tick/s —personalizado, abajo—.*

No obstante, es conveniente buscar alternativas capaces de mantener una experiencia fluida con frecuencias menores, pensando en dispositivos menos potentes o en optimizaciones para servidores de baja capacidad.

3.2.3 Desarrollo de la solución

3.2.3.1 Prototipo de Protovolkers

El proyecto Protovolkers nació con la intención de desarrollar un videojuego de carreras centrado en la simulación física de los vehículos. El uso intensivo del motor de físicas Nvidia PhysX fue establecido desde el inicio del proyecto. Y como tal, el desarrollo comenzó con la implementación del sistema de físicas de los vehículos, el *core* fundamental sobre el que giraría tanto el desarrollo del resto del proyecto, como más tarde, toda la experiencia de juego.

Para estructurar adecuadamente este sistema, se dividió el movimiento en tres grandes áreas de las que se ha explicado el estudio en el apartado **Videojuegos de carreras con físicas** visto anteriormente: **tracción**, **suspensión** y **dirección**. Esta separación lógica permitió abordar cada uno de ellos de manera modular, facilitando su implementación primera, así como mejora y depuración posteriores.

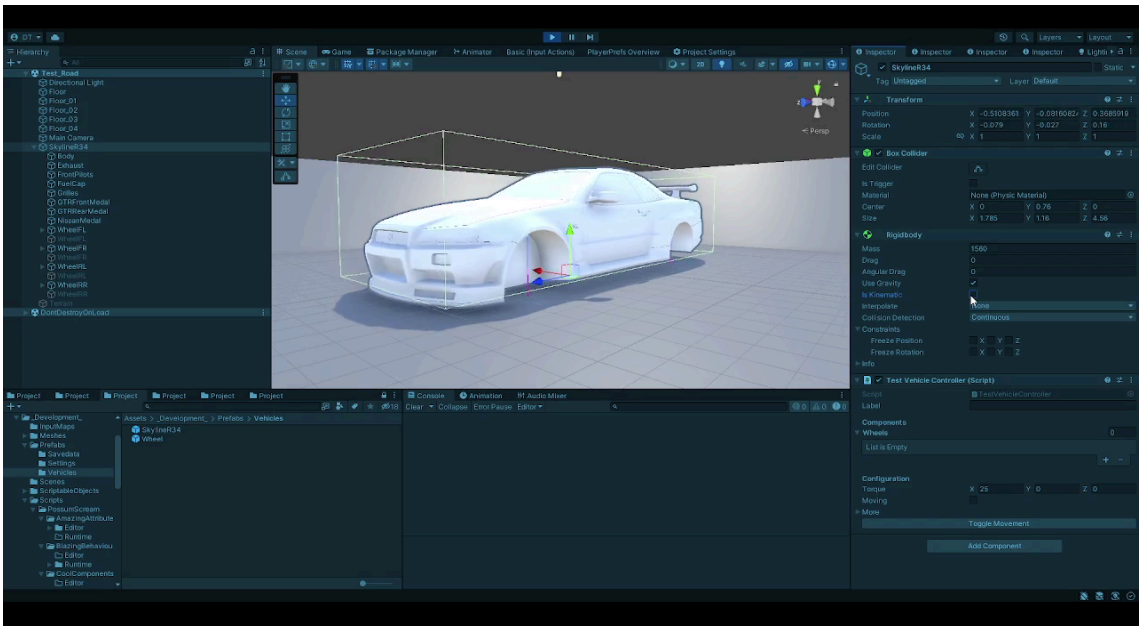


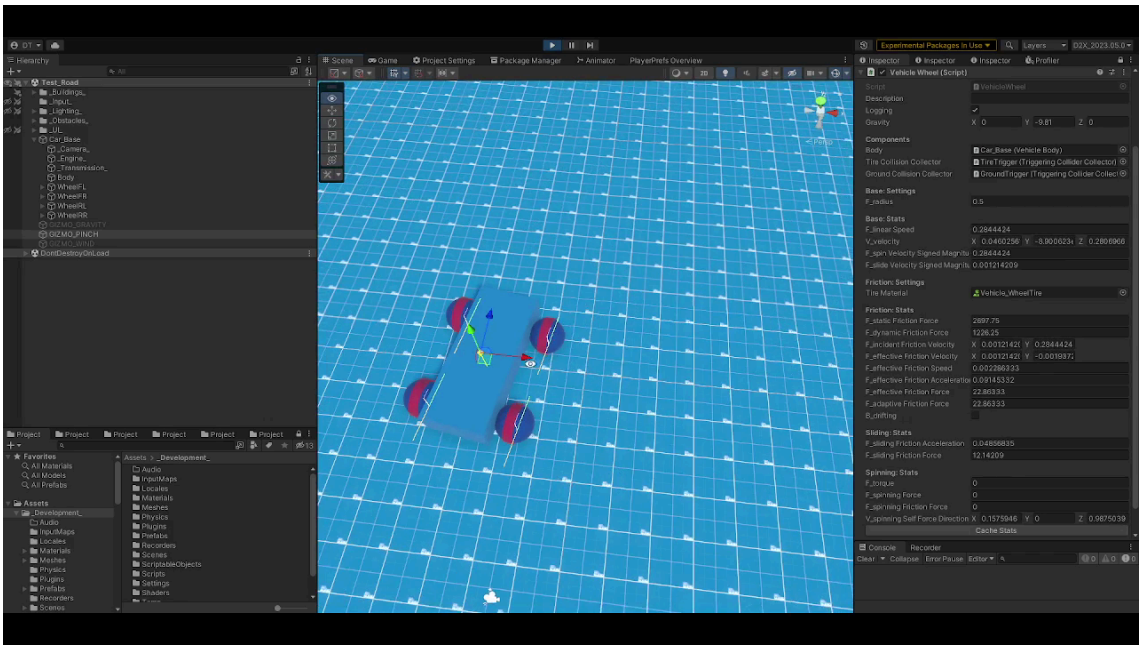
Figura 41. *[Devlog #2] Implementación del sistema de suspensión vehicular.*
Vídeo: https://youtu.be/ofZnNueQ1L0.

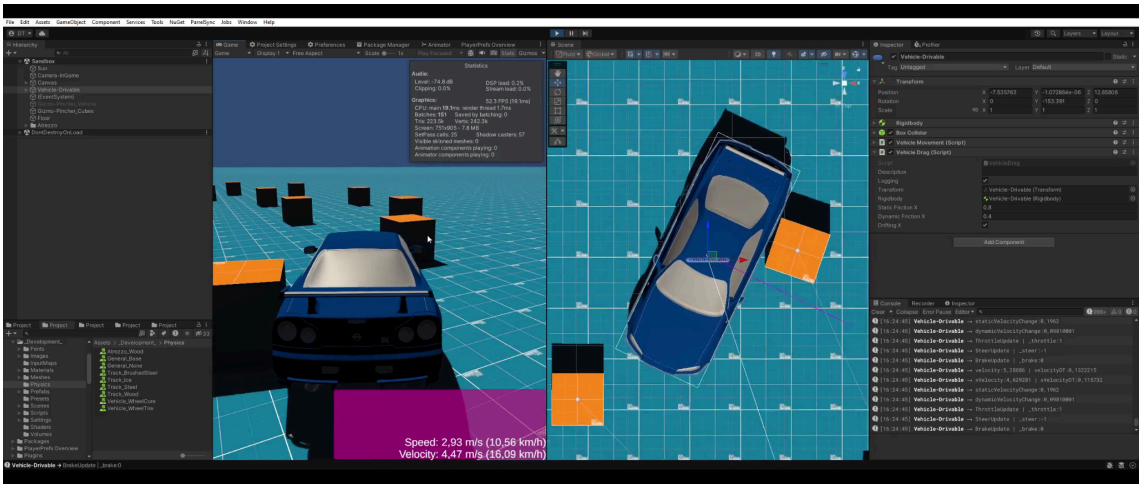
La primera funcionalidad implementada fue un sistema básico de suspensión, una de las partes fundamentales para facilitar la absorción de los baches e irregularidades del terreno y garantizar una conducción estable.

Han sido desarrolladas las simulaciones de las dos principales partes del sistema de suspensión:

- **Muelle**: Encargado de sustentar el cuerpo del vehículo por encima del nivel del suelo, ejerciendo una fuerza proporcional a la distancia respecto al suelo, teniendo en cuenta el peso y la velocidad en el eje gravitacional, siguiendo la **Ley de Hooke**.
- **Amortiguador**: Encargado de suavizar el efecto del muelle, pues por sí solo el muelle causaría una cantidad excesiva de rebotes que afectarían al manejo del vehículo.

El sistema de dirección también ha sido dividido en dos mecánicas diferenciadas: el agarre, que determina la fuerza de rozamiento que permite el giro del vehículo; y el impulso, que transmite la fuerza del motor y permite al vehículo impulsarse.





Figuras 42 y 43. *[Devlogs #12 y #16] Implementación del sistema de agarre de las ruedas.*
Vídeos: https://youtu.be/R_JUec0JVkc y https://youtu.be/JSFo6BRkEY4.

La mecánica de agarre funciona de manera que se aplica una fuerza equivalente en sentido opuesto —la fuerza de rozamiento— al deslizamiento lateral de cada uno de los puntos de agarre del vehículo —generalmente, los neumáticos sobre el asfalto—. En la Figura 22 vista durante el estudio de la aplicación de estas fuerzas, representa la fuerza resultante lateral.

La curva de deslizamiento ha sido simplificada a modo de porcentaje para la simplificación de los cálculos, pero siguiendo la **Fórmula de Pacejka** de deslizamiento de neumáticos como la representada en la Figura 13.

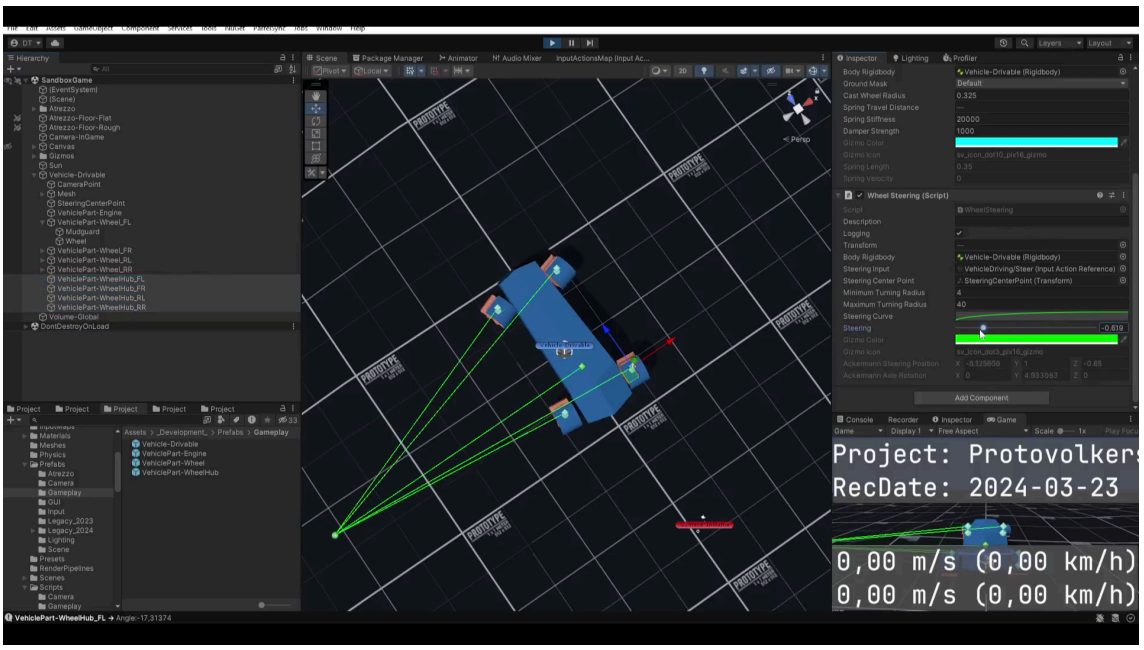


Figura 44. *[Devlog #25] Implementación del sistema de giro independiente por rueda*.
Vídeo: https://youtu.be/6jGwGeMKGl4.

Se desarrolló un sistema de dirección independiente para cada rueda basado en la **Geometría de Ackermann**. Este cálculo busca que cada rueda directriz trace una trayectoria con un radio distinto, adecuado a su posición, de manera que el punto sea coincidente para todas, mejorando el agarre de todo el vehículo.

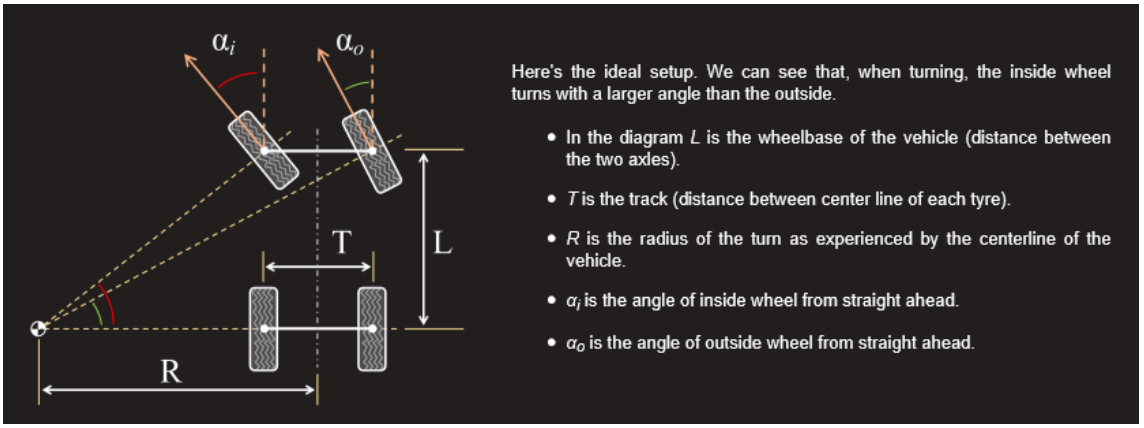


Figura 45. *Descripción del sistema de giro de Ackermann*. Fuente: DataGenetics.

Las motivaciones para la implementación de este sistema son las que siguen:

- Cada rueda cumple con las mecánicas de rozamiento y tracción proporcionalmente, similar al funcionamiento de los vehículos reales. Un acercamiento a este nivel de realismo se traduce en un mejor manejo dentro del videojuego.
- Un sistema general plenamente funcional para todos los casos, agiliza el desarrollo y la depuración a largo plazo, además de permitir el diseño de vehículos con **múltiples ejes de ruedas directrices** sin cambiar la implementación.

Un desarrollo como el de este proyecto no habría sido viable mediante improvisación, dada la complejidad técnica desde las primeras fases, es imperativo establecer un diagrama de clases que sirva como guía estructural para organizar y controlar los distintos scripts implicados no solo en el comportamiento de los vehículos, sino en todos los sistemas del juego completo.

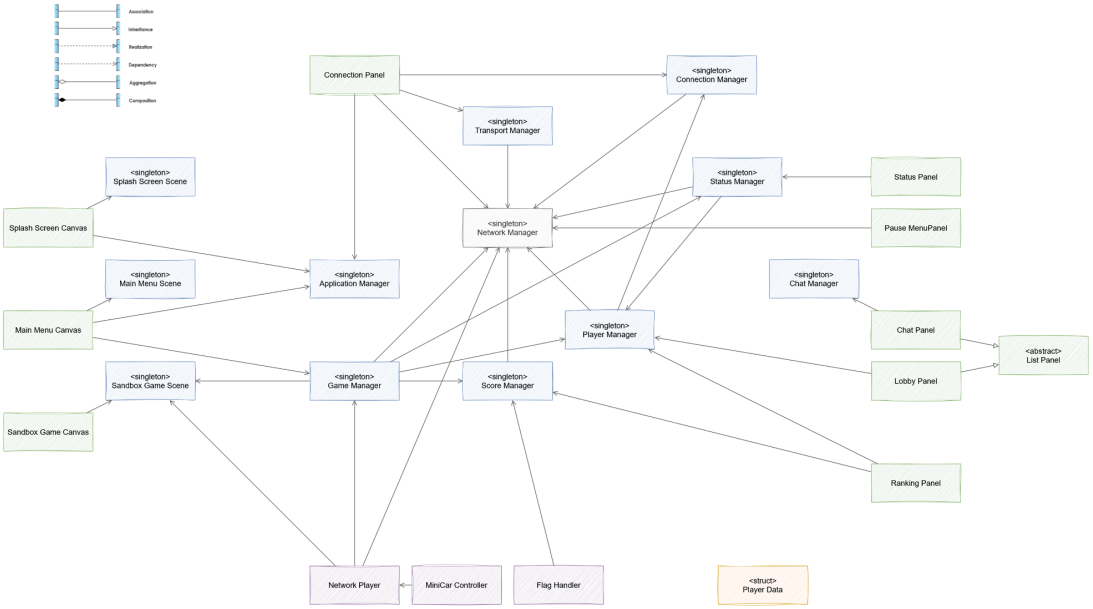


Figura 46. *Diagrama de clases de la primera versión del prototipo de Protovolkers*.

El primer diagrama de clases, que se muestra en la Figura 46, puede que fuera suficiente durante las fases iniciales del desarrollo, especialmente para estructurar y mantener el sistema de movimiento físico de los vehículos.

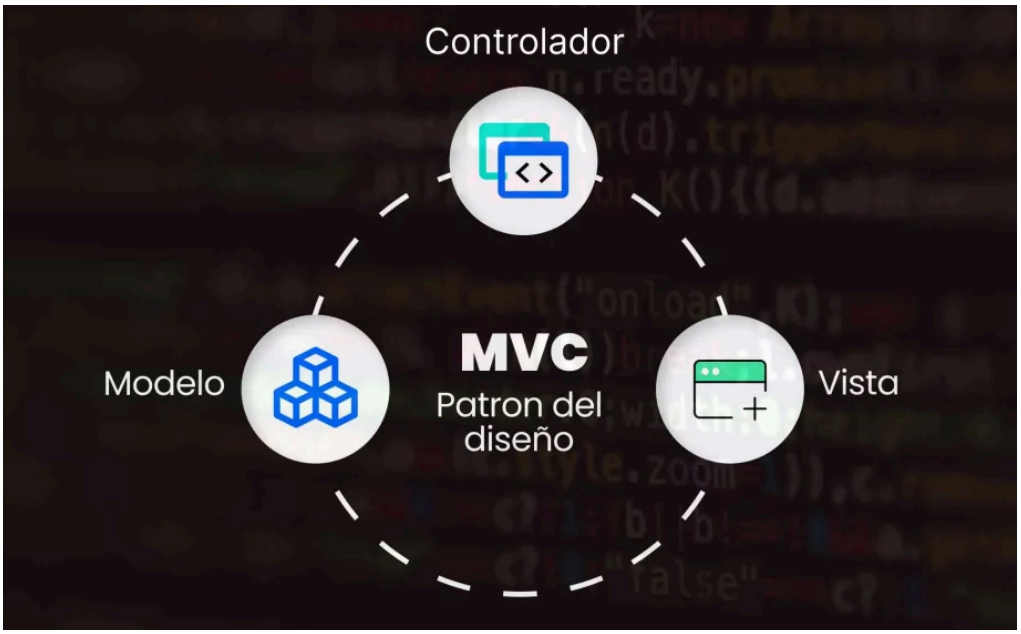


Figura 47. *Diagrama simplificado de Modelo-Vista-Controlador*. Fuente: hack(io).

Este diseño permitió implementar de forma clara las bases del comportamiento de tracción, suspensión y dirección vehiculares. Sin embargo, a medida que el proyecto fue creciendo e incorporando nuevos sistemas como el flujo de juego, mecánicas adicionales de gameplay y todo el árbol de interfaces de usuario, esta estructura inicial comenzó a volverse más complicada y difícil de manejar. Por ello, se puso en marcha un enfoque **Modelo-Vista-Controlador** (Figura 47).

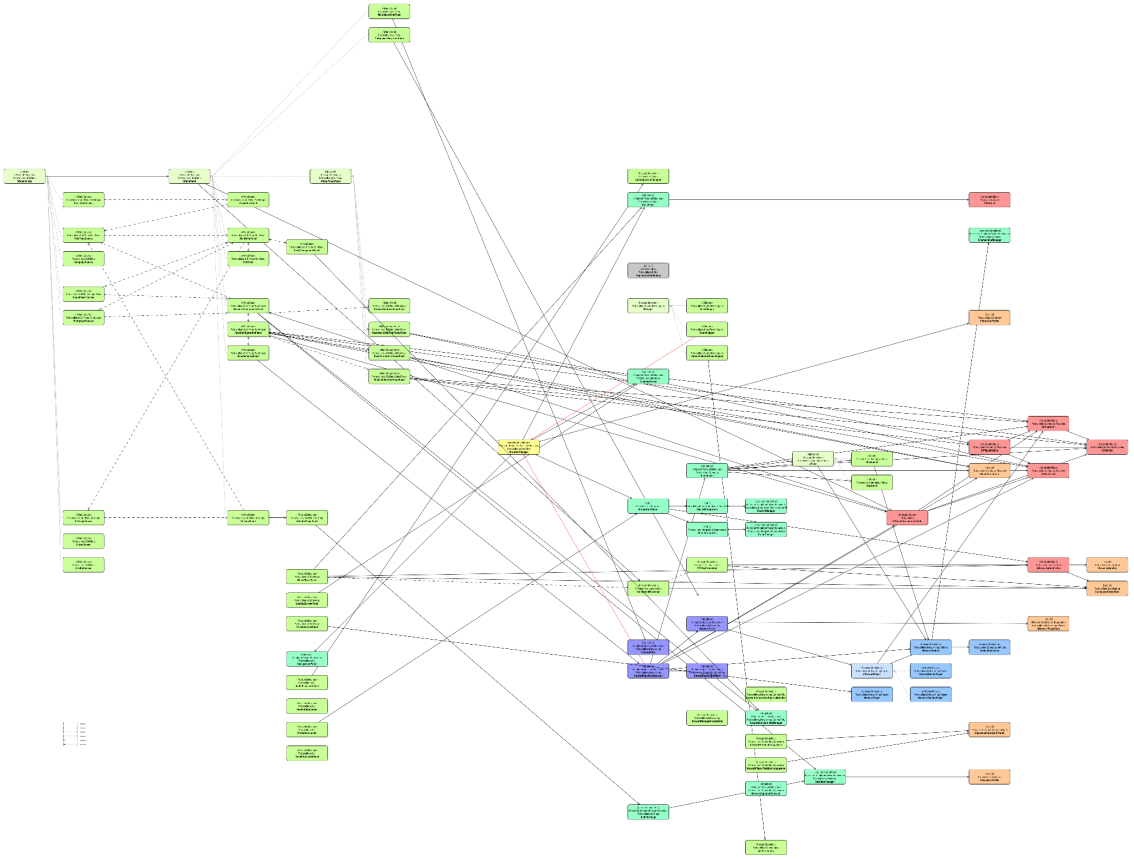


Figura 48. *Diagrama de clases de la segunda versión del prototipo de Protovolkers*.

En la Figura 48 se aprecia un cambio significativo en la organización del sistema tras una debida reestructuración y reorganización de los scripts del proyecto. En el contexto del patrón Modelo-Vista-Controlador (en adelante, MVC) esta reorganización permite una mejor separación de responsabilidades y un flujo de dependencias más lógico y controlado.

En el lado izquierdo del diagrama se encuentran los scripts dedicados a la parte visual e interfaces gráficas de usuario (la **Vista** dentro del modelo MVC), mientras que en el lado derecho se ubican los encargados de la lógica de juego, su flujo y control de otros sistemas internos (el **Controlador**). A su vez, varios de estos scripts acceden a una tercera sección situada a la derecha de estos últimos, compuesta por estructuras de datos y contenedores de información sin lógica de comportamiento (el **Modelo**).

Puntualmente, ciertos scripts visuales acceden también a esta sección de datos, pero en ningún caso se invierte la dirección de dependencia.

Si bien este nuevo tipo de diagrama ha resultado de gran utilidad durante varios meses de desarrollo, especialmente al facilitar la integración progresiva de nuevas funcionalidades dentro de la nueva estructuración del proyecto, con el aumento de la complejidad general y la incorporación de un sistema avanzado de físicas, además de su necesaria integración con el flujo completo del juego, la cantidad de scripts implicados creció considerablemente.

Como consecuencia, el diagrama comenzó a volverse excesivamente denso y difícil de seguir, perdiendo su claridad para volverse una tarea tediosa de seguir y confusa de interpretar.

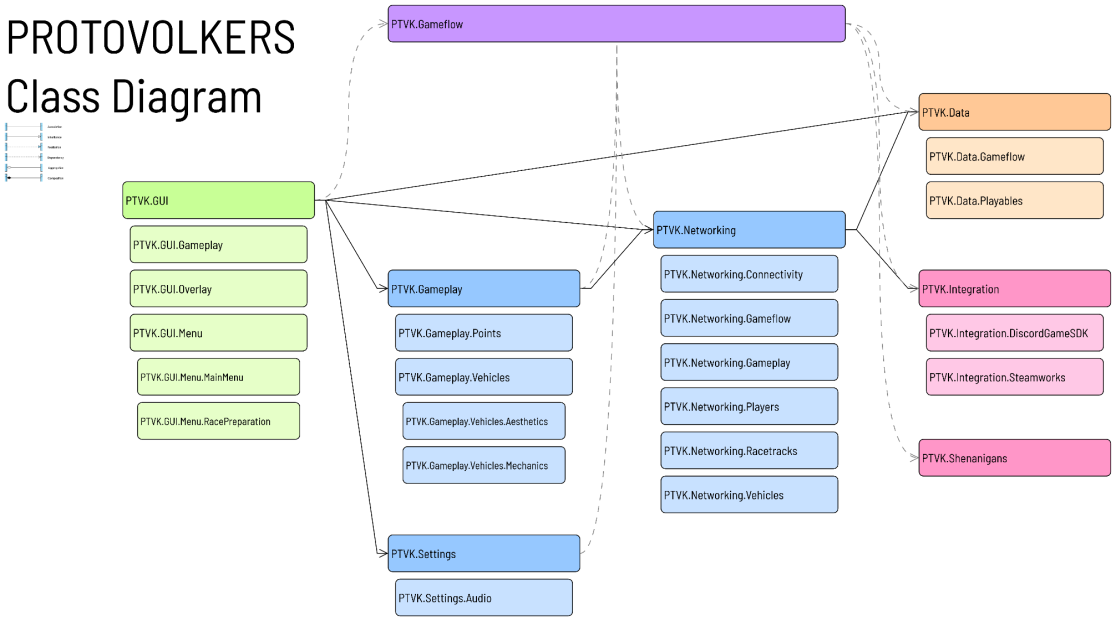


Figura 49. *Diagrama de clases de la versión final del prototipo de Protovolkers*.

Finalmente, en el diagrama representado en la Figura 49, se aprecia una simplificación significativa respecto a los anteriores. Este cambio se debe a un cambio en la manera de enfocar el esquema de clases: de un diagrama detallado a nivel de script individual, se ha pasado a un esquema estructurado a nivel de espacios de nombres o *namespaces*.

En esta nueva representación, todos los scripts dentro de un mismo *namespace* comparten visibilidad y cierto grado de accesibilidad mutua, sin embargo se establece un flujo jerárquico claro de dependencias entre los diferentes espacios de nombres, manteniendo así una organización en forma de árbol similar a la del modelo visto en la Figura 48. Esta disposición permite conservar el patrón Modelo-Vista-Controlador a la vez que recupera la utilidad visual y la comprensión del apartado de código del proyecto.

Adicionalmente, el diagrama incorpora un complemento estructural propio de otro patrón: el bus de eventos o *event bus*. Consiste en un *singleton* centralizado que actúa como canal de distribución para los eventos principales que tienen lugar a lo largo del flujo del juego, tales como la carga de escenas, la conexión de nuevos jugadores o la finalización de una carrera, entre muchos otros.

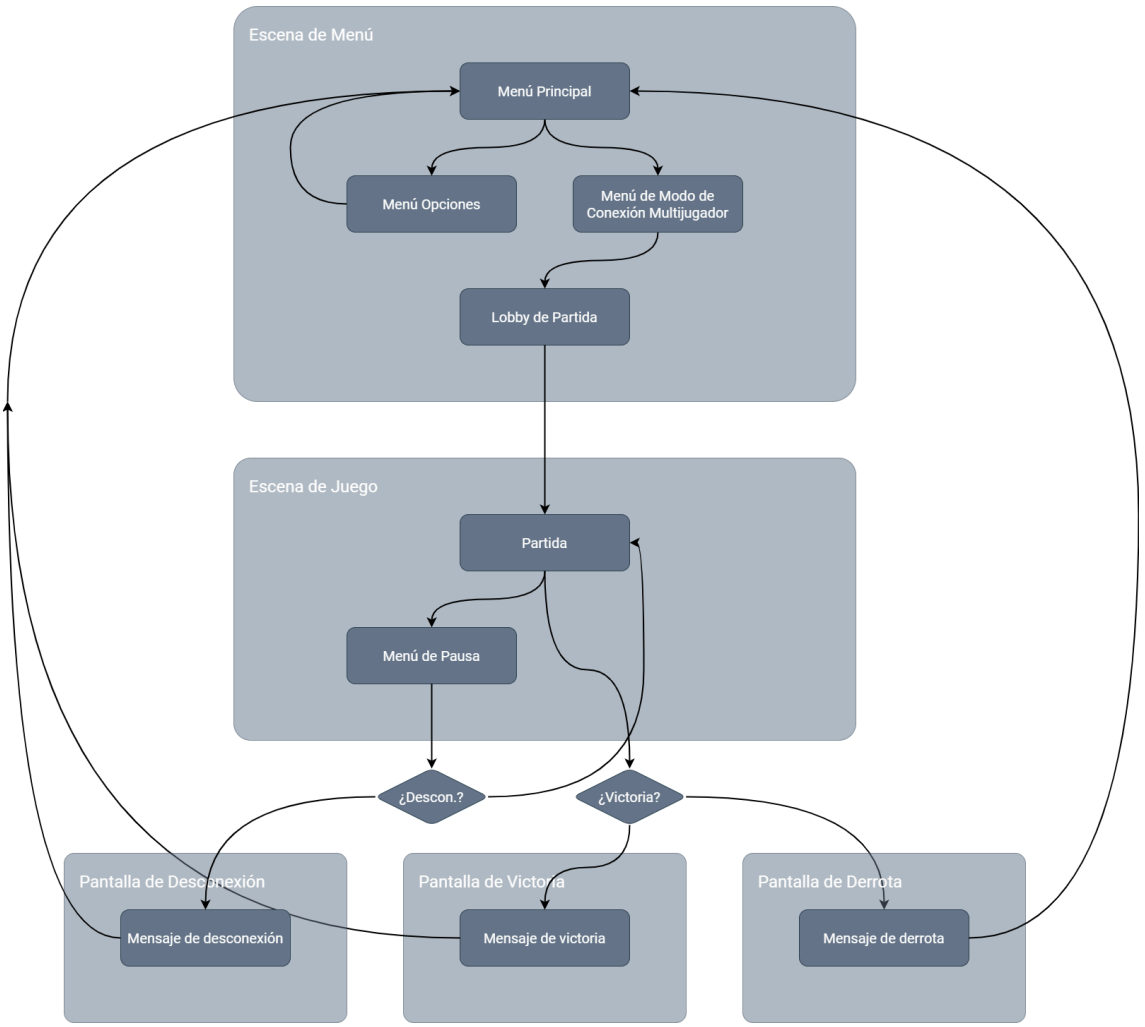


Figura 50. *Diagrama de flujo de escenas de la primera versión del prototipo de Protovolkers*.

En la anterior figura se muestra el primer diagrama de escenas diseñado para el proyecto, elaborado durante una etapa muy temprana del desarrollo. Este esquema fue concebido específicamente para cumplir con una entrega parcial correspondiente a un ejercicio académico del curso anterior, por lo que su estructura es mucho más simple acorde con los requisitos que fueron impuestos en su momento.

Por otro lado, la Figura 51 a continuación representa el diagrama de flujo de escenas planteado para el futuro del proyecto, orientado ya hacia una versión más avanzada de Protovolkers. Este nuevo diseño contempla un flujo de partida mucho más natural y modular, inspirado por otros títulos del género, y que facilitan su desarrollo.

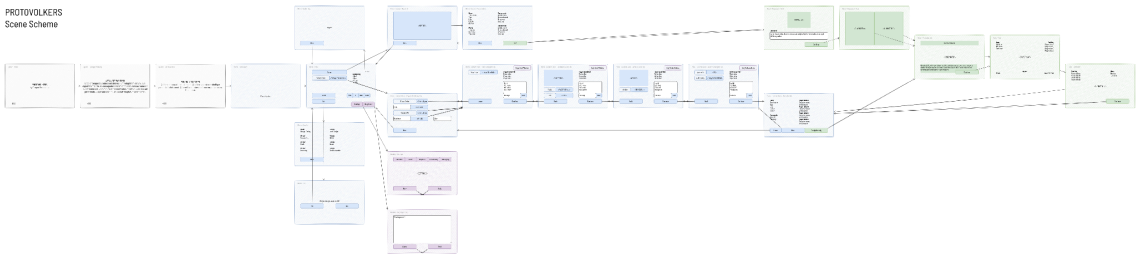


Figura 51. *Diagrama de flujo de escenas de la futura versión de Protovolkers*.

Cabe destacar que el prototipo que se presenta en este Trabajo de Fin de Grado se sitúa conceptualmente a medio camino entre ambos diagramas: se conservan ciertas estructuras simples heredadas del primer diseño, con la integración de algunos de la mayoría de conceptos del segundo. Una vez realizada la entrega del TFG, se prevé llevar a cabo una refactorización completa del flujo de escenas, con el objetivo de adoptar plenamente la estructura descrita en la Figura 51 de cara al desarrollo futuro.

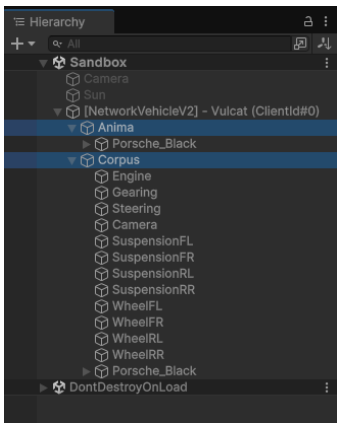
3.2.3.2 Sistema de físicas en red: *corpus* y *anima*

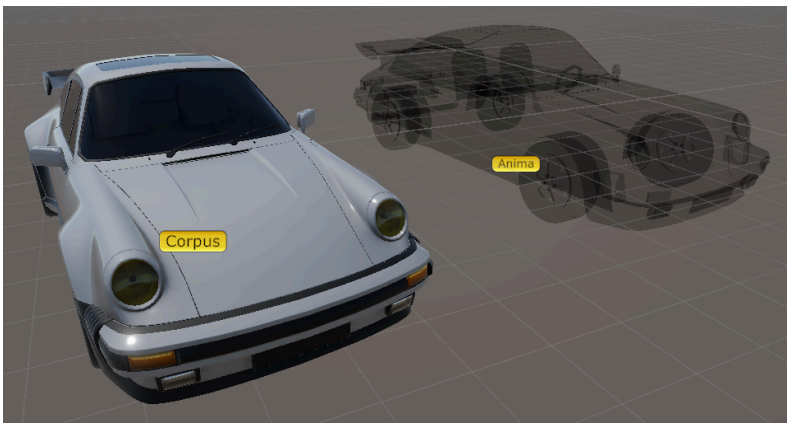
El sistema de físicas en red constituye el eje central de este Trabajo de Fin de Grado, siendo el aspecto técnico más complejo y determinante en la experiencia multijugador de Protovolkers. Su objetivo principal es lograr que múltiples jugadores puedan interactuar entre sí en tiempo real mediante vehículos controlados por físicas realistas, conservando tanto la sensación de conducción local como la coherencia entre clientes en red. Para ello, se ha desarrollado una solución personalizada que combina elementos de predicción, sincronización y gestión de colisiones.

Antes de abordar el diseño e implementación de la solución propia, se realizó un estudio de las principales metodologías existentes para la sincronización de físicas en red. Las dos más representativas son las propuestas por Glenn Fiedler tal y como se ha visto anteriormente en la sección Videojuegos con físicas en red: tanto el modelo "*Deterministic Lockstep*", presentado en 2014, como el modelo de "*State Synchronization*", expuesto en 2015.

El primero se ayuda de mantener una simulación física completamente determinista en todos los clientes, enviando únicamente las entradas de usuario o *input* a través de la red. El principal requisito para ello es contar con un motor de físicas completamente determinista, lo que garantiza una sincronización precisa si las condiciones son ideales, pero a su vez resulta extremadamente delicado frente a cualquier desincronización y requiere una arquitectura perfectamente homogénea entre jugadores. Esta sin embargo no es una solución válida, pues como ya se ha visto anteriormente, el motor Nvidia PhysX no es determinista, y la desincronización entre clientes está asegurada.

El segundo enfoque transmite directamente los estados físicos más importantes, a saber: posición, rotación, velocidad lineal y velocidad angular, de los objetos a todos los demás clientes jugadores. Este método es mucho más resiliente ante discrepancias entre los estados de los objetos de red de los clientes, aunque conlleva un consumo mayor de ancho de banda.





Figuras 52 y 53. *Jerarquía del vehículo en red —izquierda— con la separación de cuerpo (*corpus*) para la simulación local, y alma (*anima*) para la simulación en remoto —derecha—.*

El sistema desarrollado parte del enfoque de sincronización de estados, pero introduce una modificación significativa a la hora de afrontar los problemas de consistencia en la red a la vez que facilidad de desarrollo de netcode: en lugar de utilizar una única representación física por vehículo, cada entidad vehicular de red se descompone a su vez en dos componentes diferenciados fuertemente vinculados: el "*corpus*" o cuerpo y el "*anima*" o alma.

El ***corpus*** es una instancia física simulada en el cliente local, que se encarga de proporcionar una experiencia de conducción inmediata y receptiva pues responde de manera directa a las entradas del jugador, y que luego transmitirá a su *anima* para la transmisión, gestionando tanto las colisiones como el movimiento en tiempo real, sin latencias visibles.

Por otro lado, el ***anima*** representa el estado validado por el servidor, y es el que se encarga de calcular las físicas globales, verificar colisiones reales entre jugadores y aplicar las correcciones necesarias para mantener la coherencia del mundo compartido. De esta manera, se logra un equilibrio entre responsividad y seguridad, base esencial para un juego competitivo basado en contacto físico entre vehículos.

```
Event function  Tabernero
private void FixedUpdate()
{
    if (!base.IsSpawned) return;
    if (this.IsFrozen) return;
    if (base.IsOwner) OwnerFixedUpdate();
    else ObserverFixedUpdate();
    if (base.IsServer) ServerFixedUpdate();
}
```

Figura 54. *El método de actualización de las físicas efectúa actualizaciones según si la instancia del vehículo es propietaria, observadora, o gestionada por el servidor.*

Dado que el sistema es directamente dependiente de las actualizaciones del motor de físicas, las peticiones de red, aunque pocas, son efectuadas en el mismo método interno.

```
Frequently called  1 usage  Tabernero
private void OwnerFixedUpdate()
{
    UpdateAnimaRigidbodyAtServerRpc(_corpusRigidbody.position, _corpusRigidbody.rotation,
        _corpusRigidbody.linearVelocity, _corpusRigidbody.angularVelocity);
}

Frequently called  1 usage  Tabernero
private void ObserverFixedUpdate()
{
    if (CheckForNeededCompensation( rigidbodyA: _animaRigidbody,  rigidbodyB: _corpusRigidbody))
        MatchRigidbodyCorpusToAnima();
}

Frequently called  1 usage  Tabernero
private void ServerFixedUpdate()
{
    UpdateAnimaRigidbodyAtClientRpc(_animaRigidbody.position, _animaRigidbody.rotation,
        _animaRigidbody.linearVelocity, _animaRigidbody.angularVelocity);
}
```

Figura 55. *Los métodos para enviar el estado del vehículo al servidor, actualizarlo desde el servidor, o solicitar a todos los clientes que actualicen los estados de los contrincantes.*

El método de actualizaciones de las físicas realiza las órdenes pertinentes mediante llamadas RPC tanto al servidor como a todos los clientes, según el caso de la instancia.

```
[Rpc( target: SendTo.Server)]
Frequently called  1 usage  Tabernero
private void UpdateAnimaRigidbodyAtServerRpc(Vector3 position, Quaternion rotation,
    Vector3 linearVelocity, Vector3 angularVelocity, RpcParams rpcParams = default)
{
    float latency = NetworkPartyV2.CachedInstance.GetPlayerRTT(rpcParams.Receive.SenderClientId);
    _animaRigidbody.SetPoint(
        position: position + (latency * linearVelocity),
        rotation: rotation * Quaternion.Euler(latency * angularVelocity));
    _animaRigidbody.SetVelocity(linearVelocity, angularVelocity);
}


[Rpc( target: SendTo.ClientsAndHost)]
Frequently called  1 usage  Tabernero
private void UpdateAnimaRigidbodyAtClientRpc(Vector3 position, Quaternion rotation,
    Vector3 linearVelocity, Vector3 angularVelocity, RpcParams rpcParams = default)
{
    float latency = NetworkPartyV2.CachedInstance.GetPlayerRTT(rpcParams.Receive.SenderClientId);
    _animaRigidbody.SetPoint(
        position: position + (latency * linearVelocity),
        rotation: rotation * Quaternion.Euler(latency * angularVelocity));
    _animaRigidbody.SetVelocity(linearVelocity, angularVelocity);
}
```

Figura 56. *Los métodos para sincronizar los estados del* anima *tanto del servidor como de los clientes en métodos independientes.*

La sincronización de los cuerpos rígidos o *rigidbodies* de los vehículos se hace mediante el establecimiento de los valores clave que definen su situación: posición, rotación y velocidades lineal y angular tal y como se ha explicado anteriormente.

```
Frequently called  1 usage  new *
private void MatchRigidbodyCorpusToAnima() => _corpusRigidbody.SyncMovement(_animaRigidbody);
```

Figura 57. *Comando de resincronización para igualar* corpus *a* anima.

Una de las mayores ventajas de este sistema es que la diferencia de estado antes y después del establecimiento de los valores es calculado íntegramente de manera interna por el propio motor, devolviendo los *deltas* de cada uno de los parámetros.

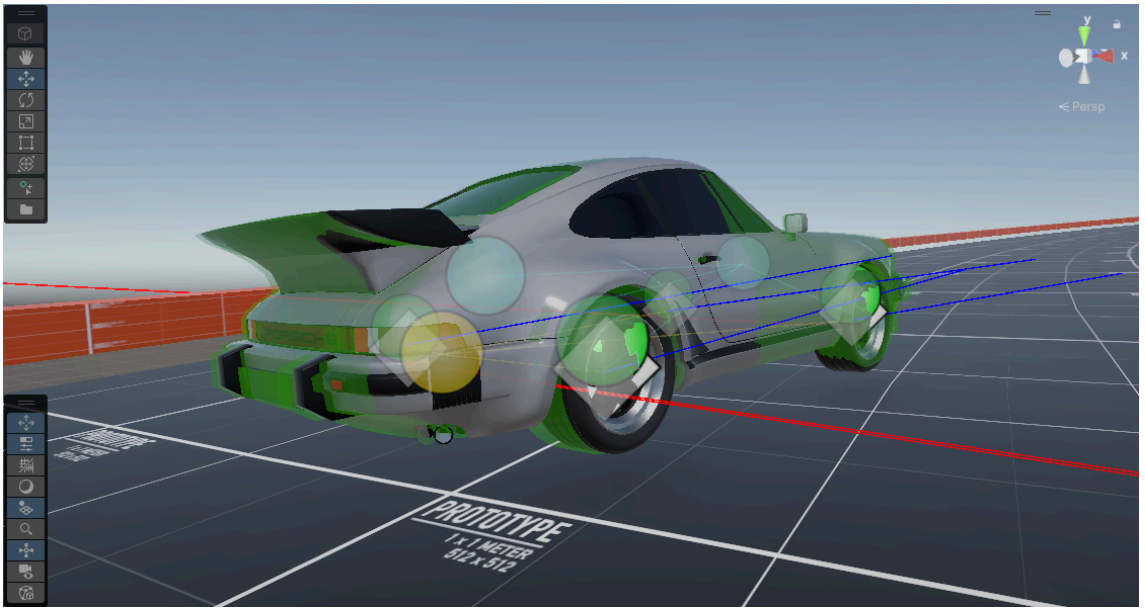


Figura 58. *Visualización de cuerpo (gris metalizado) y alma (verde transparente) parcialmente solapados.*

Otra de las ventajas adquiridas por esta técnica es que, de manera similar al componente *NetworkRigidbody* en el paquete oficial de Netcode for GameObjects, es posible configurar y controlar si los *deltas* de las posiciones son suficientemente elevados y reconciliar los estados de *corpus* y *anima* cuando se supera cierto límite especificado.

```
Frequently called  1 usage  Tabernero
private bool CheckForNeededCompensation(Rigidbody rigidbodyA, Rigidbody rigidbodyB)
{
    Vector3 delta = rigidbodyA.position - rigidbodyB.position;
    return (delta.x.Abs() > _compensationOffset.x) ||
           (delta.y.Abs() > _compensationOffset.y) ||
           (delta.z.Abs() > _compensationOffset.z);
}
```

Figura 59. *Sistema de comprobación para la reconciliación (según el* delta *de la compensación).*

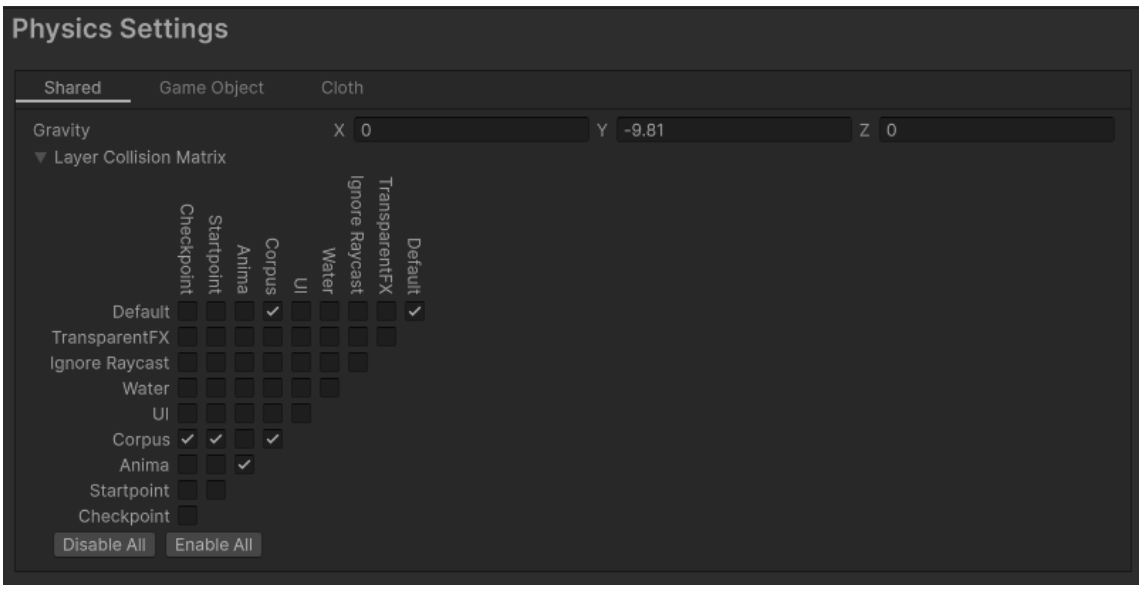


Figura 60. *Matriz de la configuración de físicas para las capas de* corpus *y* anima.

Cabe destacar que, para gestionar correctamente las interacciones físicas, es imperativo definir en la matriz de colisiones capas o *layers* tanto para *corpus* como para *anima*, pues operan de manera independiente.

Los componentes *corpus* están configurados para colisionar entre sí y con el escenario físico habitual, es la representación de los vehículos que los jugadores perciben y controlan.

En cambio, los componentes *anima* —al estar destinados a la simulación en el lado del servidor— están diseñados para colisionar únicamente entre sí, evitando cualquier interacción innecesaria con cualquier otro elemento del escenario o del cliente.

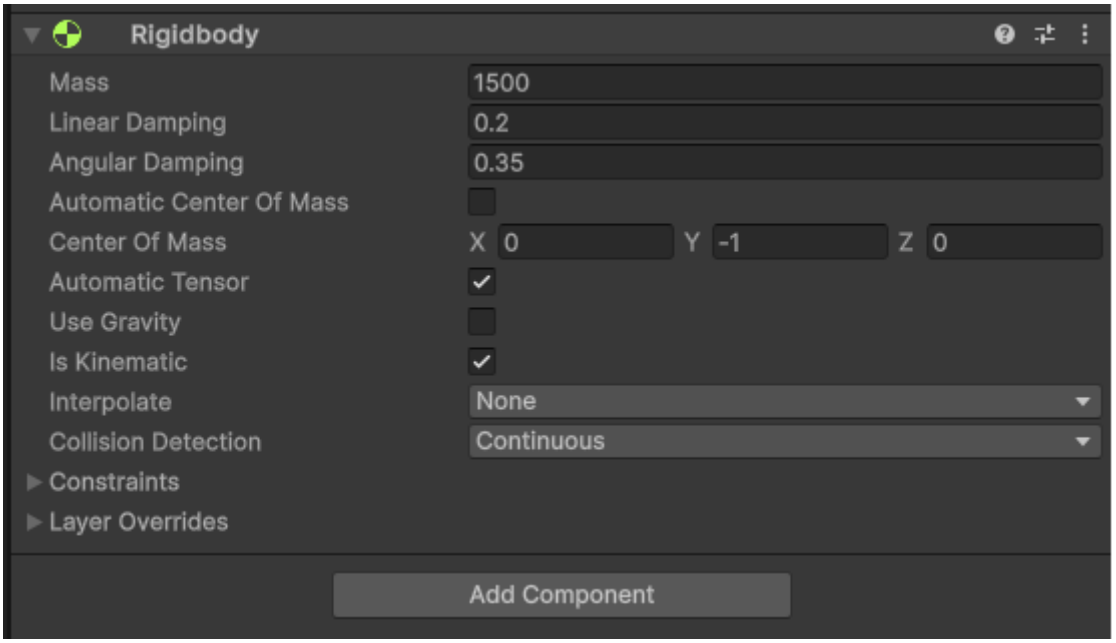


Figura 61. *El componente Rigidbody del* anima *no hace uso de la gravedad ni colisiona con otras capas.*

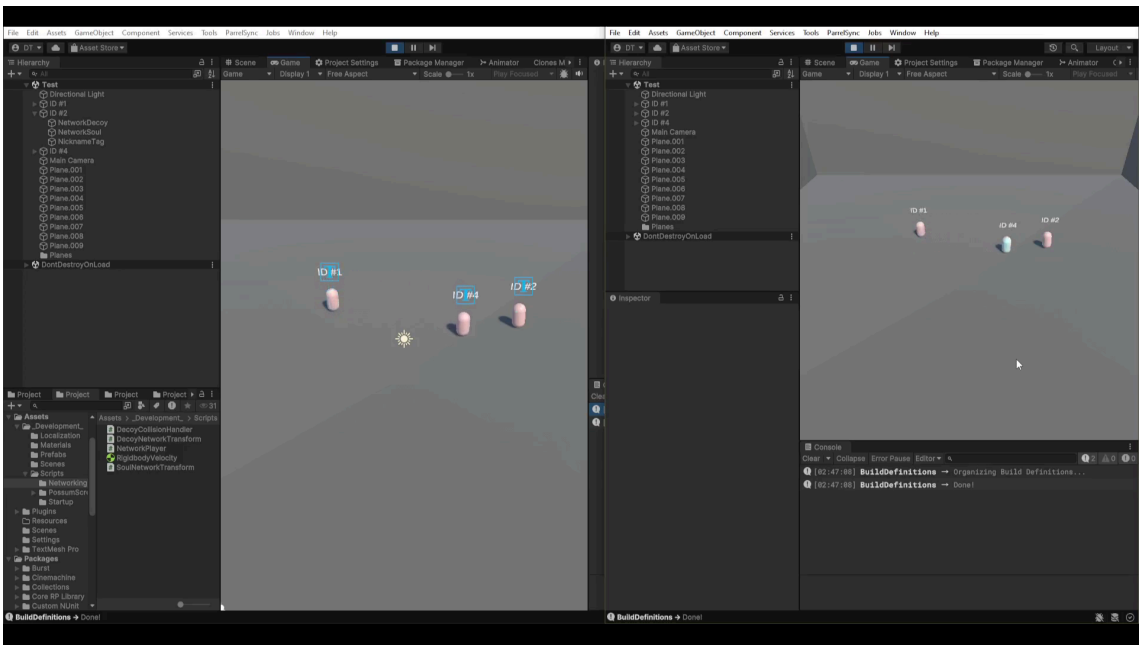


Figura 62. *[Devlog #32] Implementación del sistema de físicas en red*.
Vídeo: https://youtu.be/6fX0j74q9IM.

Durante las primeras fases del desarrollo del sistema de físicas en red, se optó por trabajar en un entorno completamente vacío, con el objetivo de aislar el comportamiento básico del sistema y validarlo antes de integrarlo con el resto del proyecto.

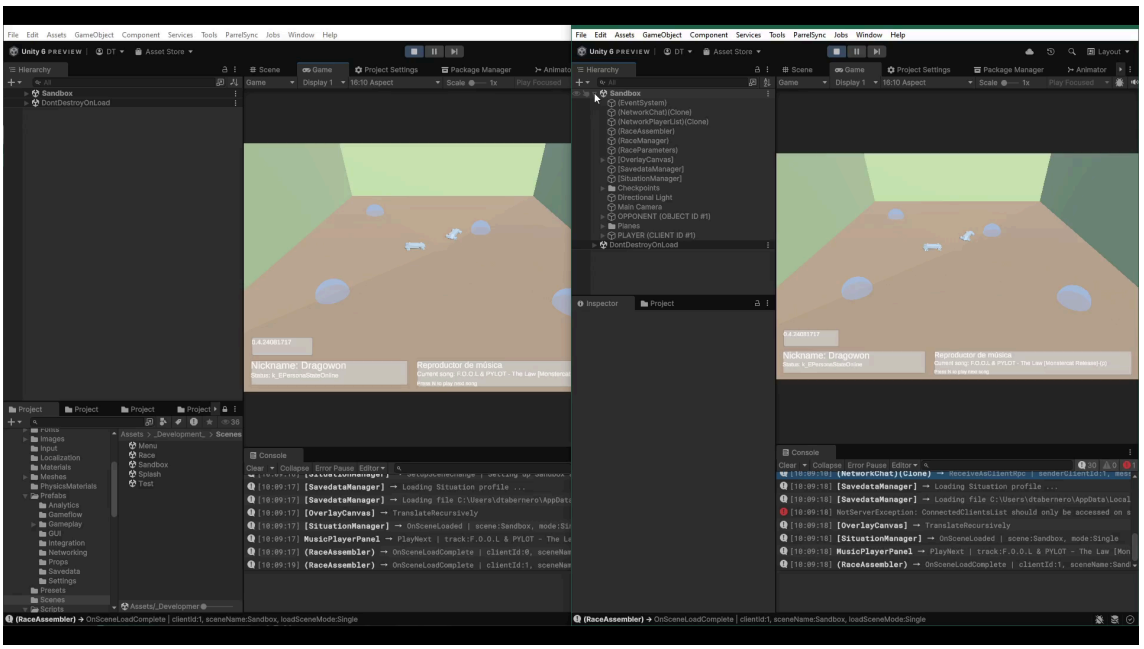


Figura 63. *[Devlog #36] Implementación del sistema de físicas en red en los vehículos*.
Vídeo: https://youtu.be/eGekxqvShbs.

Tan pronto como el sistema fue comprobado en un escenario genérico, las que al principio eran cápsulas de prueba con movimiento básico, fueron sustituídas por los vehículos del proyecto.

3.2.3.3 Integración en el proyecto

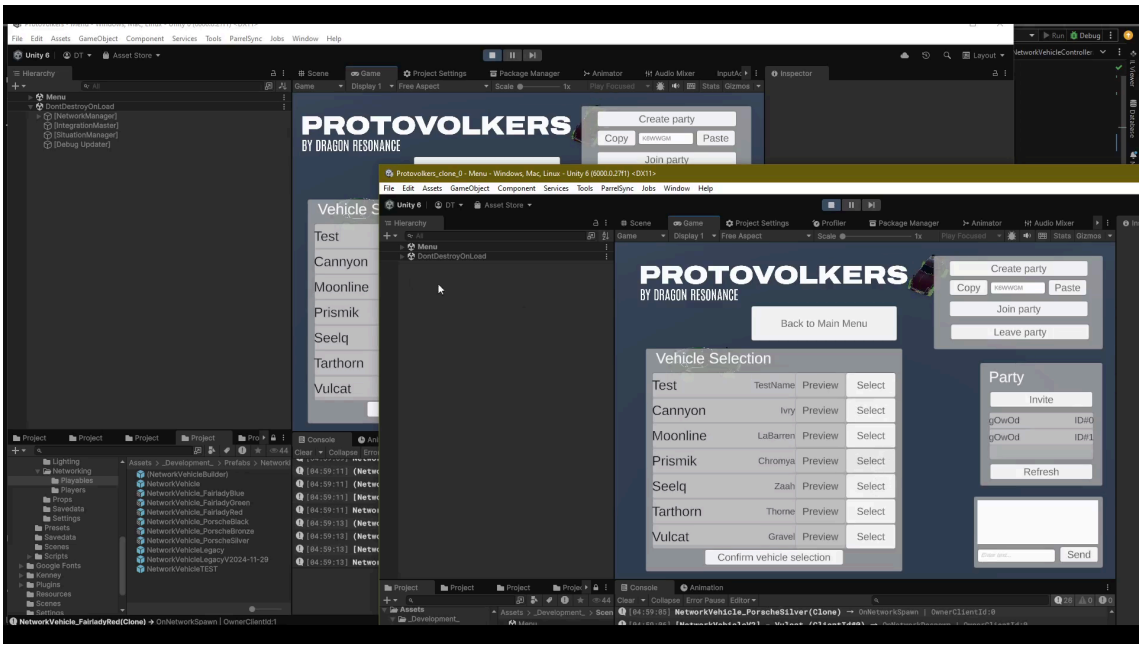


Figura 64. *[Devlog #55] Implementación del flujo de juego básico del prototipo*.
Vídeo: https://youtu.be/CuiqR654QA8.

Tras haber asentado las bases del sistema de físicas, el desarrollo de Protovolkers continuó con la fase centrada en el flujo completo del juego.

La parte importante del proyecto en este punto es garantizar que el jugador pueda experimentar una sesión de juego continua, lo cual implica implementar un ciclo completo tal y como ha sido previamente especificado tanto en la Figura 50 del primer prototipo, como en la futura implementación de la Figura 51:

1. Arranque del juego
2. Navegación por los menús
3. Configuración del circuito
4. Selección del vehículo
5. Inicio de la carrera
6. Manejo del vehículo
7. Finalización de la carrera
8. Regreso al menú principal

Naturalmente, se entiende la posibilidad de comenzar una nueva carrera sin necesidad de reiniciar el juego.

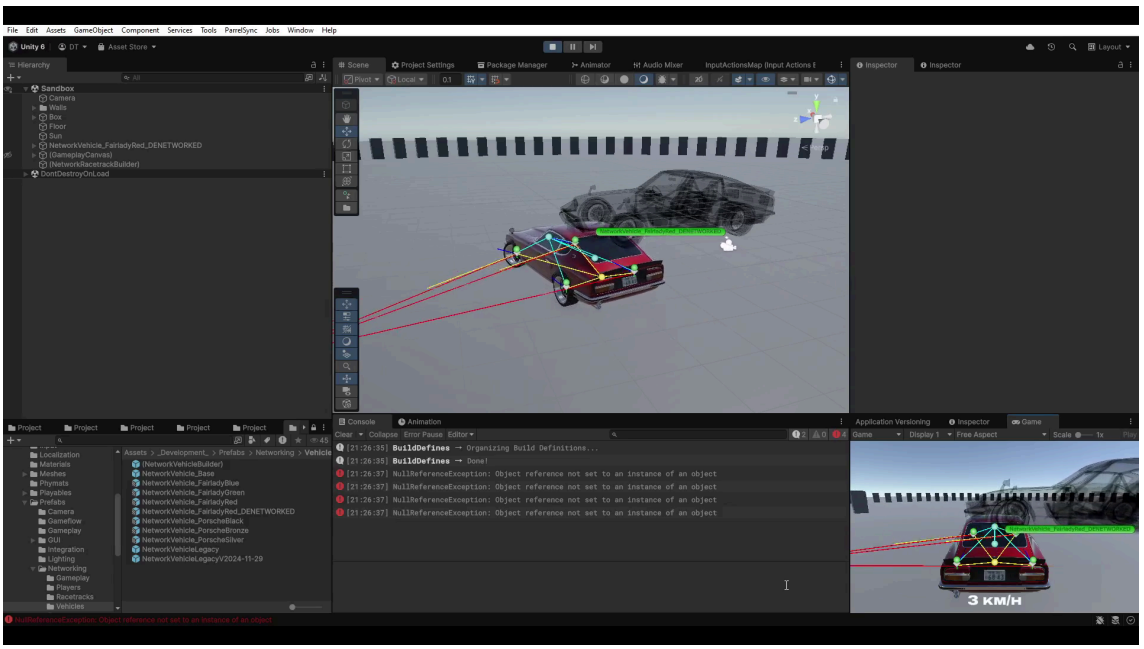


Figura 65. *[Devlog #60] Optimización y depuración de los sistemas de control del vehículo para su funcionamiento en red*. Vídeo: https://youtu.be/eV7RZMJoV1I.

Si bien en la primera implementación de los sistemas del vehículo la prioridad era que todo funcionase de manera básica, bastantes de estos sistemas presentan problemas importantes en términos de estabilidad y experiencia de usuario: comportamientos poco realistas o inconsistentes, control muy limitado para el jugador, o incluso *bugs* persistentes que, sin romper completamente la jugabilidad, sí resultaban frustrantes incluso durante las etapas de desarrollo y depuración, obligando a dedicar tiempo a su corrección antes de lo previsto.

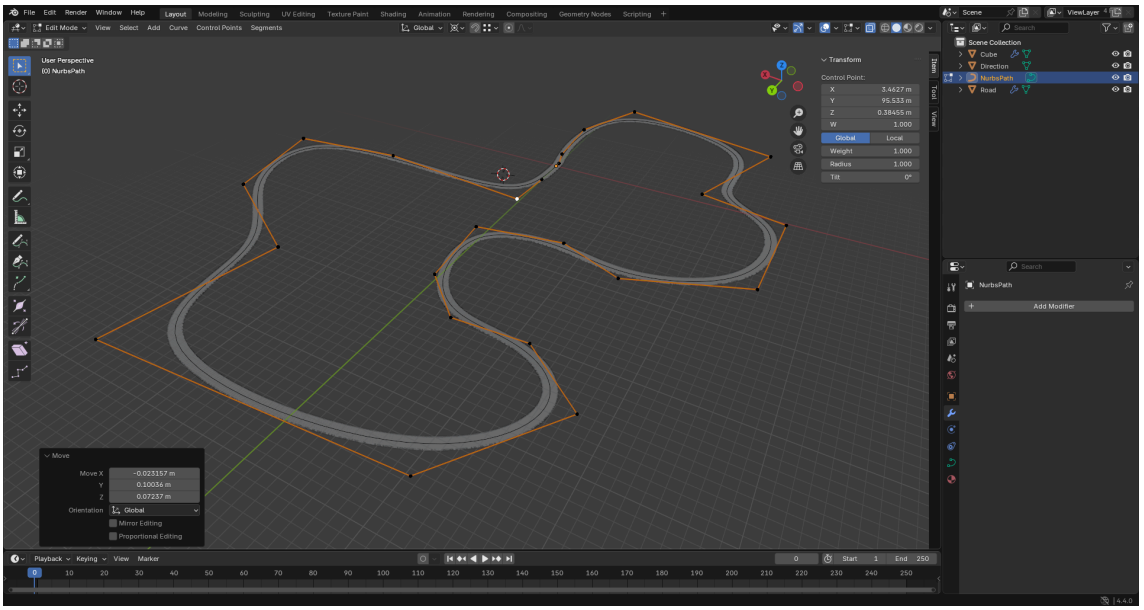


Figura 66. *[Devlog #64] Diseño y automatización de un circuito básico en Blender*.

Otra tarea no contemplada inicialmente fue la creación de un sencillo circuito en Blender.

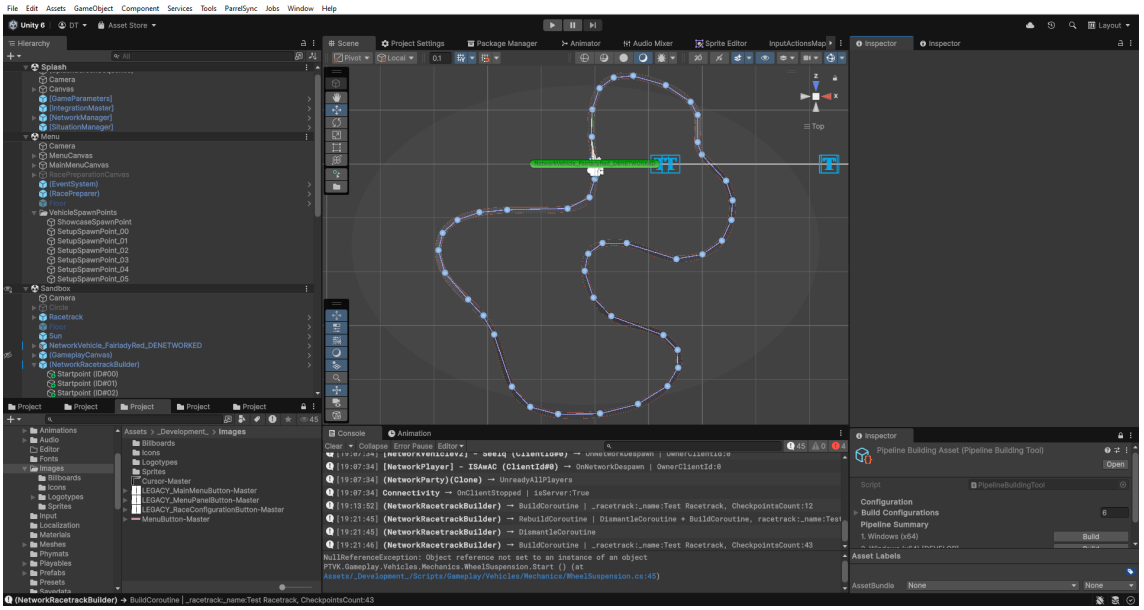


Figura 67. *[Devlog #69] Importación automatizada del circuito en Unity*.

La creación del prototipo en Blender es una decisión tomada con criterio: teniendo pleno conocimiento de sus herramientas de generación procedural y su integración con Unity, fue posible tener el circuito completamente funcional e integrado en la escena de juego en tan solo unas pocas horas.

3.2.4 Resultado final

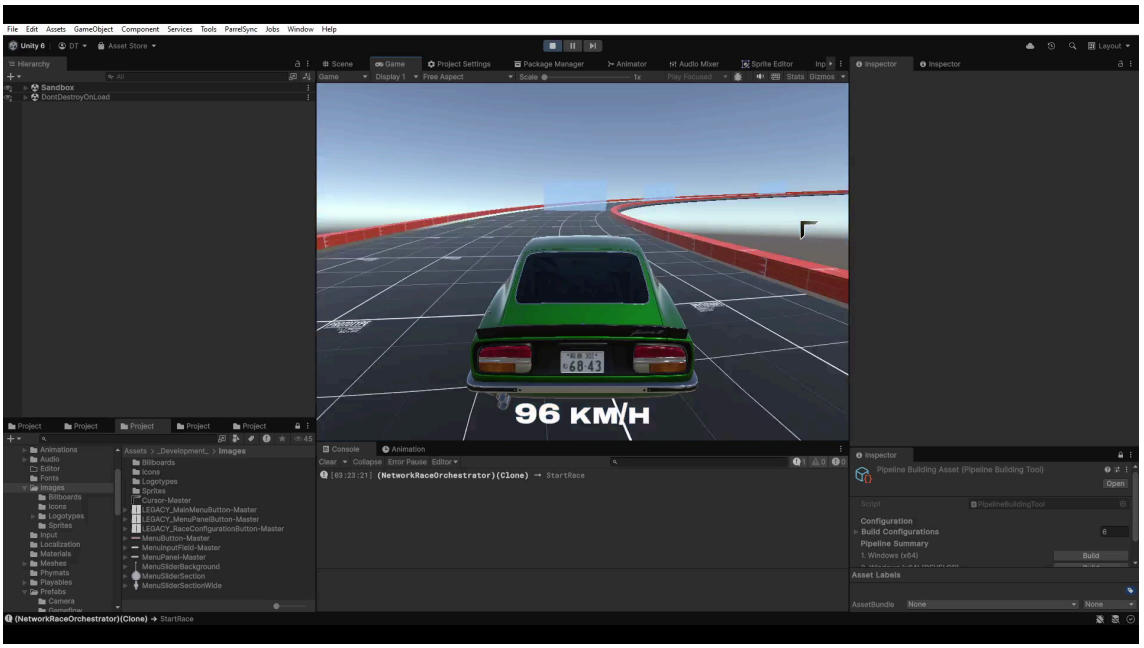


Figura 68. *[Devlog #70] Prueba de funcionamiento de un único vehículo con físicas en red*.
Vídeo: https://youtu.be/E693Ufy8qP4.

Como pasos finales, se ha asegurado un flujo de juego completo para un jugador, garantizando que todas las transiciones entre menús y escenas funcionen correctamente.

Es en este momento donde se realizan todas las comprobaciones necesarias para asegurar que el jugador pueda manejar su vehículo con soltura dentro del circuito, así como que evitar salirse accidentalmente del trazado.

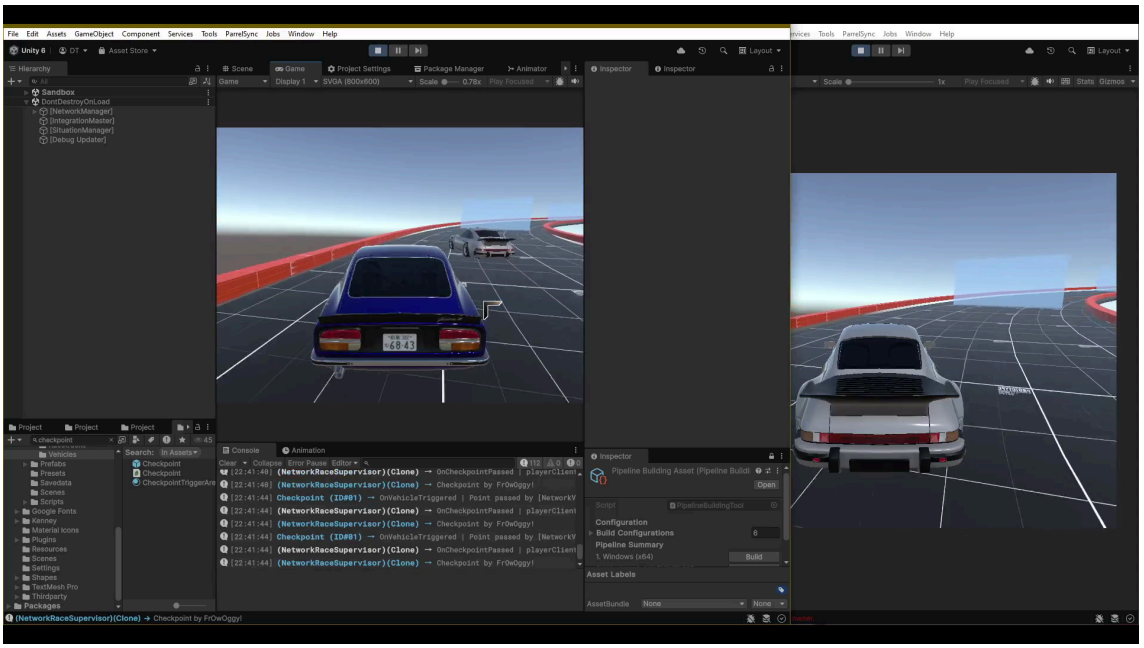


Figura 69. *[Devlog #71] Prueba de funcionamiento completa del prototipo de Protovolkers*.
Vídeo: https://youtu.be/5yBK-hiNa3U.

Con ello funcionando, y tan pronto como dos jugadores tienen la posibilidad de echar una carrera multijugador, se da por finalizado el prototipo culminando con la comprobación y depuración de los servicios multijugador de Unity combinados con su integración con Steam.

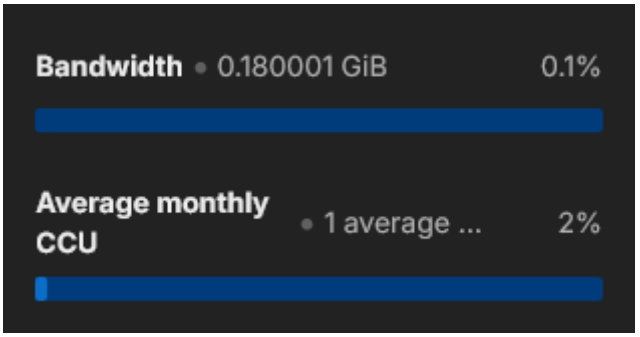


Figura 70. *Captura del resumen de consumo de ancho de banda para el proyecto Protovolkers en la web de Unity Cloud*.

Nótese que este prototipo utiliza los servicios de conectividad online ofrecidos por Unity. Debido a su funcionamiento razonablemente rápido y un precio competitivo, es bastante probable que sea considerada como la opción con la que el videojuego sea publicado.

3.3 Recursos requeridos

3.3.1 Hardware

A continuación se describe el equipo empleado durante el desarrollo.

Componente	**Modelo**
Procesador	AMD FX-6350 6-core @ 3.9GHz
Placa Base	Asus 970 PRO Gaming Aura
Memoria	4 × Kingston Fury 1866MHz 4GB CL10 (16 GB)
Tarjeta Gráfica	AMD Nitro Radeon RX 470 4GB
Disco Duro (Sistema)	Kingston UV500 SSD Sata 2280 240GB
Tarjeta de Sonido (Dedicada)	-
Mando	Xbox Wireless Controller (Xbox Series X)

Tabla 3. *Ficha técnica del equipo anterior a la actualización.*

Componente	**Modelo**
Procesador	Ryzen 9 7900 12-core @ 5.4 GHz
Placa Base	ASRock B650 PG Lightning
Memoria	2 × XPG Lancer RGB 6000MHz 16GB CL30 (32 GB)
Tarjeta Gráfica	AMD Nitro Radeon RX 470 4GB
Disco Duro (Sistema)	Western Digital Black SN770 SSD 1000GB
Tarjeta de Sonido (Dedicada)	Creative Sound Blaster Audigy Fx
Mando	Xbox Wireless Controller (Xbox Series X)

Tabla 4. *Ficha técnica del equipo posterior a la actualización.*

Nota

Debido a que los componentes han sido actualizados por motivos ajenos al proyecto, se detallan las fichas de las especificaciones, tanto anteriores como posteriores a la actualización.

3.3.2 Software

A continuación se encuentra la ficha de programas principales, así como los lenguajes de programación, y los servicios web empleados durante el desarrollo.

Herramienta	**Programa**
Sistema Operativo	Microsoft Windows 10 Enterprise LTSC (64-bits, versión 21H2)
Motor de Videojuego	Unity 6 (Versiones 6000.0.5f1, 6000.0.7f1, 6000.0.8f1, 6000.0.11f1, 6000.0.13f1, 6000.0.15f1, 6000.0.20f1, 6000.0.22f1, 6000.0.27f1 y 6000.0.41f1)
IDE de Programación	JetBrains Rider 2022.2.4 VSCodium 1.9x
Cliente Git	Fork 2.x
Editor de imágenes	Adobe Photoshop 2021 Krita 5.2
Editor de modelado 3D	Blender 4.x
Software de Documentación	LibreOffice 24.x Obsidian
Otro software	OBS Studio 31.x Steam SteamPipeGUI 1.4.0.2 Syncthing

Tabla 5. *Ficha del software local empleado*.

Uso	**Lenguaje de Programación**
Desarrollo General	C# 8.0
Herramientas y Automatización	Python 3.10 C# 8.0 Batch (PS 7.0) Bash (bash-4.x)

Tabla 6. *Ficha de los lenguajes de programación empleados*.

Herramienta	**Servicio**
Desarrollo	draw.io (diagrams.net) StackOverflow Freesound Steamworks Google Drive Google Spreadsheets

Repositorio	GitHub GitLab
Documentación	DuckDuckGo Notion Standard Notes ChatGPT Google Docs Scribbr
Store	Steam itch.io
Planificación	Notion
Marketing	ChatGPT Twitter / X Bluesky Mastodon Discord Instagram Namecheap SteamDB DeVuego

Tabla 7. *Ficha de los servicios de software empleados*.

3.4 Viabilidad e implementación

3.4.1 Unity Engine

What Unity plan am I eligible to use?

Unity Personal

Individuals, hobbyists, and small businesses using Unity to provide services to others are eligible to use Unity Personal if their respective clients, in the aggregate, have less than $200K USD of revenue or funds raised in the prior 12 months.

Individuals, hobbyists, and small businesses using Unity but not providing services to a third party are eligible to use Unity Personal if (i) for small businesses, if their aggregate gross revenue and funding are less than $200K USD, or (ii) for individuals and hobbyists if the amount generated in connection with their use of Unity is less than $200K USD.

Unity Student

Students enrolled in an accredited educational institution of legal age to consent to the collection and processing of their personal information (e.g., age 13 in the U.S., 16 in the E.U.) are eligible to use the free Unity Student plan.

Unity Pro and Enterprise

Effective January 1, 2025 upon plan purchase or renewal:

- Unity Pro will be required for businesses with revenue or funding greater than $200K USD in the last 12 months, and for those who do work with them.
- Unity Enterprise will be required for businesses with revenue or funding greater than $25 million USD in the last 12 months.

Even if you make less than these thresholds, you can purchase Unity Pro and Unity Enterprise subscriptions at any time to take advantage of the many additional features they offer. Unity Industry plan is required if you create applications outside of games or entertainment and your company's total finances exceed $1,000,000 USD. Unity Industry's features, add-ons, onboarding, and support options are tailored to your needs.

All plans are subject to Unity Terms of Service.

Figura 71. *Condiciones de los planes de Unity según la situación del desarrollador*. Fuente: Unity.

Desde siempre, Unity impone ciertas condiciones económicas a los proyectos que utilizan su motor en caso de que cumplan ciertas condiciones.

En concreto, la versión gratuita de Unity permite desarrollar y publicar videojuegos siempre que los ingresos generados durante los últimos 12 meses no superen los $200.000 (*Compare Unity Plans: Personal, Pro, Enterprise, Industry*, n.d.).

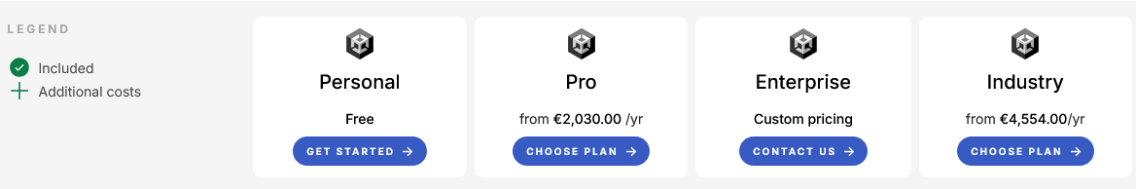


Figura 72. *Lista de precios de los planes ofrecidos por Unity*. Fuente: Unity.

En caso de superar esa cantidad, Unity exige adquirir una de sus licencias profesionales. Para la versión Unity Pro, el coste asciende a $2.030 por desarrollador al año (aproximadamente $170 al mes). También existe una versión Unity Enterprise para grandes estudios, con precios personalizados si superan la cantidad de $25M los últimos 12 meses.

Esto significa que, mientras Protovolkers no genere ingresos superiores a los $200.000 anuales, no incurre en costes directos por el uso del motor. Solo en caso de alcanzar cierto éxito económico sería necesaria una inversión proporcional al crecimiento del juego tras su lanzamiento.

3.4.2 Unity Cloud

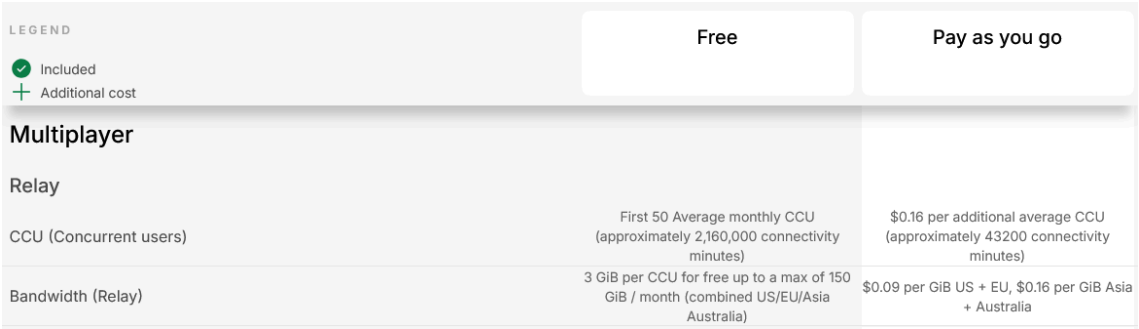


Figura 73. *Plan de precios para el servicio Relay, parte de Unity Cloud*. Fuente: Unity.

Para la conexión entre jugadores en línea, el prototipo de Protovolkers —y probablemente, el juego lanzado también— hace uso del servicio **Unity Relay**, incluido dentro de Unity Cloud.

Este servicio permite establecer partidas multijugador sin necesidad de configurar servidores dedicados, actuando como intermediario para las comunicaciones entre los clientes.

Actualmente, el plan gratuito permite hasta 50 jugadores concurrentes (CCU) y hasta 3 GiB de transferencia de datos al mes sin coste alguno, para todas las regiones combinadas (*UGS Pricing*, n.d.). En caso de superar estos límites, el coste pasa a ser el siguiente:

- $0,16 por cada jugador concurrente adicional.
- $0,09 por cada GiB adicional de datos en regiones de EE. UU. y Europa.
- $0,16 por cada GiB adicional de datos en Asia y Australia.

Estos límites son difícilmente superables durante el desarrollo del prototipo, y se consideran razonables tras el lanzamiento del videojuego. manteniendo unos gastos contenidos hasta que el proyecto escale en número de jugadores.

Nota

Además del uso de Relay, se plantea también incorporar el servicio **Unity Lobby**, combinable también dentro del ecosistema Unity Cloud, y el cual permite a los jugadores crear y unirse a salas globales previas al inicio de la partida.

También ofrece un plan gratuito, que permite hasta 10 GiB de transferencia al mes por grupo regional (dividido en EE. UU. + Europa, y Asia + Australia) (*UGS Pricing*, n.d.). En caso de superar ese umbral, los costes serían los siguientes:

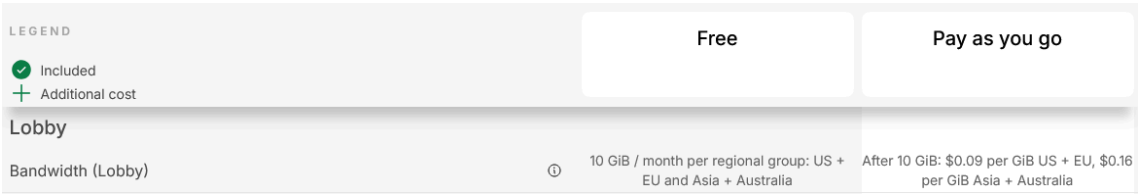
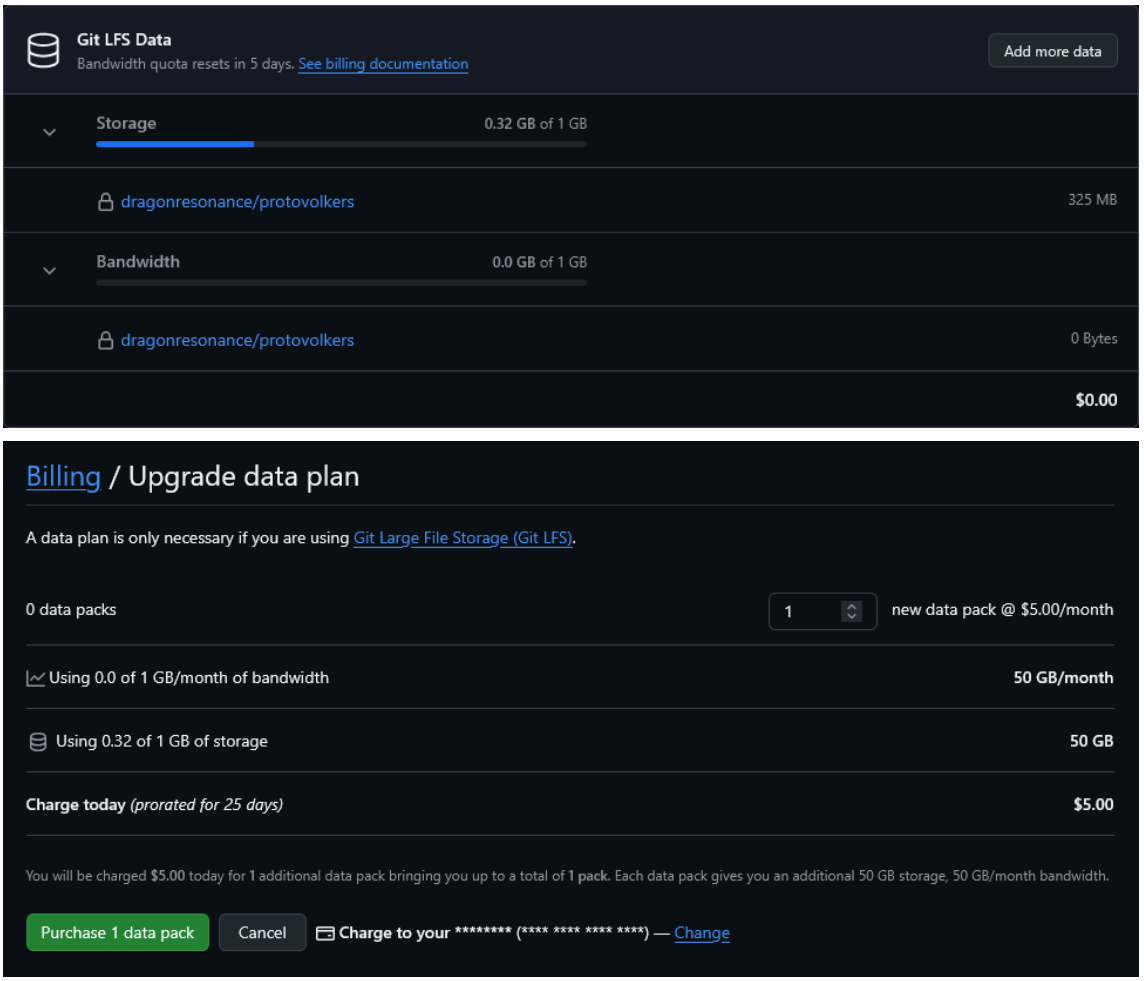


Figura 74. *Plan de precios para el servicio Lobby, parte de Unity Cloud*. Fuente: Unity.

- $0,09 por cada GiB adicional en el grupo regional de EE. UU. + Europa.
- $0,16 por cada GiB adicional en el grupo regional de Asia + Australia.

La decisión de usar este servicio se evaluará en función del progreso del desarrollo, el *feedback* de los jugadores y la necesidad de disponer de una infraestructura más organizada para partidas multijugador.

3.4.3 GitHub



Figuras 75 y 76. *Resumen del consumo de disco y ancho de banda por repositorio de GitHub —arriba— y gestor de paquetes de datos y precios —abajo—*. Fuente: GitHub.

Se utiliza GitHub para el control de versiones y la gestión del código fuente del proyecto como plataforma principal de alojamiento del repositorio Git. Aprovechando su plan gratuito, que permite hasta 1 GB de almacenamiento por repositorio sin coste.

No obstante, en caso de que el repositorio creciese más allá del límite gratuito, GitHub ofrece paquetes adicionales de almacenamiento, con un precio de $5 mensuales por cada 50 GB de espacio extra (*Acerca De La Facturación Para Large File Storage De Git*, n.d.), siendo una de las opciones más razonables y económicas en el mercado.

3.4.4 Steam (Steamworks)

Figuras 77 y 78. *Entrada dedicada al pago del Steam Direct Fee para la publicación de productos en Steam —arriba— y su devolución —abajo—*. Fuente: Steamworks.

Otra consideración importante es la publicación del videojuego en la plataforma Steam, el principal canal de distribución digital de videojuegos.

Para poder publicar un producto en Steam es necesario realizar un pago único inicial de $100 por cada entrada de producto que se desee lanzar. No obstante, este importe es reembolsado si el producto llegase a generar al menos $1.000 brutos en ventas o ingresos.

3.5 Resultados del proyecto y análisis

3.5.1 Analíticas

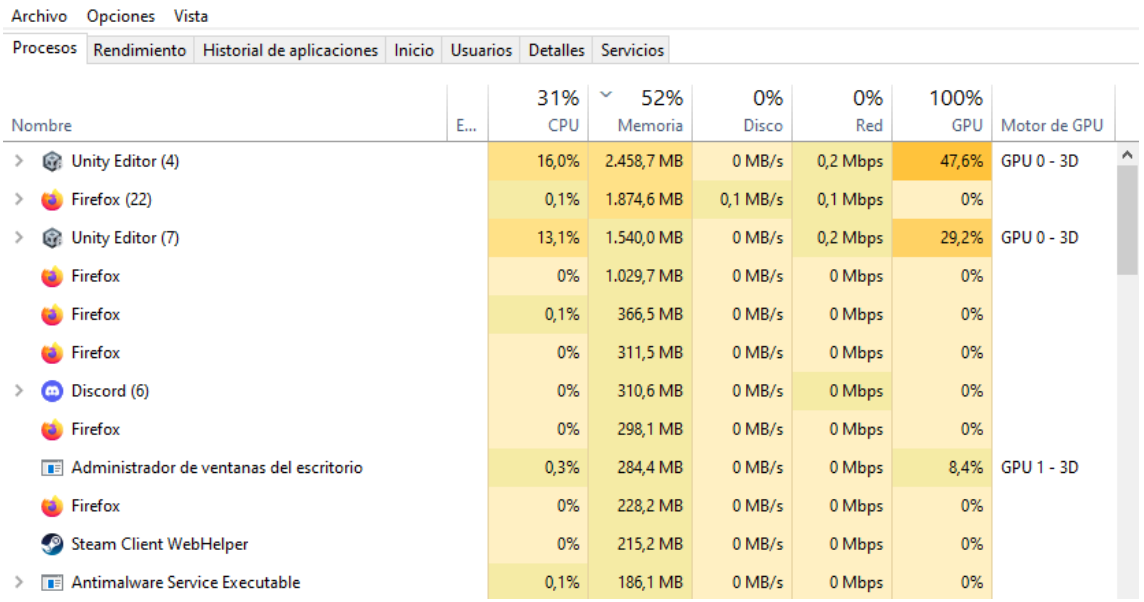
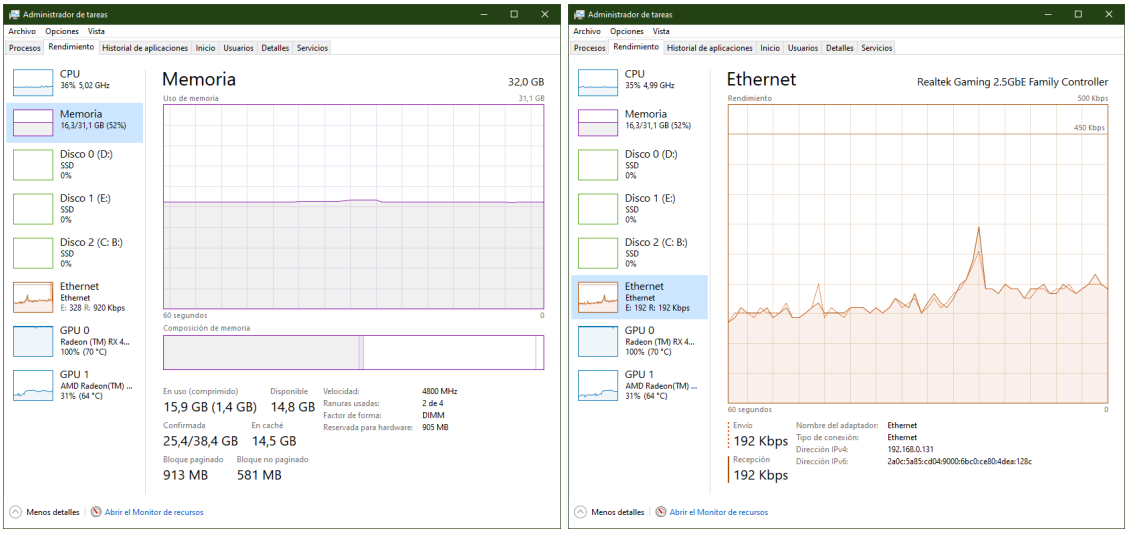


Figura 79. *Administrador de tareas de Windows mostrando los consumos de Unity (en Editor) durante una partida multijugador*. Fuente propia.

A lo largo del desarrollo del prototipo, y especialmente en sus fases finales, se ha prestado una atención cada vez mayor a los aspectos analíticos del rendimiento del sistema. En especial, se ha monitorizado de forma continuada el consumo de memoria RAM de manera constante, y el uso del ancho de banda en las sesiones multijugador.



Figuras 80 y 81. *Administrador de tareas de Windows mostrando el consumo de memoria —izquierda— y ancho de banda —derecha— de Unity (en Editor) durante una partida multijugador*. Fuente propia.

Para la obtención de los valores analíticos mencionados se han utilizado tres fuentes principales:

- El **Administrador de Tareas de Windows**, para medir el uso de procesador, memoria, GPU y red de forma general (Figuras 79, 80 y 81).
- El **Profiler de Unity**, para registrar con mayor detalle y precisión los consumos específicos del juego, tanto a nivel de CPU, GPU, RAM, y el consumo del sistema de red interno del motor (Figura 82).
- El **Dashboard de Unity Cloud**, que ofrece gráficas específicas sobre el uso de red a lo largo del tiempo en los servicios multijugador como Relay (Figuras 83, 84, 85 y 86).

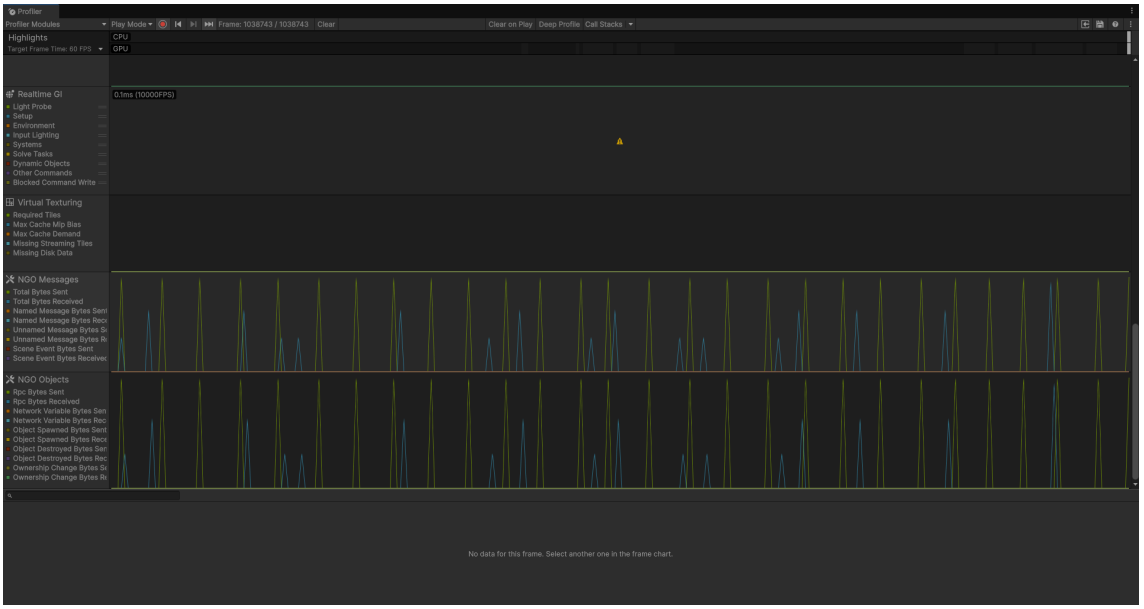
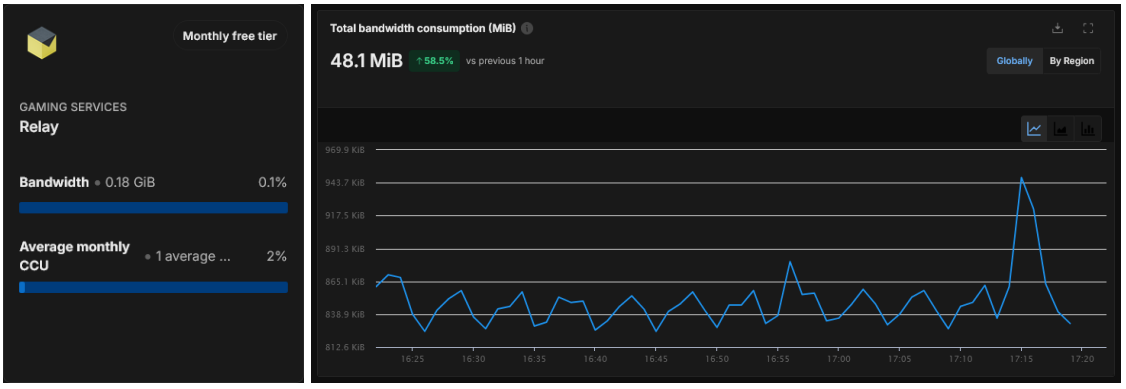
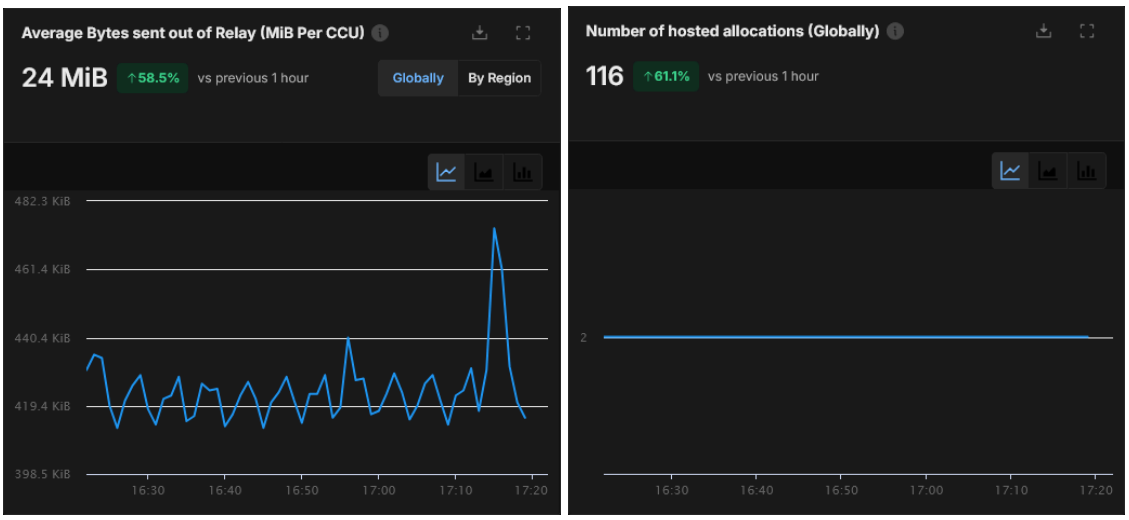


Figura 82. *Profiler de Unity durante la ejecución de una partida multijugador*. Fuente propia.



Figuras 83 y 84. *Contador —izquierda— y gráfica —derecha— de consumo de ancho de banda global a través del servicio Relay*. Fuente. Unity.



Figuras 85 y 86. *Gráficas de consumo de ancho de banda por CCU —izquierda— y espacios asignados —derecha— a través del servicio Relay*. Fuente. Unity.

3.5.2 Comparativa

Es importante destacar que los datos recogidos corresponden a un prototipo funcional en fase de desarrollo, por lo que no deben considerarse como valores definitivos ni optimizados.

Sin embargo, estos valores sí resultan útiles como referencia para establecer comparaciones aproximadas con el consumo de otros juegos disponibles actualmente en el mercado, tal y como se muestran en la tabla siguiente:

Título de videojuego	**Consumo de red (MB/h)**
Destiny 2	300
Valorant	290
Counter-Strike: Global Offensive	250
Overwatch	135
Apex Legends	120
Dota 2	120
Warframe	115
Fortnite	100
Battlefield V	100
Smite	100
PlayerUnknown’s Battlegrounds (PUBG)	80

Team Fortress 2	80
Call of Duty Modern Warfare	80
Minecraft	80
Rainbow Six Siege	70
Fall Guys: Ultimate Knockout	70
Black Desert Online	60
GTA V Online	60
Guild Wars 2	50
League of Legends	50
Call of Duty Black Ops 4	40
Call of Duty WWII	40
World of Warcraft	40
Rocket League	40
Star Wars: The Old Republic	30
Heroes of the Storm	30
Monster Hunter: World	30
The Elder Scrolls Online	20
Final Fantasy XIV	20
Runescape	15
Gwent	5
Hearthstone	3

Tabla 8. *Tabla de consumos de ancho de banda de videojuegos online conocidos*. Fuente: GamingScan.

A pesar de todo ello, este prototipo consume aproximadamente unos 48 MB/h para una partida de dos jugadores concurrentes, tal y como se muestra en la Figura 83; 24 MB/h, si es por cada jugador.

Teniendo en cuenta que las carreras en Protovolkers serán, por lo general, mucho más breves —con vueltas de entre 2 y 4 minutos, y carreras completas que rara vez superarán los 8-10 minutos—, el resto del tiempo el consumo de datos no se mantiene constante, sino que se reduce considerablemente. Se puede estimar que el consumo medio se reduzca hasta unos 20 MB/h por jugador, situándose en una línea muy inferior a la de otros juegos multijugador competitivos como Rocket League, el cual es uno de los referentes de este proyecto.

Capítulo 4. CONCLUSIONES

4.1 Conclusiones del trabajo

El resultado final de este trabajo ha sido un prototipo funcional que cumple con el objetivo general planteado: diseñar y desarrollar un sistema de físicas en red integrado en un videojuego multijugador.

Durante el proyecto, se ha conseguido entender y aplicar los principios fundamentales tanto del desarrollo de videojuegos basados en físicas como del funcionamiento de los videojuegos en red en entornos no deterministas, así como cumplir los siguientes objetivos:

- Sobre los **objetivos generales**: se ha estudiado y comprendido el funcionamiento de videojuegos con mecánicas de físicas, se ha analizado el funcionamiento de videojuegos multijugador modernos, entendiendo algunas implementaciones según el género, se han investigado y evaluado las técnicas más conocidas para el desarrollo de físicas en red, y se ha planteado y construido un sistema funcional, recurriendo únicamente a las herramientas previstas (Unity y servicios relacionados).
- Sobre los **objetivos específicos**: se ha creado un sistema de físicas en red funcional basado en la sincronización de estados mediante una estructura de "*corpus*" y "*anima*", y se ha implementado un sistema de compensación de latencia basado en técnicas de predicción de movimiento y posición.
- Sobre los **objetivos opcionales**, se ha completado un prototipo de videojuego de carreras que incorpora el sistema de físicas en red desarrollado, haciéndolo viable.

Restando el objetivo de crear un sistema general portable a múltiples tipos de proyecto y género, es una de las tareas pendientes listadas en el apartado Futura líneas de trabajo.

4.2 Conclusiones personales

Desde un punto de vista personal, este proyecto ha resultado grandemente satisfactorio por múltiples razones y, aunque partía con experiencia previa gracias a mi participación en proyectos de gran tamaño, este trabajo me ha permitido asumir un control mucho más amplio sobre los distintos aspectos del desarrollo, especialmente en las partes troncales del sistema.

He tenido la oportunidad de poner en práctica nuevas metodologías de organización, tanto en la gestión del tiempo como en la creación y uso de diagramas de diseño aplicados a diferentes áreas de un videojuego.

Asimismo, he podido profundizar en el funcionamiento interno de Unity, especialmente en el estudio detallado de su motor de físicas integrado. Además, uno de los mayores aprendizajes ha sido el desarrollo de un proyecto multijugador en red en tiempo real, un campo hasta ahora inexplorado por mi parte y del que he salido profundamente satisfecho.

Capítulo 5. FUTURAS LÍNEAS DE TRABAJO

5.1 El producto

5.1.1 Generalización

Una de las principales líneas de trabajo a futuro es continuar el desarrollo del componente de físicas en red, con el objetivo de convertirlo en un producto de propósito general.

La idea es que este sistema no esté limitado únicamente a un tipo de juego específico, como el caso de los videojuegos de carreras, sino que pueda adaptarse y ser utilizado en proyectos de cualquier género que requieran físicas sincronizadas en red.

Para ello, será necesario hacer el sistema completamente agnóstico, desacoplando aún más sus dependencias, y asegurando que su integración sea sencilla y flexible sin sacrificar cierto grado de personalización.

5.1.2 Optimización

Otra línea de trabajo prevista es la optimización del propio componente de físicas en red.

Actualmente existen partes del código duplicadas y soluciones poco refinadas que, si bien en su momento se priorizó el correcto funcionamiento del sistema, han contribuido al empeoramiento de la calidad del código.

Además, se sospecha que hay un margen significativo de mejora en la eficiencia del uso de ancho de banda. Aunque el sistema ya presenta un consumo reducido tal y como se ha visto en el apartado de Analíticas, no se ha realizado hasta ahora un esfuerzo específico en minimizar este aspecto como se debiera, por lo que se plantea trabajar en técnicas de optimización como las vistas en el apartado Optimización del envío de estados que permitan reducir aún más la cantidad de datos transmitidos durante las partidas.

5.1.3 Distribución

Finalmente, se tiene la intención de integrar este componente de físicas en red como parte de uno de los sets de herramientas libres y gratuitos para desarrollos en Unity creados en nombre del estudio Dragon Resonance.

Estos sets están disponibles públicamente tanto en GitHub como en GitLab, y se distribuyen bajo la licencia Apache 2.0, lo que permiten —y permitirá— que el componente esté preparado para ser utilizado en proyectos en producción de manera accesible para toda la comunidad de desarrolladores.

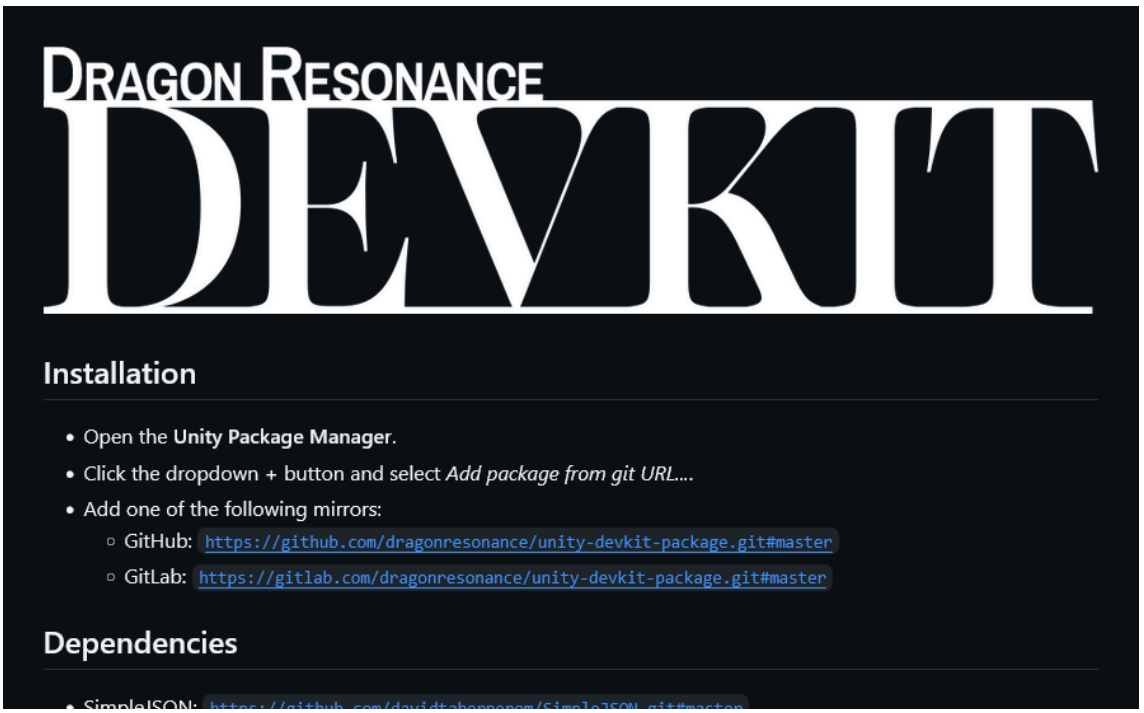


Figura 87. *Repositorio de Unity DevKit Package de Dragon Resonance*. Fuente: GitHub.

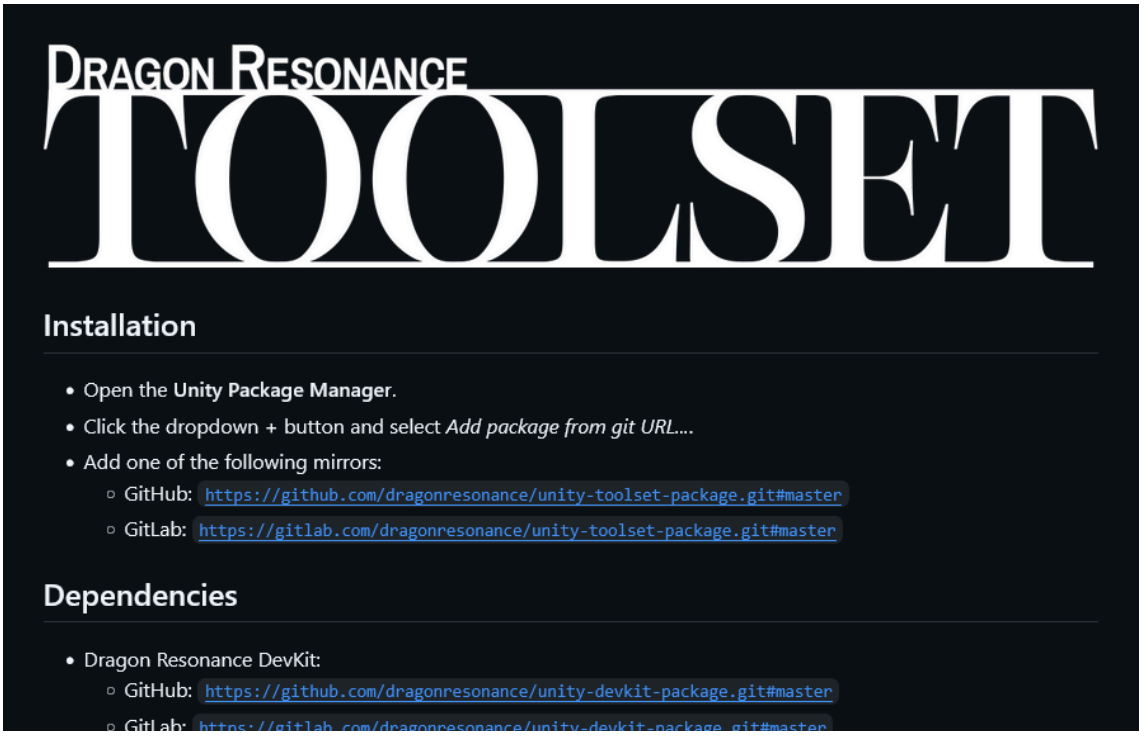


Figura 88. *Repositorio de Unity ToolSet Package de Dragon Resonance*. Fuente: GitHub.

5.2 El proyecto

5.2.1 Refactorización

De cara al futuro del proyecto Protovolkers está planificado que, una vez finalizado este Trabajo de Fin de Grado, se lleve a cabo una reestructuración completa del proyecto.

Esta reestructuración incluirá una nueva organización de las escenas y los menús, siguiendo el esquema previamente planteado en la Figura 51, así como la eliminación de código duplicado u obsoleto arrastrado de versiones anteriores.

Adicionalmente, se introducirán nuevas técnicas de simplificación de sistemas para mejorar su mantenimiento durante los meses de desarrollo por venir, así como se abordará una mejor gestión de la red, ya que actualmente algunas partes del flujo de juego y la gestión multijugador son algo caóticas y difíciles de trabajar a medio y largo plazo.

5.2.2 Finalización

Naturalmente, el objetivo es completar el desarrollo de Protovolkers.

Actualmente, aún existen varias mecánicas y partes de la lógica del juego que no han sido implementadas, así como sistemas existentes que requieren un nivel de gestión manual excesivo y que podrían ser automatizados para aligerar el flujo de trabajo.

Figura 89. *Set de iconos de Protovolkers para múltiples resoluciones de pantalla*.

Cabe destacar que toda la apariencia visual y sonora del proyecto se encuentra todavía en una fase preliminar: imágenes, modelos, sonidos, música, animaciones y transiciones no han sido definidos más allá de un reducido conjunto de bocetos conceptuales y una —extensa— colección de referencias externas.

5.2.3 Distribución

Finalmente, cabe destacar que Protovolkers aspira a convertirse en un videojuego completo y ser publicado en diversas tiendas digitales.

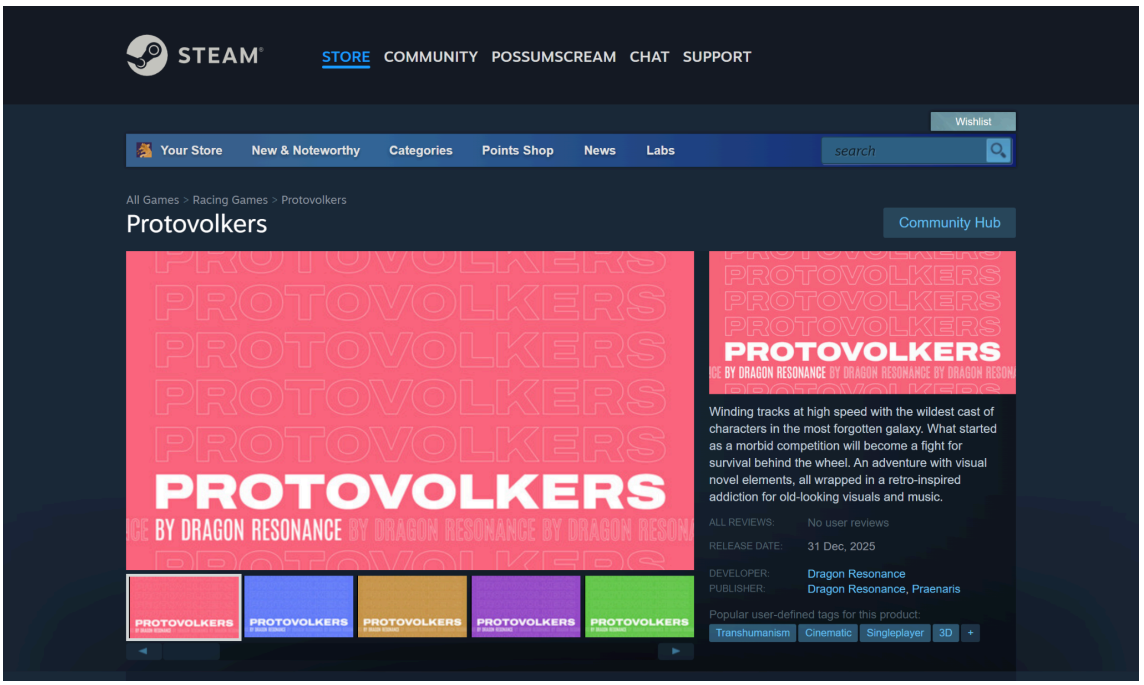


Figura 90. *Página de Protovolkers en Steam*. Fuente: Steam.

Actualmente, ya dispone de páginas reservadas tanto en Steam (Figura 90) como en itch.io (Figura 91), y no se descarta su expansión a otras plataformas y tiendas, dependiendo de la evolución del proyecto y las oportunidades que puedan surgir.

No obstante, antes de alcanzar estos objetivos comerciales, resulta fundamental completar el desarrollo del videojuego, para lo que se estima que aún serán necesarios varios meses de trabajo intensivo.

Sin embargo, tras más de tres años de avance continuo, el desarrollo del proyecto sigue siendo plenamente asumible y progresa de manera favorable y constante.

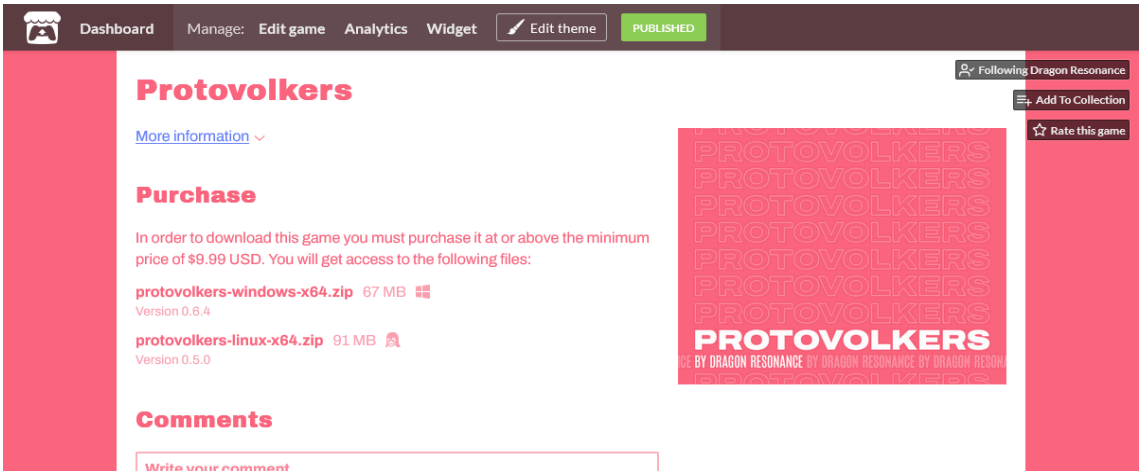


Figura 91. *Página de Protovolkers en itch.io*. Fuente: itch.io.

5.3 El estudio

5.3.1 Dragon Resonance

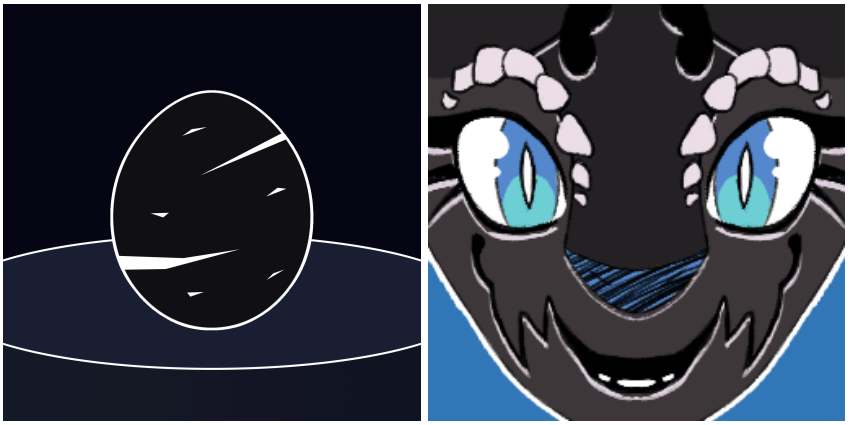
Figura 92. *Logotipo principal de Dragon Resonance*.

El estudio Dragon Resonance se fundó con el objetivo de intentar entrar en la industria de los videojuegos ofreciendo propuestas propias que destacan por su nivel artístico y técnico relativamente alto, manteniendo precios accesibles, y preservando siempre un espíritu indie que define su identidad.

Uno de los puntos que más lo caracteriza es su enfoque multi-género, creando nuevas propuestas únicas y diferentes a una amplia variedad de géneros dispares. La idea detrás de ello es sorprender siempre a la audiencia, creando una incertidumbre sobre cuál será el siguiente proyecto, tanto respecto a la historia, género, plataforma, o tecnología aplicada, por nombrar algunas.

5.3.2 Makoro

Makoro (en kanji, 真心) es la figura pública y representativa del estudio Dragon Resonance. En la narrativa interna del universo del estudio, Makoro es un personaje que se presenta como una criatura monstruosa creada en el laboratorio de *Dragon Resonance Experimental Software Systems* o *DRESS*, y que es la cara visible del proyecto y el motor detrás de su promoción.



Figuras 93 y 94. *Icono de Makoro anterior —izquierda— y posterior a su creación —derecha—.*

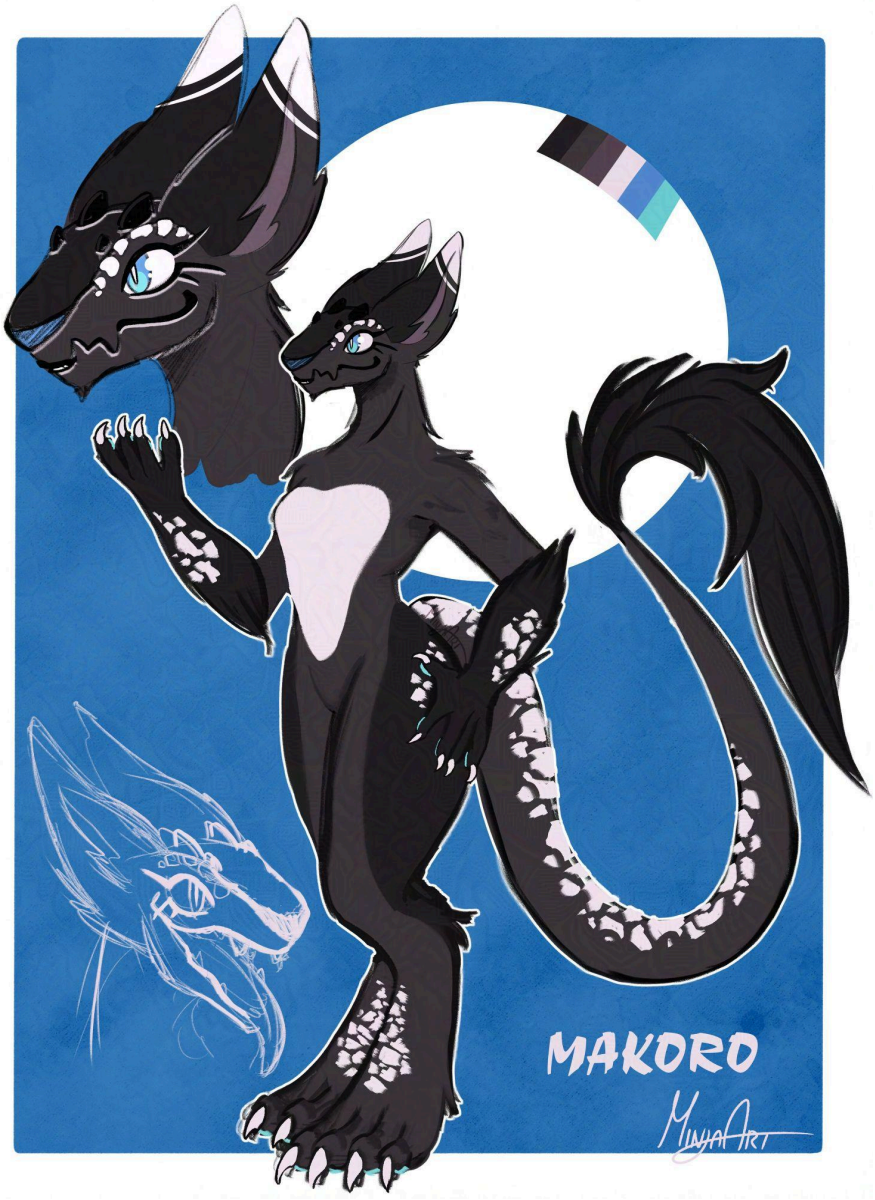


Figura 95. *Ficha del diseño de personaje de Makoro*. Artista: Minja.

Se muestra con una personalidad femenina carismática, cercana a la comunidad, e involucrada activamente en los avances y noticias del estudio. Es conocida por su interacción constante con los seguidores del estudio, compartiendo actualizaciones sobre el desarrollo de los proyectos, su pasión por los videojuegos, y su estilo único de comunicación, que a menudo incluye toques de humor, emoticonos, y un tono siempre correcto y cercano.

"Good night, my dear ones. 💕"

—Makoro

5.3.3 Presencia en internet

5.3.3.1 Página web principal

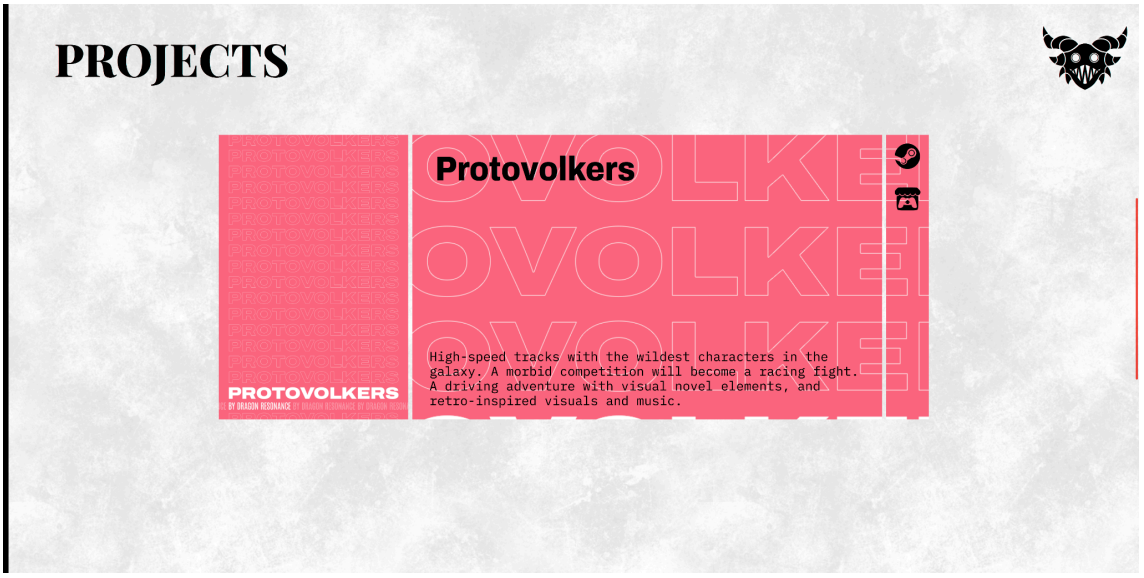
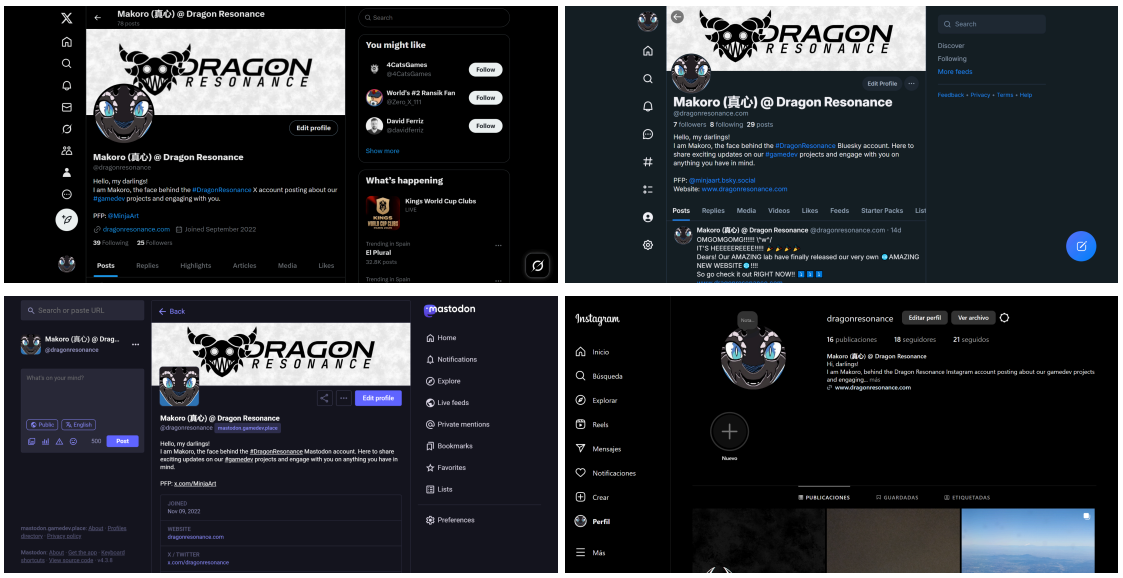


Figura 96. *Apartado de proyectos de la página web*. Fuente: Dragon Resonance.

Dragon Resonance dispone de una página web oficial en www.dragonresonance.com, la cual actúa como punto de referencia para el estudio.

En ella se recopila información general sobre su lore interno, sus proyectos en curso, sus miembros, así como todas las redes sociales y de desarrollo en las que opera.

5.3.3.2 Redes sociales como Makoro

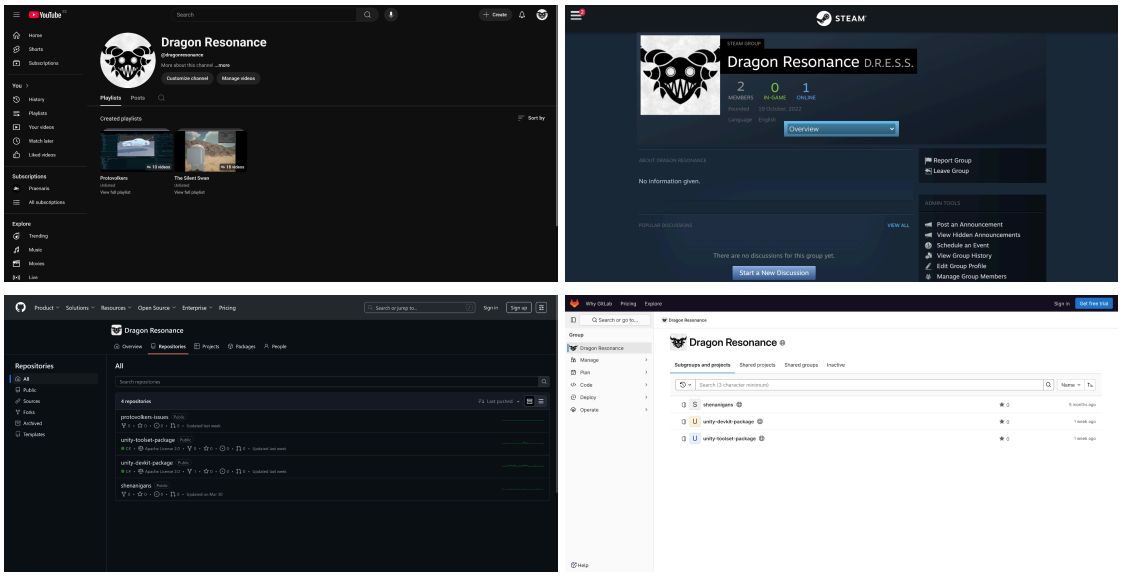


Figuras 97, 98, 99 y 100. *Presencia como Makoro en X —arriba izquierda—, Bluesky —arriba derecha—, Mastodon —abajo izquierda— e Instagram —abajo derecha—.*

La comunicación con el público se realiza principalmente a través de redes sociales como X (anteriormente Twitter), Instagram, Mastodon e Instagram, donde el estudio se expresa mediante su personaje e imagen pública: Makoro.

Esta figura actúa como portavoz informal del estudio, compartiendo avances de desarrollo, mensajes motivacionales y curiosidades con el tono amigable y reconocible descrito en el apartado anterior.

5.3.3.3 Redes sociales como estudio



Figuras 101, 102, 103 y 104. *Presencia como estudio en YouTube —arriba izquierda—, Steam Developers —arriba derecha—, GitHub —abajo izquierda— y GitLab —abajo derecha—.*

Adicionalmente, Dragon Resonance mantiene presencia en las plataformas de desarrollo y comunicación Discord, YouTube, Steam, itch.io, GitHub y GitLab como estudio de desarrollo.

Capítulo 6. BIBLIOGRAFÍA

Acerca de la facturación para Large File Storage de Git. (n.d.). GitHub Docs. https://docs.github.com/es/billing/managing-billing-for-your-products/managing-billing-for-git-large-file-storage/about-billing-for-git-large-file-storage

Acerca de la nueva plataforma de facturación. (n.d.). GitHub Docs. https://docs.github.com/es/billing/using-the-new-billing-platform/about-the-new-billing-platform

Ackerman Steering. (2012, January 6). DataGenetics. https://datagenetics.com/blog/december12016/index.html

Armitage, G., Claypool, M. & Branch, P. (2006). *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. https://theswissbay.ch/pdf/Gentoomen%20Library/Game%20Development/Programming/Networking%20and%20Online%20Games%20-%20Understanding%20and%20Engineering%20Multiplayer%20Internet%20Games.pdf

Berens, K., & Howard, G. (2008). *The rough guide to videogames*. Dorling Kindersley Ltd. https://archive.org/details/roughguidetovide0000bere

Bernier, Y. (2001). *Server In-game Protocol Design and Optimization*. Valve Developer Community. https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization

Brown, M. [Game Maker's Toolkit] (2023, February 17). *How To Think Like A Game Designer* [Video]. YouTube. https://www.youtube.com/watch?v=iIOIT3dCy5w

Burnhill, D. (2009). *The Ackerman steering principle*. RCTEK. https://web.archive.org/web/20160713113803/http://www.rctek.com/technical/handling/ackerman_steering_principle.html

ColourShedProductions. (2016, August 1). *Trackmania Nations Forever review - ColourShed* [Video]. YouTube. https://www.youtube.com/watch?v=On9BK5AMhrE

Compare Unity Plans: Personal, Pro, Enterprise, Industry. (n.d.). Unity. https://unity.com/products/compare-plans

Cone, J. [GDC] (2018, April 24). *It IS rocket science! The Physics of Rocket League detailed* [Video]. YouTube. https://www.youtube.com/watch?v=ueEmiDM94IE

D. Neumann, M., Dion, L., & Snapp, R. (2021). *Teaching Computational Thinking: An Integrative Approach for Middle and High School Learning*. MIT Press. https://direct.mit.edu/books/oa-monograph/5243/Teaching-Computational-ThinkingAn-Integrative

Donovan, T., & Garriott, R. (2010). *Replay: The History of Video Games*. Yellow Ant Media Limited. https://archive.org/details/replayhistoryofv0000dono/page/n5/mode/2up

Dragon Resonance. (n.d.). *Unity DevKit Package*. GitHub. https://github.com/dragonresonance/unity-devkit-package

Dragon Resonance. (n.d.). *Unity ToolSet Package*. GitHub. https://github.com/dragonresonance/unity-toolset-package

Fiedler, G. (2010). *Networking for Physics Programmers*. GameDevs.org. https://www.gamedevs.org/uploads/networking-for-physics-programmers.pdf

Fiedler, G. (2014, November 29). *Deterministic lockstep*. Gaffer on Games. https://gafferongames.com/post/deterministic_lockstep/

Fiedler, G. (2015, January 5). *State synchronization*. Gaffer on Games. https://gafferongames.com/post/state_synchronization/

Fiedler, G. (2015). *Physics for Game Programmers: Networking for Physics Programmers*. Game Developers Conference (GDC). https://gdcvault.com/play/1022195/Physics-for-Game-Programmers-Networking

Flashbulb Games. (2020, October 13). *What is Trailmakers?* [Video]. YouTube. https://www.youtube.com/watch?v=vD7Iz5J6OB4

Havok to join Microsoft. (2015, October 2). The Official Microsoft Blog. https://blogs.microsoft.com/blog/2015/10/02/havok-to-join-microsoft/

Hunicke, R., LeBlanc, M., & Zubek, R. (2004). *MDA: A Formal Approach to Game Design and Game Research*. https://www.researchgate.net/publication/228884866_MDA_A_Formal_Approach_to_Game_Design_and_Game_Research

Itterheim, S. [CodeSmile]. (2024, May). *Network Collision not working*. Unity Discussions. https://discussions.unity.com/t/network-collision-not-working/946523

Jolt Physics. (n.d.). https://jrouwe.github.io/JoltPhysics/#deterministic-simulation

Kent, S. L. (2001). *The ultimate history of video games: From Pong to Pokémon and Beyond : the Story Behind the Craze that Touched Our Lives and Changed the World*. https://archive.org/details/ultimatehistoryo0000kent

López, A., Moriano, A., & Álvarez, R. (2012, June 12). *El modelo de neumático de Pacejka: aproximación polinomial de la fórmula mágica*. Asociación Española De Ingeniería Mecánica. https://www.xixcnim.uji.es/CDActas/Documentos/ComunicacionesOrales/18-25.pdf

Luckerson, V. (2018, October 23). *Why 'The Legend of Zelda: Ocarina of Time' will always be the "Best game ever."* The Ringer. https://www.theringer.com/2018/10/23/video-games/legend-of-zelda-ocarina-time-best-game-ever-1998

Monster, M. (2003, November). *Car Physics for Games*. https://asawicki.info/Mirror/Car%20Physics%20for%20Games/Car%20Physics%20for%20Games.html

Muñoz, F. J. (2019, July 5). *Pokémon GO: así ha evolucionado el juego de Niantic en sus tres años de vida*. Vidaextra. https://www.vidaextra.com/analisis/pokemon-go-asi-ha-evolucionado-juego-niantic-sus-tres-anos-vida

Nobes, M. (2024, March 16). *Atari 2600: the console that launched a gaming revolution*. 80s Heaven. https://80sheaven.com/atari-2600-the-console-that-launched-a-gaming-revolution/

Padden, J. (2021, August 28). *What was the Great Video Game Crash of 1983?* The BugSplat Blog. https://blog.bugsplat.com/great-video-game-crash-1983/

Pomeyrol, J. (2025, March 4). *Godot 4.4: motor de juegos se actualiza con mejoras varias*. MuyLinux. https://www.muylinux.com/2025/03/04/godot-4-4/

Srisuchat, P. (2012). *Development of a car physics engine for games*. Bournemouth University. https://nccastaff.bournemouth.ac.uk/jmacey/MastersProject/MSc12/Srisuchat/Thesis.pdf

Steam Direct Fee. (n.d.). Steamworks documentation. https://partner.steamgames.com/doc/gettingstarted/appfee

Stewart, S. (2022, January 10). *How much data does online gaming use?* GamingScan. https://www.gamingscan.com/how-much-data-does-online-gaming-use/

Tester, E. (2023, December 17). *Cathode Ray Tube Amusement Device: the first electronic game*. CitizenSide. https://citizenside.com/technology/cathode-ray-tube-amusement-device-the-first-electronic-game/

The Review Den. (2023, June 6). *WIPEOUT PULSE - Last, best & final form* [Video]. YouTube. https://www.youtube.com/watch?v=9tT5K0ey62s

TrackMania Nations Forever Depots. (2022, September 21). SteamDB. https://steamdb.info/app/11020/depots/

UGS pricing. (n.d.). Unity. https://unity.com/products/gaming-services/pricing

Unity Cloud Compare plans & Pricing. (n.d.). Unity. https://unity.com/products/compare-plans/unity-cloud

Unreal Engine New Chaos Physics system screams with In-Depth Intel. (2019, June 11). Intel. https://www.intel.com/content/www/us/en/developer/articles/technical/unreal-engines-new-chaos-physics-system-screams-with-in-depth-intel-cpu-optimizations.html

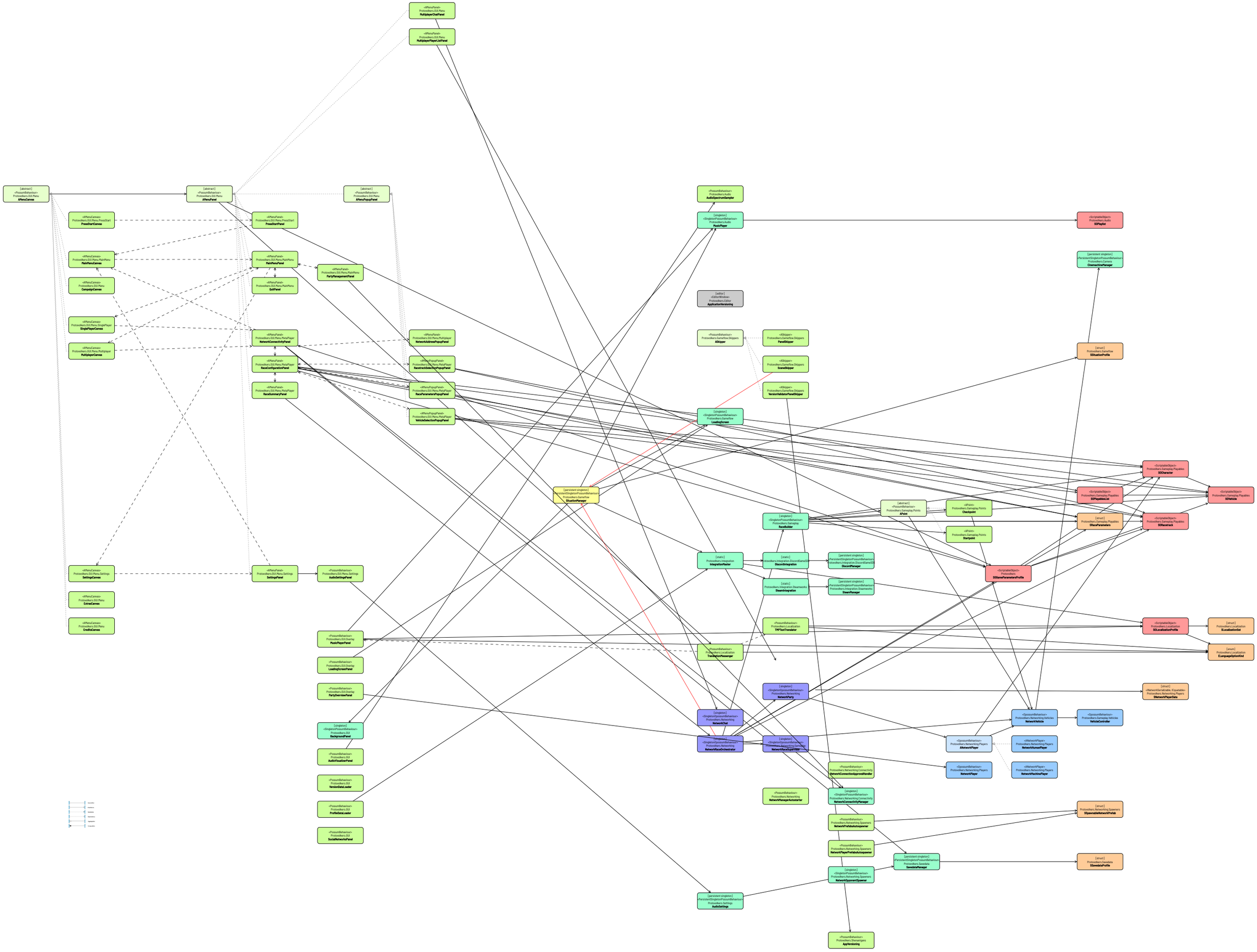
X-Engineer. (n.d.). *Vehicle acceleration and maximum speed modeling and simulation*. https://x-engineer.org/vehicle-acceleration-maximum-speed-modeling-simulation/

Capítulo 7. ANEXOS

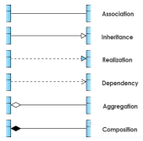
A continuación se presentan los documentos pertinentes mencionados a lo largo de este trabajo. Estos anexos incluyen material adicional, como trabajos ajenos, tablas de datos, código fuente, y cualquier otro recurso considerado relevante.

Índice de los anexos:

Anexo 1. Protovolkers Drawio Class Diagram (Legacy)..... A1
Anexo 2. Protovolkers Drawio Class Diagram (New)..... A2
Anexo 3. Protovolkers Drawio Scene Scheme..... A3
Anexo 4. Protovolkers Git Report..... A4
Anexo 5. Protovolkers Notion Tasks..... A15


«AMenuPanel»
ProtoWorkers.GUI.Menu
MultiplayerChatPanel
«AMenuPanel»
ProtoWorkers.GUI.Menu
MultiplayerPlayerListPanel
[abstract]
«PossumBehaviour»
ProtoWorkers.GUI.Menu
AMenuCanvas
[abstract]
«PossumBehaviour»
ProtoWorkers.GUI.Menu
AMenuPanel
[abstract]
«PossumBehaviour»
ProtoWorkers.GUI.Menu
AMenuPopupPanel
«AMenuCanvas»
ProtoWorkers.GUI.Menu.PressStart
PressStartCanvas
«AMenuPanel»
ProtoWorkers.GUI.Menu.PressStart
PressStartPanel
«AMenuCanvas»
ProtoWorkers.GUI.Menu.MainMenu
MainMenuCanvas
«AMenuPanel»
ProtoWorkers.GUI.Menu.MainMenu
MainMenuPanel
«AMenuPanel»
ProtoWorkers.GUI.Menu.MainMenu
PartyManagementPanel
«AMenuCanvas»
ProtoWorkers.GUI.Menu
CampaignCanvas
«AMenuPanel»
ProtoWorkers.GUI.Menu.MainMenu
QuitPanel
«AMenuCanvas»
ProtoWorkers.GUI.Menu.SinglePlayer
SinglePlayerCanvas
«AMenuCanvas»
ProtoWorkers.GUI.Menu.Multiplayer
MultiplayerCanvas
«AMenuPanel»
ProtoWorkers.GUI.Menu.MetaPlayer
NetworkConnectivityPanel
«AMenuPanel»
ProtoWorkers.GUI.Menu.MetaPlayer
RaceConfigurationPanel
«AMenuPanel»
ProtoWorkers.GUI.Menu.MetaPlayer
RaceSummaryPanel
«AMenuPanel»
ProtoWorkers.GUI.Menu.Multiplayer
NetworkAddressPopupPanel
«AMenuPopupPanel»
ProtoWorkers.GUI.Menu.MetaPlayer
RacetrackSelectionPopupPanel
«AMenuPopupPanel»
ProtoWorkers.GUI.Menu.MetaPlayer
RaceParameterPopupPanel
«AMenuPopupPanel»
ProtoWorkers.GUI.Menu.MetaPlayer
VehicleSelectionPopupPanel
«AMenuCanvas»
ProtoWorkers.GUI.Menu.Settings
SettingsCanvas
«AMenuPanel»
ProtoWorkers.GUI.Menu.Settings
SettingsPanel
«AMenuCanvas»
ProtoWorkers.GUI.Menu
ExtrasCanvas
«AMenuCanvas»
ProtoWorkers.GUI.Menu
CreditsCanvas
«PossumBehaviour»
ProtoWorkers.GUI.Menu.Settings
AudioSettingsPanel
«PossumBehaviour»
ProtoWorkers.GUI.Overlay
FluidPlayerPanel
«PossumBehaviour»
ProtoWorkers.GUI.Overlay
LoadingScreenPanel
«PossumBehaviour»
ProtoWorkers.GUI.Overlay
PartyOverviewPanel
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.GUI
BackgroundPanel
«PossumBehaviour»
ProtoWorkers.GUI
AudioVisualizerPanel
«PossumBehaviour»
ProtoWorkers.GUI
VersionDataLoader
«PossumBehaviour»
ProtoWorkers.GUI
ProfileDataLoader
«PossumBehaviour»
ProtoWorkers.GUI
SocialNetworksPanel
«PossumBehaviour»
ProtoWorkers.Audio
AudioSpectrumSampler
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Audio
MusicPlayer
[editor]
«EditorWindow»
ProtoWorkers.Editor
ApplicationVersioning
«PossumBehaviour»
ProtoWorkers.GameFlow.Skippers
AISkipper
«ASkipper»
ProtoWorkers.GameFlow.Skippers
PanelSkipper
«ASkipper»
ProtoWorkers.GameFlow.Skippers
SceneSkipper
«ASkipper»
ProtoWorkers.GameFlow.Skippers
VersionValidatorPanelSkipper
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.GameFlow
LoadingScreen
[persistent singleton]
«PersistentSingletonPossumBehaviour»
ProtoWorkers.GameFlow
SituationManager
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Gameplay
RaceBuilder
[abstract]
«PossumBehaviour»
ProtoWorkers.Gameplay.Points
APoint
«APoint»
ProtoWorkers.Gameplay.Points
Checkpoint
«APoint»
ProtoWorkers.Gameplay.Points
Startpoint
[static]
ProtoWorkers.Integration
IntegrationRuster
[static]
ProtoWorkers.Integration.DiscordGameSDK
DiscordIntegration
[persistent singleton]
«PersistentSingletonPossumBehaviour»
ProtoWorkers.Integration.DiscordGameSDK
DiscordManager
[static]
ProtoWorkers.Integration.Steamworks
SteamIntegration
[persistent singleton]
«PersistentSingletonPossumBehaviour»
ProtoWorkers.Integration.Steamworks
SteamManager
«PossumBehaviour»
ProtoWorkers.Localization
TMPTextTranslator
«PossumBehaviour»
ProtoWorkers.Localization
TranslationManager
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Networking
NetworkParty
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Networking
NetworkChat
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Networking
NetworkRaceOrchestrator
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Networking.Racing
NetworkRaceSupervisor
«PossumBehaviour»
ProtoWorkers.Networking.Connectivity
NetworkConnectionApprovalHandler
«PossumBehaviour»
ProtoWorkers.Networking
NetworkManagerAutostarter
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Networking.Connectivity
NetworkConnectivityManager
«PossumBehaviour»
ProtoWorkers.Networking.Spawners
NetworkPrefabsAutospawner
«PossumBehaviour»
ProtoWorkers.Networking.Spawners
NetworkPlayerPrefabsAutospawner
[singleton]
«SingletonPossumBehaviour»
ProtoWorkers.Networking.Spawners
NetworkOpponentSpawner
«PossumBehaviour»
ProtoWorkers.Shenanigans
AppVersioning
[persistent singleton]
«PersistentSingletonPossumBehaviour»
ProtoWorkers.Settings
AudioSettings
[persistent singleton]
«PersistentSingletonPossumBehaviour»
ProtoWorkers.Savedata
SavedataManager
«PossumBehaviour»
ProtoWorkers.Networking.Vehicles
NetworkVehicle
«PossumBehaviour»
ProtoWorkers.Gameplay.Vehicles
VehicleController
«PossumBehaviour»
ProtoWorkers.Networking.Players
ANetworkPlayer
«ANetworkPlayer»
ProtoWorkers.Networking.Players
NetworkHumanPlayer
«PossumBehaviour»
ProtoWorkers.Networking.Players
NetworkPlayer
«ANetworkPlayer»
ProtoWorkers.Networking.Players
NetworkMachinePlayer
«ScriptableObject»
ProtoWorkers.Audio
SOPlaylist
[persistent singleton]
«PersistentSingletonPossumBehaviour»
ProtoWorkers.Cameras
CinematicViewManager
[struct]
ProtoWorkers.GameFlow
SSituationProfile
«ScriptableObject»
ProtoWorkers.Gameplay.Playables
SOCharacter
«ScriptableObject»
ProtoWorkers.Gameplay.Playables
SOPlaylistList
«ScriptableObject»
ProtoWorkers.Gameplay.Playables
SOVehicle
[struct]
ProtoWorkers.Gameplay.Playables
SRaceParameters
«ScriptableObject»
ProtoWorkers.Gameplay.Playables
SORacetrack
«ScriptableObject»
ProtoWorkers
SOGameParametersProfile
«ScriptableObject»
ProtoWorkers.Localization
SOLocalizationProfile
[struct]
ProtoWorkers.Localization
SLocalizationSet
[enum]
ProtoWorkers.Localization
ELanguageOptionGUid
[struct]
«INetworkSerializable, IEquatable»
ProtoWorkers.Networking.Players
SNetworkPlayerData
[struct]
ProtoWorkers.Networking.Spawners
SSpawnableNetworkPrefab
[struct]
ProtoWorkers.Savedata
SSavedataProfile

PROTOVOLKERS Class Diagram



PTVK.Gameflow

PTVK.GUI

PTVK.GUI.Gameplay

PTVK.GUI.Overlay

PTVK.GUI.Menu

PTVK.GUI.Menu.MainMenu

PTVK.GUI.Menu.RacePreparation

PTVK.Gameplay

PTVK.Gameplay.Points

PTVK.Gameplay.Vehicles

PTVK.Gameplay.Vehicles.Aesthetics

PTVK.Gameplay.Vehicles.Mechanics

PTVK.Settings

PTVK.Settings.Audio

PTVK.Networking

PTVK.Networking.Connectivity

PTVK.Networking.Gameflow

PTVK.Networking.Gameplay

PTVK.Networking.Players

PTVK.Networking.Racetracks

PTVK.Networking.Vehicles

PTVK.Data

PTVK.Data.Gameflow

PTVK.Data.Playables

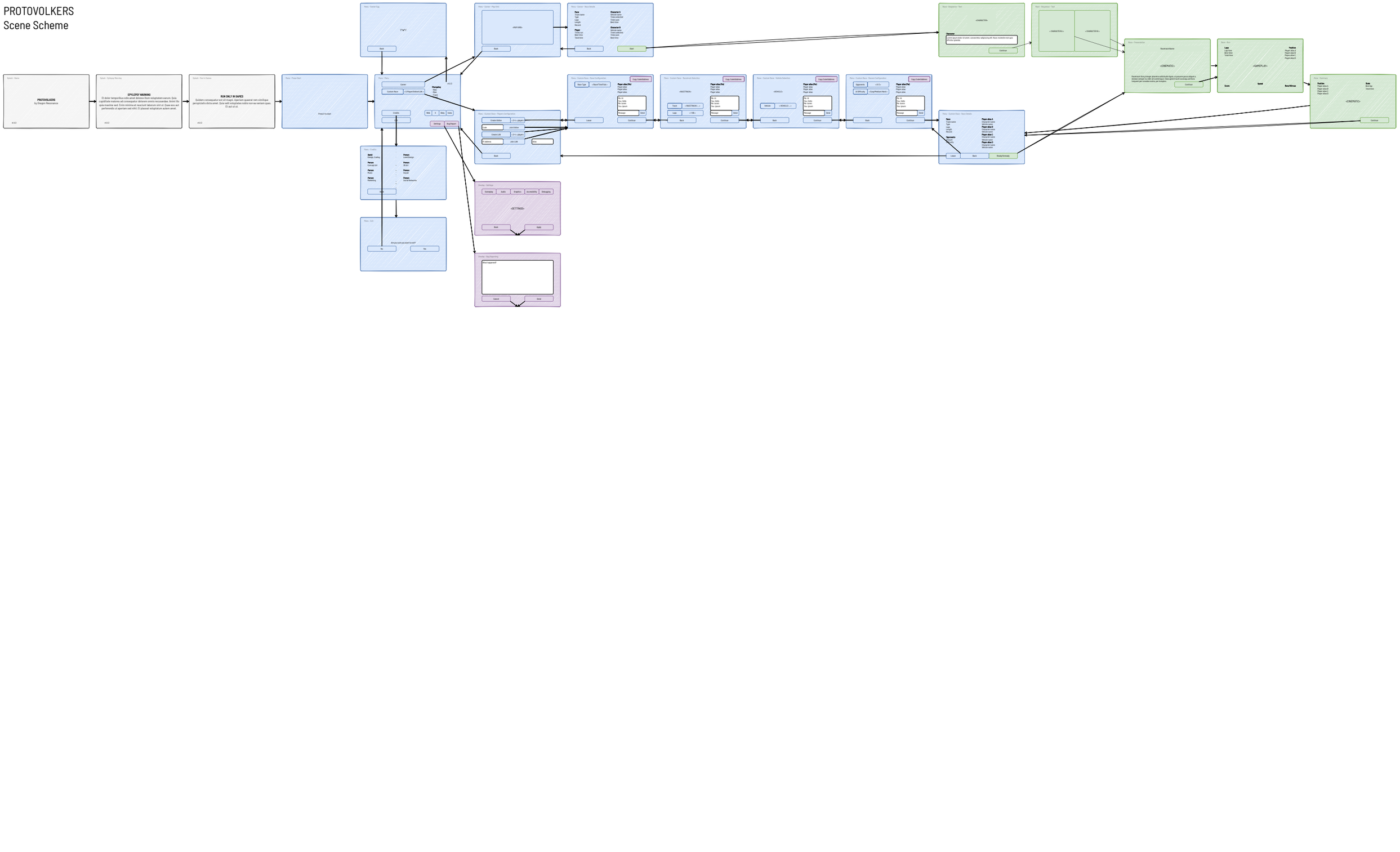
PTVK.Integration

PTVK.Integration.DiscordGameSDK

PTVK.Integration.Steamworks

PTVK.Shenanigans

PROTOVOLKERS
Scene Scheme



Id	Message	Datetime
22f9862	Initialize repository	2024-05-30T01:09:55+02:00
ade9f17	Create basic development folders	2024-05-30T02:20:22+02:00
6082054	Import the PossumScream repositories for future adaptation	2024-05-30T17:58:10+02:00
22ed769	Update the Contexter system, configure the project and update to the latest version	2024-05-30T19:42:38+02:00
a6676ff	Update to Unity 6.0.4, create test scene, create test particle emitter	2024-06-04T19:09:02+02:00
e08d90f	Update .gitignore and .contexter files, add PyCharm project into Automations folder	2024-06-07T02:45:30+02:00
2a6e6e9	Create better donots deletion script	2024-06-09T00:29:03+02:00
17df913	Create python script for itch.io auto-publishing, optimize Unity builds	2024-06-09T02:05:13+02:00
8fe9508	Fix some script namings	2024-06-09T12:10:21+02:00
19e758e	Import simple localization file	2024-06-09T13:37:54+02:00
e881fa6	Update project to Unity 6000.0.5	2024-06-13T20:23:33+02:00
f3797a0	Add startup checker against remote JSON config file, fix URP XR warning temporarily	2024-06-14T02:05:38+02:00
31107c7	Modularize startup checker and app config getter	2024-06-14T20:00:24+02:00
0f6460e	Create basic multiplayer scene and prefabs	2024-06-15T18:10:48+02:00
0d5ced5	Import the HierarchyFolders package	2024-06-15T18:11:14+02:00
1ff2eb3	Import the ParrelSync package, develop simple sphere player prefab with text an custom network transform	2024-06-16T01:21:09+02:00
27b9ca2	Develop client prediction in server	2024-06-16T02:16:33+02:00
56d10f3	Develop server prediction in client	2024-06-16T03:27:41+02:00
186d3c5	Change donot deletion script to analyze pipeline folders only, update playerbuild file configuration	2024-06-18T00:22:54+02:00
4f562d4	Develop automation script for Steamworks publishing	2024-06-19T03:36:09+02:00
fc87c7f	Create a fully working networking physics system	2024-06-20T02:48:15+02:00
7e74751	Optimize NetworkPlayer prefab, remove unneeded scripts	2024-06-21T17:53:11+02:00
45b2d8a	Create basic menu scene with basic panel scripts and navigation main-quit	2024-06-21T18:49:01+02:00
b0be296	Create the rest of basic panels and their scripts	2024-06-21T19:10:15+02:00
36fdc39	Make optional publishing for some platforms by skipping platform-specific key, add develop builds for Itch.io	2024-06-21T19:22:56+02:00
7186fb7	Build new version, setup pipeline file to work with develop builds only for now	2024-06-21T19:39:34+02:00
24e7365	Develop simple Mixamo animated bot with animator	2024-06-22T18:35:27+02:00
6c2ab1b	Rework pipeline and main menu panels	2024-06-22T18:48:02+02:00
d98912b	Create basic NetworkManager and NetworkChat singleton prefabs, create basic network chat, create panels for connectivity and chat	2024-06-23T03:18:45+02:00
855a40d	Upgrade project to Unity 6000.0.7, modify pipeline to work with 64-bit only, refactor main panels into canvases	2024-06-23T21:07:25+02:00
4b508d8	Reorganize panels, optimize GUI elements management, create custom network prefabs list	2024-06-24T05:08:35+02:00
66b480d	Fix and simplify NetworkChat system and panel, import custom font for button prefabs	2024-06-27T15:16:59+02:00
0ac3294	Create basic savedata abstract class and singleton subclasses	2024-06-27T18:59:53+02:00
f2128f4	Backup existing NetworkPlayer, create basic connection system, create basic player listing system	2024-06-30T04:23:35+02:00
3156bfc	Add refresh player list button	2024-06-30T04:30:57+02:00
b784d32	Setup custom cursor	2024-07-01T14:28:31+02:00
f8986ec	Optimize cursor settings to use the Photoshop file and create its preset	2024-07-01T20:01:10+02:00
15c1fb0	Fix network player list with the right hierarchy and network variables	2024-07-05T03:58:15+02:00
a96d821	Improve NetworkChat to recognize sender client ID, update ChatPanel to show the sender's nickname	2024-07-05T04:13:43+02:00
fabba49	Update project to Unity 6000.0.8f1	2024-07-06T01:06:09+02:00
92c0e67	Develop basic implementation of insta-translation	2024-07-07T13:16:51+02:00
ecb8a24	Customize menu button prefab sprite	2024-07-08T23:34:11+02:00
4e3503b	Develop savedata automatic backup system	2024-07-09T01:40:01+02:00
8484bfe	Setup custom build icon	2024-07-10T07:08:27+02:00
058fadf	Update DragonResonance DevKit package	2024-07-10T07:25:04+02:00
a27b9f7	Update project to Unity 6000.0.11f1, generate first Linux mono build	2024-07-19T18:58:25+02:00
f03b14a	Update DragonResonance packages, setup pipeline to work on Windows x64 only for now	2024-07-20T17:07:46+02:00
82eded3	Refactor the PossumScream directory into the DragonResonance directory	2024-07-20T17:17:28+02:00
661629d	Update Rider VCS paths	2024-07-20T17:21:29+02:00
365e771	Create missing canvas objects and their bindings, build for Windows	2024-07-21T00:15:03+02:00
2560455	Update ODVL table and add unavailable labels in menus	2024-07-23T17:08:53+02:00
9d57485	Refactor the three savedata systems into one	2024-07-24T02:01:10+02:00
9b77bc1	Add simple Protovolkers specific localization table file	2024-07-26T19:04:59+02:00
1da928a	Create MusicPlayer and SituationManager scripts and prefabs, create OverlayCanvas, refactor GUI scripts into Menu and Overlay child namespaces	2024-07-26T20:17:35+02:00
a7e3f0a	Fix MusicPlayer prefab have SituationManager script instead of MusicPlayer script	2024-07-26T20:19:57+02:00
f79ab8b	Import sample songs and sounds, develop basic playlist and player systems, create basic audio mixer, add music controls to input map, link music player to the music player panel, update and use newer DRDevKit methods	2024-07-27T03:21:29+02:00
edd052e	Add mirror repo links to the README file	2024-07-28T02:15:05+02:00

Id	Message	Datetime
7009c4b	Develop SituationManager to setup the music player to fit the scene	2024-07-31T02:12:11+02:00
b46d33c	Update HierarchyFolders package	2024-07-31T02:12:27+02:00
a5d4cd9	Create bare race organizer script and prefab, create bare single player panels	2024-07-31T05:56:19+02:00
7e2992e	Fix MusicPlayer singleton instance is sometimes not instanced in Editor when script tries to subscribe to its event	2024-07-31T05:57:09+02:00
1549bd8	Refactor multiplayer panels to indicate its canvas and use case	2024-07-31T05:59:25+02:00
1f0d299	Develop simple panel transition between SinglePlayer canvas panels	2024-07-31T06:23:06+02:00
e46616e	Interconnect single player panels to retrieve the race configuration, store the race config into a JSON file, fix savedata JSON indentation bug	2024-08-01T03:06:02+02:00
1477c99	Make delete-donots python script mark matches better and more visible	2024-08-01T03:06:35+02:00
a78f0a0	Fix TextTranslator accepts spaces before and after the key, update ODVL with more records, add labels for input fields	2024-08-01T03:21:14+02:00
9584d83	Refactor dependencies to have a dedicated script for Single Player race setup	2024-08-01T20:10:38+02:00
fdb057e	Update configuration for build	2024-08-02T03:06:51+02:00
8ff4dce	Refactor to use RaceOrganize for that purpose	2024-08-03T17:27:03+02:00
dcae803	Organize Race Preparer and Assembler, create LoadingCanvas, create Sandbox scene, update ODVL	2024-08-05T19:54:15+02:00
b56476b	Import and configure DOTween package, create basic loading screen	2024-08-05T23:54:30+02:00
d4517af	Merge overlay scripts to work in the same gameobject, add basic Steamworks integration and profile panel	2024-08-07T01:26:08+02:00
d60e4c7	Update project to Unity 6000.0.13f1	2024-08-08T23:13:06+02:00
b4e95dc	Import downloaded testing music, rework the SituatiuonManager to work on a per-situation basis, add empty Splash and Race scenes	2024-08-09T02:41:02+02:00
2f181ec	Upgrade playlist script to allow random unrepeated playlists	2024-08-09T03:42:49+02:00
e0468d5	Setup basic Steamworks Rich Presence configuration and parameters	2024-08-09T17:30:34+02:00
aa7c922	Prepare for new build	2024-08-10T00:07:19+02:00
211acb9	Reduce default background music volume, setup canvas prefabs in main scenes	2024-08-10T02:09:33+02:00
6e8c705	Reorganize single player canvas to allow NetworkManager to host	2024-08-10T20:44:59+02:00
7dda325	Setup apropiate NetworkConnectionApprovalHandler and integrate it with the NetworkManager	2024-08-10T20:45:58+02:00
1f0d6c2	Reorganize canvases and network connections, refactor the RaceParams and merge it with RaceOrganizer, make single player use NetworkManager, configure SituationManager to handle network scenes	2024-08-11T02:52:07+02:00
128d3a0	Import testing camera and analytics scripts, update packages	2024-08-12T22:36:48+02:00
1de32c4	Import and integrate the Analytics package with scene, panels and prefab, update the localization files	2024-08-13T00:59:47+02:00
9b3c1b3	Update build data	2024-08-13T01:12:38+02:00
d0d0c1e	Create build panel to show the current build version	2024-08-13T03:54:07+02:00
a5a5921	Refactor versioning system to work within the project, remove unused Contexter variables	2024-08-13T23:32:13+02:00
2f36bbc	Fix analytics sometimes crash the game on splash scene change	2024-08-14T22:50:20+02:00
066010b	Add analytics decline button	2024-08-14T23:57:42+02:00
649a21f	Update data for latest build	2024-08-15T00:41:55+02:00
9b24a34	Fix SteamManager is recreated every time the game runs	2024-08-15T22:40:59+02:00
7635447	Lock unused menu buttons, recover basic NetworkPlayer script, increase bg music volume, create checkpoint prefab and script, create AUTOBOT script for dev automation	2024-08-16T01:27:42+02:00
cebd36e	Create and assign default physics material	2024-08-16T02:02:42+02:00
349419d	Make RaceOrganizer a RaceParameters persistent singleton, create RaceManager singleton prefab	2024-08-16T02:12:54+02:00
a7c48ca	Prepare for build version 0.4.24081600	2024-08-16T02:23:47+02:00
1f16f4b	Add the network address to the situation config exchange file	2024-08-16T12:48:33+02:00
11ff746	Create basic opponent spawning inside the RaceAssembler script	2024-08-16T13:21:00+02:00
0f5d903	Make RaceParameters non-persistent, setup panels to configure network parameters, load config before assembling race	2024-08-16T15:01:11+02:00
83d8fb3	Import a vehicle mesh and prefab, refactor the NetworkPlayer into NetworkVehicle, add the ready field in the PlayerData struct	2024-08-17T10:49:45+02:00
721f5ec	Update the DevKit package	2024-08-17T10:50:00+02:00
68500ab	Refactor race gameplay script and prefabs into the Gameplay namespace	2024-08-17T11:26:07+02:00
5cae187	Remove the testing random name generation in the NetworkVehicle script	2024-08-17T11:26:38+02:00
8dbc42b	Make controllers disable eachothers on enabled	2024-08-17T11:33:24+02:00
ed12f89	Fix AUTOBOT script skips race configuration	2024-08-17T12:47:52+02:00
85039ef	Prepare basic throttle movement for testing vehicles	2024-08-17T17:47:43+02:00
0d8b3de	Fix application version for the latest build	2024-08-17T17:57:54+02:00
07e4cdb	Customize meshes color, tweak default physics material friction values, make soul mesh look like a ghost	2024-08-17T19:23:29+02:00
5df9442	Setup config for the latest build	2024-08-17T20:00:06+02:00
c473265	Update ODVL with more entries, change default key to SANDWICH	2024-08-20T08:31:20+02:00
7d1e15b	Make audio panel work and setup audio mixer volumes	2024-08-20T14:03:26+02:00
08a6500	Update project to Unity 6000.0.15f1	2024-08-20T18:51:33+02:00
cb16cf6	Change LoadConfig method with LoadData for savedata stuff, setup audio mixer, settings and manager scripts	2024-08-20T21:03:13+02:00
71cca3c	Create basic menus for multiplayer game	2024-08-21T19:49:13+02:00

Id	Message	Datetime
bb8319e	Set volume settings to the lowest at application startup	2024-08-21T19:52:35+02:00
446eb0c	Set default multiplayer address to localhost during development	2024-08-21T19:53:42+02:00
6c9f6e4	Set multiplayer opponents to zero by default during development	2024-08-21T19:57:02+02:00
29bce29	Add basic scene loaded event into the RaceAssembler	2024-08-21T21:48:14+02:00
409518c	Import Shapes package	2024-08-22T12:15:24+02:00
5abac2f	Regenerate and configure the Shapes package	2024-08-22T13:00:32+02:00
ae8e54b	Add the Mathfs package to the project	2024-08-25T17:31:10+02:00
856624c	Rework the application versionning to make it more natural for players	2024-08-25T18:14:11+02:00
15b1591	Recreate basic splash screen with basic presentation title, subtitle and version. Start v0.5 development	2024-08-25T19:40:12+02:00
748804f	Fix and update minor Shapes script	2024-08-27T10:50:18+02:00
f4b11f5	Update building pipeline and scenes to work with clean ones, create AudioVisualizer basic script	2024-08-27T17:35:14+02:00
6054c56	Refactor SteamHandler into SteamIntegration	2024-08-27T17:56:42+02:00
fc1f3d7	Add basic Discord Game SDK integration and connect it to the SituationManager	2024-08-27T20:24:00+02:00
056177f	Disable canvas scaling in splash screen	2024-08-27T20:30:11+02:00
82ab2db	Add SituationManager+integration prefabs to the Splash scene	2024-08-27T20:33:35+02:00
1260cdb	Reorganize Discord import, add downloaded Redline song to teh splash playlist, fix audio mixer volume problem	2024-08-29T16:26:36+02:00
676a877	Create splash screen skipper to jump to the next scene or unload in editor mode	2024-08-30T00:26:13+02:00
c76122c	Fix Discord integration throws error when the client is not running	2024-08-30T21:19:02+02:00
56429ea	Update both icon and cursor images	2024-09-01T17:15:06+02:00
317c8b6	Update splash screen appeareance	2024-09-01T17:30:27+02:00
0c68bc4	Refactor localization files, fix Discord integration field, create IntegrationMaster to centralize integration commands	2024-09-02T19:15:24+02:00
336f52d	Import new and better fonts	2024-09-02T19:15:35+02:00
7f83b2e	Refactor VersionPanel into VersionTextLoader	2024-09-02T19:26:34+02:00
288130	Fix splash screen version loader component	2024-09-02T19:28:50+02:00
b4623d4	Reorganize better input maps for future fix	2024-09-02T20:00:20+02:00
f376ffe	Disallow multiple AMenuCanvas component	2024-09-02T20:01:13+02:00
7c82482	Move analytics panel to the right folder	2024-09-02T20:01:46+02:00
3652383	Refactor SplashSkipper into a generic SceneSkipper, fix the splash screen SceneSkipper instance	2024-09-02T20:55:00+02:00
bfb0936	Rework AMenuCanvas and AMenuPanel for better activation and deactivation with coroutines	2024-09-02T21:09:52+02:00
a50d337	Replace SwitchCanvas methods with the optimized ones	2024-09-02T21:11:42+02:00
eacb37c	Create SDF assets for Orbitron font	2024-09-03T00:03:52+02:00
5ff8b6f	Create SDF assets for Antonio font	2024-09-03T00:22:42+02:00
cc3214a	Create SDF assets for Archivo Black font	2024-09-03T00:23:39+02:00
e616a39	Prepare splash screen with opacity, update localization assets, set AMenuPanel with default values on awake	2024-09-03T00:41:44+02:00
9f28204	Customize texts in press start panel	2024-09-03T00:42:01+02:00
65ccaa1	Fix localization system and file asset, finalize audio visualizer, customize press start canvas	2024-09-03T04:23:28+02:00
dd7ec3a	Fix spectrum bars spawn visibly	2024-09-03T04:31:41+02:00
bcc6907	Purge font assets cache	2024-09-03T04:32:53+02:00
e293b25	Purge font assets cache #2	2024-09-03T04:33:25+02:00
0e6e5dd	Develop full transition from splash screen to main menu scene	2024-09-03T23:17:30+02:00
e8eb76d	Import audio sweep audioclip to test spectrum visualizer	2024-09-04T02:12:38+02:00
e87c858	Make panel animations virtual to allow for overrides	2024-09-04T02:13:15+02:00
b37f1f1	Make simple multilayer animation for the press start screen	2024-09-04T02:13:54+02:00
7001508	Fix canvas and panel transitions from and to other similar elements	2024-09-06T02:45:39+02:00
cb0f8df	Refine subtitle animation	2024-09-06T02:47:40+02:00
bcb4cfe	Move input maps management to the SituationManager script	2024-09-06T03:14:57+02:00
2ac0a2f	Remove hybrid map from the input map settings	2024-09-06T03:16:01+02:00
5,91E+85	Move the Translate input action to the General action map, add the translation messenger to the first canvas too	2024-09-06T03:21:26+02:00
04a4ad6	Fix Discord integration disposing throws error when there is no instance	2024-09-06T03:32:37+02:00
f169eaf	Port menu canvas from previous menu scene	2024-09-06T03:33:07+02:00
9affc78	Update font alphas	2024-09-06T03:33:30+02:00
e93a513	Refactor main menu canvas, move Animator component to the canvas itself	2024-09-07T02:25:03+02:00
29bd981	Inport and configure japanese fonts as fallbacks, upgrade the localization system with Unity SystemLanguage enum, add third extra language field	2024-09-07T02:59:16+02:00
2245b2d	Update font atlases	2024-09-07T03:00:10+02:00
c3d89f4	Configure input system to allow up to three insta-translations	2024-09-07T18:45:24+02:00
67e1692	Disable translation info logging	2024-09-07T19:06:49+02:00
a98691c	Create BackgroundCanvas prefab with BackgroundPanel singleton ot handle background appeareance	2024-09-08T16:23:39+02:00
e39f38d	Import another downloaded song to test audio visualizer	2024-09-08T16:32:18+02:00

Id	Message	Datetime
618531f	Update font atlases	2024-09-08T17:13:33+02:00
b444682	Rework the splash screen with warning messages and better behaviour, unity and optimize SituationManager	2024-09-08T23:44:31+02:00
a67f36e	Fix some bugs in the base localization table	2024-09-09T00:16:40+02:00
86e8fac	Add skipper indicator to the GUI items in the splash screen	2024-09-09T21:00:04+02:00
e0c02ee	Make SituationManager change the background color	2024-09-11T00:12:59+02:00
1120489	Add the ability to disable the Graphic component on panel awake	2024-09-11T00:42:57+02:00
3e65e5b	Refactor MainMenu GUI elements into their folder and namespace	2024-09-11T01:04:58+02:00
040c52f	Update font atlases	2024-09-11T01:06:48+02:00
00f85d0	Fix canvas bug where sometimes is not activated inside a coroutine	2024-09-11T18:41:20+02:00
947ae47	Create main menu button and audio visualizer GUI prefabs	2024-09-12T18:33:08+02:00
0a788df	Refactor version and profile data loaders, port and upgrade the OverlayCanvas prefab, make a standalone prefab out of the audio visualizer	2024-09-12T20:15:09+02:00
d0addc0	Optimize the SavedataManager editor class	2024-09-12T20:15:37+02:00
ac0220a	Add skip in editor option to the situation manager, make the background canvas persistent, remove obsolete situation fields	2024-09-12T23:51:13+02:00
eca2d58	Add menu entries, refactor GUI prefabs for better naming convention, add more localized entries for menu	2024-09-13T01:16:34+02:00
54abe4b	Refactor AudioVisualizer into AudioSpectrumSampler	2024-09-13T19:55:48+02:00
b4d8c5c	Refactor the AnalyticsManagerPanel to inherit from the AMenuPanel	2024-09-13T20:45:07+02:00
cde0cf3	Remove unneeded OverlayCanvas script	2024-09-14T01:13:29+02:00
1f29b18	Customize main menu button prefab to fit to animatable content	2024-09-14T03:00:25+02:00
d40b27f	Create custom basic main menu button animator	2024-09-14T03:31:20+02:00
3ef69cd	Customize main menu button animation with colors and sizes	2024-09-14T03:59:39+02:00
eed3f4b	Fix main menu button animations	2024-09-14T20:46:44+02:00
c57238d	Add keyboard arrows to GUI navigation	2024-09-14T20:46:58+02:00
8186e0a	Reorganize MainMenu canvas to fit the 1280px width target	2024-09-14T21:05:03+02:00
6f2af16	Import the full Archivo font family	2024-09-14T21:09:09+02:00
3fea04b	Update localization file, customize main menu title and remove unneeded prefab variants	2024-09-14T21:44:45+02:00
2919fef	Create basic main menu foot button prefab	2024-09-14T23:58:43+02:00
532e1fa	Place report bug button in the correct place and size	2024-09-16T12:48:40+02:00
d5f2f5d	Fix SituationManager tries to access editor-only alias field	2024-09-16T13:17:40+02:00
d9e222e	Make localization system work with different kinds on sheets, add localization formatting	2024-09-16T23:54:47+02:00
9b2d3ef	Disable main title prefab info logging, fix a few menu scene values	2024-09-16T23:56:11+02:00
a732347	Update font atlases	2024-09-17T00:00:44+02:00
b80a3c1	Fix main menu button animations, customize main menu foot button animations	2024-09-17T18:48:01+02:00
74b45bd	Create basic social network button links autocreator	2024-09-17T22:54:30+02:00
e8ff1a6	Create social network button links with autocreator	2024-09-17T23:31:19+02:00
343579c	Add working links to the social buttons	2024-09-17T23:38:51+02:00
7d293a1	Allow to disable social buttons autocreation	2024-09-17T23:44:49+02:00
36c000a	Reduce localization logging, fix social network links do not work sometimes	2024-09-18T01:37:46+02:00
488af8d	Customize disabled buttons for the next update	2024-09-18T01:57:29+02:00
3f0a1c6	Add Orbitron-Regular japanese fallback, increase the blinking waiting time to 2 seconds	2024-09-18T02:07:11+02:00
2004b9c	Develop the quit panel and connect it to the main menu panel	2024-09-19T01:39:08+02:00
88b335b	Fix DevKit autofocus, fix normal state button animators, fix panels coroutines stop too early, reduce panels' transition time	2024-09-19T20:11:47+02:00
5a3b6b8	Fix translation system to send message also when panel is set active	2024-09-19T20:32:24+02:00
d0ddb57	Import initial sound and new menu song, sort playlists reset method	2024-09-20T00:19:53+02:00
fa86d51	Update project to Unity 6000.0.20f1	2024-09-20T12:00:17+02:00
63dc66c	Import the Makoro easter egg picture, create a logotype image preset	2024-09-20T14:46:22+02:00
f16104e	Reorganize animation files	2024-09-20T19:18:51+02:00
ebe6032	Prepare a fresh SettingsCanvas with new SettingsPanel	2024-09-20T19:38:31+02:00
51abd53	Update font atlases	2024-09-20T20:03:56+02:00
f1bb5a1	Create scriptabelobject scripts for characters, vehicles and racetracks	2024-09-21T21:07:30+02:00
edeb3f5	Create scriptableobject assets for the characters and tests for the racetrack and the vehicle	2024-09-21T21:08:08+02:00
a494bb8	Refactor character assets into their folder, add assets for the vehicles	2024-09-22T00:38:30+02:00
cd4a221	Fill the rest of the vehicle names	2024-09-22T13:39:14+02:00
c34f20b	Create more basic racetrack scriptableobject assets	2024-09-23T13:43:30+02:00
4,33E+85	Prepare both SavedataManager and AudioSettings to load and apply on awake	2024-09-24T17:27:05+02:00
07a4b2e	Move localization profile to the localization folder	2024-09-24T17:27:46+02:00
4b0c1a5	Fix logging warnings from DevKit package	2024-09-24T17:28:10+02:00
19db6fd	Make savedata returns a default file for each profile it cannot retrieve	2024-09-24T17:45:09+02:00
f4718dc	Create basic audio settings panel and assign the basic sliders	2024-09-24T18:24:18+02:00
d8bb671	Create basic settings panel, optimize some button animators, fix some button prefabs alignment	2024-09-25T16:43:35+02:00

Id	Message	Datetime
e6a5355	Fix social networks are not recreated on the main menu	2024-09-25T16:57:54+02:00
83d18fe	Customize settings GUI elements, fix RawImage components are not maskable	2024-09-25T18:46:02+02:00
ea69b43	Disable settings canvas by default again	2024-09-25T18:47:34+02:00
89e5bfd	Develop the audio settings to work	2024-09-25T21:51:22+02:00
1da0e05	Update font atlases	2024-09-25T21:54:18+02:00
b10a906	Fix compilation shenanigans, update Orbitron font atlas	2024-09-25T22:33:59+02:00
1ff518f	Set default audio mixer master volume setting to minimum	2024-09-27T23:48:39+02:00
56940ba	Update font atlases	2024-09-27T23:48:49+02:00
66bcd31	Remove completely the analytics package	2024-09-27T23:49:32+02:00
b6f8142	Fix social icons display weird in main menu	2024-09-27T23:49:58+02:00
260426d	Recover and update the version checker and validator and integrate with the skippers in Splash scene	2024-09-27T23:50:46+02:00
1763bd1	Disconnect project form Unity Cloud	2024-09-27T23:55:07+02:00
4ab8f7a	Fix settings panels are shown off center when playing on a 16:10 display	2024-09-28T00:00:25+02:00
cf439de	Update font atlases	2024-09-28T00:05:43+02:00
70ffb6d	Fix versioning script throws exception when resource cannot be fetched or node cannot be parsed	2024-09-28T16:32:36+02:00
65ab96b	Allow SituationManager to work without playlist assigned, change starting sound from music to sound	2024-09-29T00:42:50+02:00
cf41e1d	Import Barlow Condensed fonts, develop a way better music player panel with animations, reorganize GUI prefabs in folders like the scripts	2024-09-29T05:44:10+02:00
7,50E+285	Update font atlases	2024-09-29T05:46:54+02:00
e941fbd	Comment legacy methods in other canvases	2024-09-30T22:32:26+02:00
b12be1c	Reduce music player marquees	2024-09-30T22:35:56+02:00
a59ee2f	Create the MultiplayerCanvas object and link it to the main menu	2024-09-30T23:21:14+02:00
723e865	Prepare both MultiplayerCanvas and SinglePlayerCanvas to be navigated from main menu	2024-10-02T22:13:50+02:00
4c3d163	Create the TrackSelectionPanel script and prefab	2024-10-03T20:56:38+02:00
43ac2ff	Create panels and prefabs for both single player and multiplayer GUI navigation	2024-10-03T23:58:51+02:00
b9fb69a	Add autogenerated VSCode files	2024-10-04T12:14:05+02:00
036198f	Create AMenuPanelPopup and make RaceConfiguration, TrackSelection and VehicleSelection	2024-10-05T00:52:02+02:00
eebaff2	Format Playlist scriptableobject script	2024-10-05T01:01:33+02:00
6107d60	Fix navigation panel navigation scripts	2024-10-05T01:29:39+02:00
ce42d91	Fix deactivation coroutine has no sense to have a parameter	2024-10-05T01:54:51+02:00
28d9616	Develop popup system to flow forward and backwards	2024-10-08T22:54:34+02:00
60bcb71	Add custom editor for AMenuPanelPopup children	2024-10-08T22:59:16+02:00
7896d59	Refactor full single player and multiplayer canvases and most panels, create network address popup panel, create connectivity GUI prefab	2024-10-10T00:26:38+02:00
3d91b1d	Make go back button in connectivity panel return to main menu canvas	2024-10-10T00:29:31+02:00
1e512aa	Customize connectivity panel for billboards and text, import basic billboard images, fix localization keys for taglines and descriptions	2024-10-10T01:40:27+02:00
69b1d71	Prepare project for next build	2024-10-10T02:18:43+02:00
3f311c5	Update project to Unity 6000.0.22f1, fix graphics API crash on some builds	2024-10-10T02:56:17+02:00
1d9390d	Update teh Contexter database	2024-10-10T03:19:21+02:00
7491555	Upgrade cursor for better visibility	2024-10-11T12:51:35+02:00
f562520	Update cursor preset, delete legacy cursor	2024-10-11T12:52:51+02:00
96a1d98	Add the full network address popup flow	2024-10-11T22:43:17+02:00
c1bbddf	Fix graphics APIs listings	2024-10-11T22:43:40+02:00
ae38ffa	Update font atlases	2024-10-11T22:44:08+02:00
626ac56	Organize a few lines in NetworkAddressPopupPanel script	2024-10-11T23:56:26+02:00
81c009b	Configure network connectivity and server start	2024-10-13T00:09:58+02:00
744ccbf	Rework networking panels, develop navigation up to the race configuration panel	2024-10-13T01:14:09+02:00
3f195b9	Merge both OnHostGameButtonClicked methods for just a parameterized one	2024-10-13T01:20:03+02:00
a812148	Update billboards to 4K	2024-10-13T16:46:51+02:00
4b8083d	Add components to the RaceConfigurationPanel	2024-10-13T17:09:22+02:00
eb0e68f	Optimize button assets, prepare race configuration button prefab and assets	2024-10-15T20:35:05+02:00
48a0b25	Customize race configuration buttons	2024-10-15T23:37:41+02:00
0f5edee	Refactor older GUI scripts and mark them as legacy	2024-10-16T02:00:25+02:00
23efa59	Configure race configuration navigation, delete legacy GUI scripts, configure and optimize AUTOBOT for the latest setup	2024-10-16T03:34:18+02:00
64c8198	Update Autobot to run up to RaceConfigurationPanel	2024-10-16T03:37:32+02:00
5be0b1d	Update font atlases	2024-10-16T03:37:41+02:00
f3b14d4	Create PlayablesList scriptableobject for easy playables access	2024-10-17T19:22:46+02:00
3c3f2c0	Optimize TMPTextTranslator code for better readability	2024-10-17T19:23:12+02:00
a9266b4	Fix panel multiload	2024-10-18T00:04:04+02:00
7b05898	Remove obsolete methods in GUI navigation	2024-10-18T00:04:19+02:00
41b1d2b	Prepare race configuration popups base	2024-10-18T02:09:11+02:00

Id	Message	Datetime
d63d9c7	Fix stored network address is the host one, create three racetrack assets, optimize race parameters, make data be stored into the game file	2024-10-18T03:32:52+02:00
26522b7	Make AUTOBOT go up to the summary panel	2024-10-18T03:34:26+02:00
d1324b6	Fix savedata manager throws error when two saves are backup in the same second	2024-10-18T03:44:51+02:00
43a96a7	Update font atlases	2024-10-18T03:46:21+02:00
d7fc5cc	Update font atlases	2024-10-18T03:47:57+02:00
f7a7cf2	Make use of a separated struct for the race parameters, develop custom JSON node formatter	2024-10-19T00:46:38+02:00
22fd405	Add the basic structure to the summary panel	2024-10-19T01:36:34+02:00
73d5ac8	Create the most basic network player script	2024-10-19T02:01:26+02:00
a1b2fd4	Prepare panel for network connectivity	2024-10-19T03:04:48+02:00
f05167b	Hide info logs from savedata manager	2024-10-19T03:05:04+02:00
6009c4a	Fix ready and unready buttons	2024-10-19T22:34:58+02:00
c4d5ff9	Reorganize network managerprefab and its relative components	2024-10-20T00:34:34+02:00
16c728f	Hide aliases warning in editor and IDE	2024-10-20T00:49:39+02:00
3f69a8f	Prepare network autospawner, setup naming for spawned objects	2024-10-20T01:48:38+02:00
5791bc1	Refactor network components for better organization	2024-10-20T03:09:15+02:00
2b6e0a3	Make ConnectivityPanel work with the User savedata profile	2024-10-20T03:10:47+02:00
5283dd3	Create general parameters asset for everything to access to	2024-10-20T03:30:14+02:00
221d866	Replace default fields with the params profile	2024-10-20T04:27:27+02:00
6c97a97	Refactor the network prefabs and scripts	2024-10-22T19:58:26+02:00
ee57691	Prepare AUTOBOT to automate connection for host and clients	2024-10-22T20:38:31+02:00
8452287	Reorganize GUI so NetworkRaceManager get race values, prepare NetworkParty to successfully retrieve all players	2024-10-23T03:02:59+02:00
6d4d899	Fix main title logo font in both romani and japanese kanji	2024-10-23T12:10:34+02:00
b01c251	Update font atlases	2024-10-23T12:28:25+02:00
f55fc9f	Make players go to the main racing scene via networking	2024-10-28T17:09:32+01:00
5a10e5b	Make network objects auto-rename themselves	2024-10-28T18:34:13+01:00
ce0f53b	Delete the legacy RaceManager prefab and script	2024-10-28T19:09:26+01:00
d1f4985	Prepare Race scene and subscribe to scene management	2024-10-28T19:28:10+01:00
be08b56	Update NGO, MultiplayerTools, InputSystem, Cinemachine and RiderAPI packages	2024-10-28T19:30:34+01:00
e71076a	Fix NetworkParty bug when trying to get own unspawned player	2024-10-28T19:57:54+01:00
6289c6a	Create RaceBuilder script and prefab to build the race in the scene	2024-10-29T01:09:58+01:00
bdb79d1	Create testing playables of all kinds, prepare the RaceBuilder script and its custom Editor for racetrack developement	2024-10-29T02:48:12+01:00
d82352e	Fix autobot script to properly end in clients	2024-10-29T17:47:41+01:00
0138b2c	Develop checkpoint building and dismantling system, store full checkpoint points in the racetrack assets	2024-10-29T19:21:15+01:00
91441ae	Make race builer work automatically for every player	2024-10-29T20:00:27+01:00
bed20cb	Update font atlases	2024-10-29T20:15:47+01:00
ff96896	Refactor NetworkVehicle prefabs and scripts	2024-10-30T18:06:58+01:00
d2e6664	Fill racetrack scriptableobject assets with max drivers and startpoint points	2024-10-30T18:43:17+01:00
badf278	Develop generic point building system, create startingpoints for vehicles	2024-10-30T19:40:18+01:00
4046765	Rename test buggy prefab	2024-10-31T00:35:57+01:00
b535db6	Rework the race builder to spawn NetworkVehiclePrefab instances	2024-10-31T01:36:51+01:00
1495095	Prepare race builder to spawn network vehicle prefabs at their positions	2024-10-31T02:03:09+01:00
f558555	Separate the RaceBuilder script into multiple partial files	2024-10-31T02:14:02+01:00
9947bb2	Make the RaceBuilder.RemovePoints method delete any leftover too	2024-10-31T02:17:33+01:00
e9f9dc4	Adjust starting grid, make NetworkVehicle generate the mesh	2024-10-31T02:36:42+01:00
5068d08	Disable unused fields warning in NetworkConnectivityPanel	2024-10-31T02:37:01+01:00
c45d80a	Rework testing circuit, increase floor size, modify point prefab triggers	2024-10-31T03:17:03+01:00
5914ff0	Add 2 more points to the testing track	2024-10-31T03:28:46+01:00
ff5e878	Fix NetworkVehicle prefab is not in the network prefab list	2024-10-31T18:59:56+01:00
45758dd	Make AUTOBOT be able to wait until enough players are connected	2024-10-31T19:00:19+01:00
1492025	Develop the network vehicle prefab spawning sequence for each player	2024-10-31T20:42:02+01:00
d397d02	Make client be able to autostop when disconnected from server	2024-11-01T03:01:30+01:00
d11917f	Fix game parameters profile missing or duplicated in some AMenuPanel inherits	2024-11-02T16:59:11+01:00
89af0a6	Fix error throws in race preparation and network objects spawning	2024-11-03T03:22:00+01:00
06ceadb	Refactor RaceBuilder to generate props progressively	2024-11-03T03:50:30+01:00
b07879c	Develop further race setup sequence with player-vehicle assignment	2024-11-03T04:39:04+01:00
1ed9b34	Delete obsolete race assembler and parameters, refactor a new VehicleController	2024-11-04T18:46:30+01:00
0d3c055	Make input be sent from NetworkPlayer to NetworkVehicle to VehicleController	2024-11-04T21:10:12+01:00
9,26E+06	Speed up development assets to navigate faster	2024-11-05T02:16:32+01:00
0d6bd2c	Create basic local vehicle movement system	2024-11-05T02:57:16+01:00

Id	Message	Datetime
2,18E+99	Fix falling vehicles, create floor prefab, update font atlases, increase version to 0.5.5	2024-11-05T03:33:27+01:00
7c7c24e	Increase version to 0.5.6 so the version is changed during development	2024-11-05T03:36:51+01:00
a65007e	Setup simple Cinemachine system with cached cameras, setup vehicle camera focus	2024-11-06T22:47:48+01:00
0cc61a3	Fix camera do not follow the vehicle properly	2024-11-07T17:09:05+01:00
7cdd28f	Fix race scene builds before it should	2024-11-08T21:36:18+01:00
530aea7	Import Kenney fonts, refactor Google Fonts, create core font assets	2024-11-09T02:25:47+01:00
4c009cf	Refactor button fonts	2024-11-09T02:40:15+01:00
85a15a7	Recreate main title and subtitle font references	2024-11-09T03:40:28+01:00
3b1473f	Import the TinyCsvParser package	2024-11-09T03:57:45+01:00
c6f735a	Fix network objects are not properly cached	2024-11-09T18:22:03+01:00
7b9a442	Update font atlases	2024-11-09T18:23:16+01:00
4b570ff	Refactor NetworkRaceOrchestrator and main fonts	2024-11-09T18:44:45+01:00
05414b5	Create NetworkRaceSupervisor script and prefab, update NetworkPrafabList, setup vehicle camera further away	2024-11-09T19:30:12+01:00
60293fe	Adjust panel transition durations, update font atlases	2024-11-10T01:50:41+01:00
5a7365e	Synchronize vehicles and repositioning, prepare Orchestrator for preparation, prepare Supervisor for future development	2024-11-11T02:22:02+01:00
2a54a6e	Update the Shapes package to Unity 6 - compatible	2024-11-11T02:34:07+01:00
6c39b10	Fix client execution duplication	2024-11-11T02:55:34+01:00
05def1b	Adjust race preparation timings, make vehicles wait until all them are unlocked with the race start	2024-11-11T03:44:32+01:00
15f7100	Reduce race scene preparation time	2024-11-11T03:49:00+01:00
bac10c0	Temporarily replace the human player with the machine player	2024-11-12T18:25:57+01:00
cc7d962	Setup network players with the checkpoints queue	2024-11-12T19:15:15+01:00
480d319	Develop basic AI for vehicle control	2024-11-12T23:40:49+01:00
b2d9483	Fix point triggers, fix vehicle forces, fix checkpoints listing, make network vehicles work	2024-11-13T21:20:45+01:00
ce2bde2	Upgrade vehicle physics and controller to have a better use of the brake, tune machine player to throttle and brake accordingly, add new physmat with no friction	2024-11-17T02:17:54+01:00
e30ecec	Develop a very basic machine player controller	2024-11-17T20:17:11+01:00
c55e387	Update project to Unity 6000.0.27f1	2024-11-19T21:00:50+01:00
ecb0caf	Upgrade NGO and git packages	2024-11-21T17:03:16+01:00
cb4ae70	Update .git configuration files	2024-11-25T13:30:25+01:00
8547242	Backup everything to rework the networking part	2024-11-29T19:07:30+01:00
6c188e4	Prepare scene for lobby connection from scratch	2024-12-03T00:32:07+01:00
63d7ce4	Develop upgraded Direct+Relay network connectivity system	2024-12-04T03:20:00+01:00
20db208	Setup relay service for NGO	2024-12-04T23:10:39+01:00
2acad78	Reinforce the NetworkConnectivityManager for relay+direct reliable working	2024-12-05T01:35:11+01:00
a7f1b9b	Stylize the multiplayer lobby panel	2024-12-05T01:51:29+01:00
1ed8dce	Make networking system with Steamworks integration	2024-12-07T04:11:16+01:00
5e7c4ac	Rework panels to use simple versions of both scripts and assemblies	2024-12-08T18:35:32+01:00
7058e2c	Add main menu buttons for placeholder Sandbox and working Quit	2024-12-08T18:53:01+01:00
b5b9008	Resize main menu panel, update Protovolkers logotypes, insert logo in main menu	2024-12-08T19:01:56+01:00
48d3dc6	Party player list now updates whit the nickname change event, create multi integration tool with system nickname by default	2024-12-08T23:48:58+01:00
37ee889	Make multi integration tool to return steam name as nickname if possible	2024-12-09T00:02:03+01:00
cc7fa6c	Fix ApplicationHandle missing dependency, upgrade VersionDataLoader to VersionGetter with generic types, add changelog panel to main menu	2024-12-09T12:58:30+01:00
b6edb12	Build v0.6.1	2024-12-09T13:01:16+01:00
f123636	Separate menu canvas and main menu canvas	2024-12-09T18:23:47+01:00
1091047	Refactor both Vehicles and Characters for better dependency	2024-12-09T18:53:42+01:00
ef522bc	Fix GameParametersProfile with newer playables	2024-12-09T19:15:24+01:00
9c50f13	Prepare navigation for canvases and panels	2024-12-10T00:08:40+01:00
2160635	Develop navigation up to the race summary panel	2024-12-10T00:44:32+01:00
1e90938	Develop newer NetworkParty system	2024-12-11T00:07:22+01:00
6447394	Develop working network chat	2024-12-11T01:21:57+01:00
858724f	Build v0.6.2	2024-12-11T01:25:25+01:00
4baf256	Change TryGetNickname with GetNickname, as it will always return something	2024-12-11T19:26:27+01:00
7faa8ab	Create a separate method for generating single player data	2024-12-11T19:44:41+01:00
c6a786a	Develop race summary member list	2024-12-11T23:41:58+01:00
2bd81f9	Refactor VersionGetter into Miscellany	2024-12-11T23:46:48+01:00
390335f	Remove some legacy dependencies, setup menu starting active panel	2024-12-11T23:52:26+01:00
00bc6ba	Develop RacePreparer singleton to check for all ready players	2024-12-12T01:06:53+01:00
5f3064a	Develop basic SituationManager with scene loading	2024-12-13T01:52:35+01:00
956b91e	Update DevKit	2024-12-13T17:38:13+01:00

Id	Message	Datetime
8bc8b7a	Fix autospawner to log only spawning prefabs	2024-12-13T17:39:51+01:00
913a798	Fix party member items with client ID	2024-12-13T18:23:47+01:00
344d7af	Use newer overriden Awake in NetworkPartyV2	2024-12-13T20:19:00+01:00
a4e0cf4	Refactor legacy NetworkParty	2024-12-13T20:19:54+01:00
af30b54	Make NetworkParty a standalone persistent singleton network object	2024-12-13T23:10:53+01:00
2f8583a	Add client ID field in race summary	2024-12-13T23:27:26+01:00
5975a54	Replace the ready player status event with a network list event	2024-12-14T00:21:29+01:00
28f6509	Replace VersionGetter with the ToolSet component equivalent	2024-12-15T18:19:37+01:00
54f48cb	Add empty camera to Sandbox scene to test	2024-12-15T18:20:40+01:00
2c79c2d	Replace ready players local Dictionary with NetworkList	2024-12-15T18:24:35+01:00
22a4905	Fix ready players cleanup is tried by clients	2024-12-15T18:26:19+01:00
ac541ae	Disallow try to start race if not server	2024-12-15T18:32:10+01:00
72d197e	Import GLTF plugin	2024-12-15T20:38:15+01:00
6602954	Import and configure two testing vehicles as prefabs and variants	2024-12-15T20:38:47+01:00
6ea225f	Prepare newer NetworkVehicle prefab, link this with the playable vehicle assets	2024-12-16T00:44:55+01:00
b4e0825	Track GLB files as LFS	2024-12-16T00:49:20+01:00
094e926	Update .gitattributes date version	2024-12-16T00:51:19+01:00
53fd36e	Prepare vehicle selection GUI listing and controls	2024-12-16T01:30:47+01:00
0df3b42	Fix SituationAttribute file name	2024-12-16T23:46:06+01:00
913c560	Create a GameParameters singleton to replace the scriptabelobject to use the editor capabilities	2024-12-17T00:10:27+01:00
09bef4f	Optimize the GameParameters singleton and its attributes	2024-12-17T00:54:22+01:00
842afa3	Add camera object to downloaded vehicle prefabs	2024-12-17T01:06:10+01:00
ac3d303	Create TODO for SituationAttribute	2024-12-17T01:09:52+01:00
3366b76	Bind vehicle selection panels to the VehicleBuilder empty singleton	2024-12-17T01:39:49+01:00
08af886	Add Key parameter to playables	2024-12-17T01:54:32+01:00
48c3b24	Add methods to retrieve playables from GameParameters	2024-12-17T02:04:23+01:00
c53b8a0	Fix downloaded vehicle prefabs camera has AudioListener component	2024-12-17T02:15:18+01:00
64c05d7	Develop VehicleBuilder showcase method to spawn the chosen vehicle in the background	2024-12-17T02:15:52+01:00
888813d	Start VehicleBuilder development	2024-12-18T10:15:39+01:00
dc9dff8	Refactor game data model into own namespace	2024-12-18T10:23:04+01:00
fcf2aab	Refactor Vehicle panels to use the data directly	2024-12-20T17:32:29+01:00
82cba87	Fix network vehicle is not spawned properly	2024-12-20T18:57:29+01:00
4315dc8	Disable preview button to avoid confusion during development	2024-12-20T18:58:47+01:00
609b3fb	Fix previous spawned network vehicles stay in scene	2024-12-20T19:27:58+01:00
6dc54f2	Setup cameras in vehicles and menu	2024-12-22T02:04:49+01:00
9d377bc	Setup network vehicle prefab variants, bind vehicles to their profiles, fix vehicle builder	2024-12-22T02:53:15+01:00
b5ff41b	Develop camera system with auto focus on own vehicle	2024-12-22T03:15:31+01:00
1e51d2a	Develop basic input system so sends the inputs to the vehicle	2024-12-22T18:43:01+01:00
3135105	Add sun and floor to sandbox scene	2024-12-23T03:22:38+01:00
06588b8	Add basic vehicle movement physics	2024-12-23T03:53:23+01:00
557d985	Make vehicles spawn in different positions	2024-12-23T04:57:03+01:00
9e9b2e8	Fix network vehicles are not binded to network players	2024-12-23T18:49:05+01:00
7885ab9	Refactor the vehicles camera from each vehicle to the general prefab	2024-12-23T18:53:05+01:00
c169c47	Constrain the camera to the corpus to follow	2024-12-24T02:36:58+01:00
f25c765	Set FixedUpdate to 30 updates per second	2024-12-24T03:17:34+01:00
0c0d0b4	Add easy ping calculation method to the party script	2024-12-24T03:18:14+01:00
5660f11	Rework and refine network physics for the vehicle controller	2024-12-24T03:53:58+01:00
8a15eb5	Remove unused code from the new NetworkVehicleController	2024-12-24T04:26:48+01:00
26abcc8	Replace the Networking.Playables namespace with Networking.Vehicles as it is more specific	2024-12-27T02:47:17+01:00
24d2a55	Remove the legacy NetworkVehicle	2024-12-27T02:49:08+01:00
2fb3a6f	Mark VehicleController to rework with the real mechanics	2024-12-27T03:10:52+01:00
869f6e1	Rename the Playables folder in Networking prefabs with Vehicles, like the scripts	2024-12-27T03:12:59+01:00
1174b23	Remove broken NetworkVehicleTest prefab, create a denetworked variant of a NetworkVehicle prefab in Sandbox	2024-12-27T03:21:01+01:00
6c43bd1	Add basic vehicular engine and driver system scripts for further development	2024-12-27T22:57:11+01:00
ae1a10b	Update Vector extensions, add basic vehicle engine sound and controller script, update audio mixer to sound	2024-12-28T02:02:34+01:00
230187	Clean EngineSound code	2024-12-28T22:45:06+01:00
e956e2d	Develop basic engine operation	2024-12-29T00:01:37+01:00
88118ac	Tune up the engine curves	2024-12-29T03:11:27+01:00
f216117	Develop simple clutch and connect it to the driver and engine	2024-12-29T03:37:00+01:00
9d192eb	Refactor engine to use deceleration instead of friction	2024-12-29T04:33:32+01:00

Id	Message	Datetime
7b662a1	Add the differential base script and connect it to the clutch	2024-12-29T04:42:57+01:00
a1ee595	Make vehicle engine sound 3D spatial	2024-12-29T17:02:05+01:00
775b857	Develop basic wheel force scripts	2024-12-29T20:27:06+01:00
5,39E+236	Reorder vehicle prefab elements to fix parenting issues	2025-01-03T01:26:13+01:00
67b46f0	Fix Fairlady pivots	2025-01-03T01:35:17+01:00
18416c1	Refix the network vehicle prefab and the red Fairlady variants	2025-01-03T02:20:58+01:00
1df66e4	Create a box prefab, add a wall of boxes in the Sandbox scene	2025-01-03T02:46:45+01:00
b19713c	Merge the clutch into the engine script for easier management	2025-01-03T03:25:32+01:00
ba5a143	Set a Driver component icon in vehicel prefab	2025-01-03T03:43:03+01:00
d9b4497	Rework the engine acceleration and deceleration formulas for better feeling	2025-01-03T03:49:06+01:00
aa1b09b	Update Dragonres packages, rework engine behaviour to use the normalized operating load	2025-01-03T04:49:00+01:00
ad3078d	Refactor the wheel force script into the spin script with properties, add differential properties	2025-01-04T05:02:47+01:00
7a038fa	Increase box size for development	2025-01-05T17:42:07+01:00
16ced06	Prepare autobrake and engine adjustment	2025-01-05T18:50:05+01:00
10483ce	Remove unused code	2025-01-05T18:59:33+01:00
464bf98	Disable older controller movement	2025-01-06T02:37:40+01:00
fb83cb1	Create a steering script and bind it to the wheels	2025-01-06T02:59:49+01:00
ad28b34	Add OnValidate to the engine sound script	2025-01-06T03:07:32+01:00
5805abe	Develop the simple cosmetic wheel rotation script	2025-01-06T04:49:36+01:00
df30a9a	Fix vehicle Ackermann steering	2025-01-09T01:04:09+01:00
df00abf	Increase the default turning radius	2025-01-09T18:47:20+01:00
3568aef	Prepare editor movement override, create base for wheel grip and position	2025-01-10T01:44:08+01:00
08bdd53	Add simple velocimeter	2025-01-10T21:01:17+01:00
aa6b7d6	Develop simple steering mechanic	2025-01-11T02:39:20+01:00
b291f17	Port the legacy suspension system to the new vehicle controller system	2025-01-11T18:20:20+01:00
a7c02c1	Prepare wheels position script	2025-01-11T18:31:56+01:00
20950ba	Fix VehicleBuggy mesh	2025-01-12T00:02:58+01:00
115ea8f	Fix clutch automation	2025-01-12T16:17:13+01:00
dc0c7b5	Fix the base network vehicle prefab	2025-01-12T18:31:26+01:00
3f4be51	Rework the network vehicles to use the same parent prefab	2025-01-12T20:35:51+01:00
0d9bc77	Fix vehicle steering	2025-01-13T10:47:41+01:00
3874caa	Make vehicle base faster by default	2025-01-13T10:54:05+01:00
8d3a2a5	Remove unused code from the newer network vehicle controller	2025-01-13T15:14:49+01:00
ac47559	Prepare basic network racetrack builder prefab and base script	2025-01-14T01:39:27+01:00
d561482	Update racetrack navigation	2025-01-15T00:33:37+01:00
196c1ce	Port basic racetrack building script	2025-01-16T01:00:26+01:00
b534b2f	Fix racetrack is only built locally, re-enable preview buttons	2025-01-16T02:03:18+01:00
3f4ba6a	Fix null errors when using the base vehicle	2025-01-16T17:50:08+01:00
9dc9b9e	Fix ownership error when building the racetrack	2025-01-16T17:50:25+01:00
df3b2fe	Create vehicle component to access ownership conveniently	2025-01-16T19:51:28+01:00
c889c7f	Fix suspension working in networking	2025-01-16T20:16:13+01:00
2a36d13	Fix vehicle tight compensation offset	2025-01-17T02:26:00+01:00
bcf3b07	Recover the Network Race Orchestator and Supervisor for the race management	2025-01-17T19:52:53+01:00
dc6e7ab	Fill the other racetrack files with sample data	2025-01-17T20:18:46+01:00
f530de3	Separate Orchestrator and Supervisor	2025-01-17T20:19:48+01:00
05cc1af	Make spawned points children of the builder	2025-01-18T16:27:27+01:00
d9fe480	Create tiny race for development	2025-01-18T18:06:44+01:00
c9c48e9	Refactor the racetrack points prefabs to use the latest scripts	2025-01-18T18:07:12+01:00
824ae17	Fix connectivity bug because of legacy game parameters profile	2025-01-18T19:55:39+01:00
23cacb2	Fix too many unmanaged calls for checkpoints	2025-01-18T20:31:16+01:00
d7fa203	Refactor network race managers to the PTVK namespace	2025-01-19T02:33:40+01:00
40a785b	Fix GameParameters sometimes are not fetched on time	2025-01-19T17:33:49+01:00
da976e7	Make the server do the checkpoints	2025-01-19T17:37:03+01:00
6fb895d	Rework the network race supervisor to handle the checkpoints and winning condition	2025-01-20T00:56:52+01:00
c0f3ab7	Add parenthesis in some returns	2025-01-20T00:57:57+01:00
25627fe	Create events for racetrack and players race progress, add Supervisor event automation	2025-01-20T03:17:05+01:00
f050cca	Fix checkpoint check can get out of range	2025-01-20T03:31:35+01:00
5c008c1	Fix race players management for supervisor and its events	2025-01-22T02:00:31+01:00
ab2334b	Extract the Situation struct to the Data namespace	2025-01-22T02:50:56+01:00
27d5904	Remove separator from supervisor editor drawer	2025-01-22T02:51:12+01:00
72c04f4	Cleanup orchestrator script	2025-01-22T02:59:53+01:00

Id	Message	Datetime
d89f523	Refactor and optimize Application Versioning and Situation Attribute	2025-01-22T03:42:41+01:00
8f6b12a	Refactor Situation struct into Data.Gameflow and playables into Data.Playables	2025-01-22T19:44:45+01:00
68a7745	Refactor the Scene Skipper into a Situation Skipper	2025-01-22T20:27:45+01:00
c45eed3	Refactor OnReadyPlayersListChanged event and trigger	2025-01-22T20:30:38+01:00
71b3d3a	Refactor RacePreparer into ReadyPlayersHandler, move the orchestrator to the Gameflow subnamespace, remove obsolete RaceParameters prefab	2025-01-23T04:02:11+01:00
642037f	Handle the racer loading for all clients	2025-01-23T04:42:23+01:00
7ca314e	Develop basic orchestrator race starting flow	2025-02-11T01:57:52+01:00
ac9d60d	Prepare freeze and unfreeze vehicles system, make orchestrator unfreeze vehicles	2025-02-13T03:09:45+01:00
30ccec6	Simplify orchestrator script into one	2025-02-14T02:13:37+01:00
1022bbe	Develop basic teleportation for the vehicle controller	2025-02-14T02:47:04+01:00
c228a4e	Develop basic race gameflow	2025-02-14T03:24:24+01:00
f9cb086	Freeze vehicles when the race is ended	2025-02-14T03:29:48+01:00
6c9b833	Prepare the most basic race for testing	2025-02-14T20:40:53+01:00
c063fb9	Fix race rebuild and gameflow	2025-02-15T02:24:53+01:00
5db40e1	Build v0.6.3	2025-02-15T02:54:32+01:00
ca054f5	Update project to Unity 6000.0.41f1, import the Auditor package	2025-03-11T22:37:25+01:00
506d3f5	Update aupgrade cleanup files	2025-03-13T18:07:16+01:00
73b6ee9	Reenable denetworked vehicle	2025-03-13T18:08:59+01:00
1d45338	Fix the steering system	2025-03-15T06:01:04+01:00
06685c1	Optimize wheel grip system	2025-03-16T00:37:33+01:00
815ac4d	Fix engine and throttling system	2025-03-16T03:40:45+01:00
6fc8065	Add units to the speedometer	2025-03-16T03:49:27+01:00
2990413	Optimize the editor tools autogenerated files	2025-03-16T03:50:04+01:00
b5c6ed7	Rework suspension, spinning and gripping systems, fix dependencies and duplicated variables	2025-03-16T17:54:24+01:00
5f8e7d2	Update font atlases	2025-03-16T17:54:45+01:00
642aa4e	Develop and bind gearbox to the transmission	2025-03-17T14:01:33+01:00
3e0fa2d	Fix apply force only if wheel is grounded	2025-03-17T22:22:42+01:00
8a02f99	Develop and rework the gearbox and basic braking system	2025-03-19T04:18:15+01:00
7a70e55	Fix and optimize braking system	2025-03-20T03:19:00+01:00
55afd49	Fix ghost collisions, hide network ghost by default	2025-03-21T00:39:41+01:00
b99f195	Set grid material to sandbox floor prefab	2025-03-22T17:19:17+01:00
91a257d	Reimplement initial checking system	2025-03-30T17:49:47+02:00
f5f9da9	Delete legacy SituationSkipper	2025-03-30T17:51:27+02:00
2955548	Create basic custom vehicle models	2025-03-31T03:35:57+02:00
e3743b3	Create basic racetrack mesh	2025-04-01T00:32:08+02:00
2c06fc4	Optimize track array modifier with length	2025-04-01T00:56:29+02:00
a5487fa	Reorganize meshes and Sandbox scene	2025-04-01T19:54:16+02:00
6498795	Prepare racetrack prefab	2025-04-01T20:00:08+02:00
ccbee03	Optimize testing racetrack mesh	2025-04-01T20:31:45+02:00
f3d3ea0	Fix racetrack unused polys	2025-04-01T23:32:19+02:00
c12faab	Setup racetrack with colliders	2025-04-02T23:35:51+02:00
d3ca794	Rename legacy scenes to show date	2025-04-03T19:17:26+02:00
a84b16e	Rework the splash screen sequence	2025-04-04T01:15:39+02:00
0573ff5	Import more Google Fonts	2025-04-05T16:20:25+02:00
fb22438	Add more keybindings to the splash screen skipping action	2025-04-05T16:49:38+02:00
a577d93	Remove some unused TMP font assets, upgrade splash screen sequence and appeareance, skip DOTween animation	2025-04-05T17:58:44+02:00
df59586	Allow to skip sequence on editor	2025-04-05T18:27:48+02:00
b26fbea	Set splash screen to skip on editor	2025-04-06T17:32:21+02:00
a7a0ebf	Reorganize legacy button prefabas and animations, create improved main menu button sprite	2025-04-07T01:33:26+02:00
d51ccc1	Create simple button sound script	2025-04-07T01:55:04+02:00
e9a5b3b	Fix interactable font atlas configuration	2025-04-07T02:09:32+02:00
bbaced0	Refactor button click sound	2025-04-07T22:43:18+02:00
96fee02	Rename game startup sound with original name too	2025-04-07T22:44:41+02:00
79a1641	Replace grey buttons with the upgraded prefab	2025-04-08T19:08:04+02:00
c1ed628	Setup racetrack race, hide testing vehicles	2025-04-08T19:44:06+02:00
82aa6c0	Setup custom panel backgrounds, customize party members panel	2025-04-09T00:41:45+02:00
d3d66f2	Reorganize panel prefabs, add new input fonts, customize main menu panels and distribution	2025-04-09T19:24:40+02:00
16693dd	Finish main menu customization	2025-04-11T02:14:07+02:00
f215bc0	Update font atlases	2025-04-11T03:24:33+02:00
16e8335	Fix menu input field prefab mask	2025-04-12T01:54:26+02:00

Id	Message	Datetime
25c6096	Update font atlases	2025-04-12T17:29:47+02:00
b59a491	Remove some vehicles to fit in the prototype build	2025-04-12T18:48:49+02:00
be6f0cc	Rework and fix the race orchestation and behaviour	2025-04-14T00:25:35+02:00
c1b4076	Build v0.6.4	2025-04-22T03:42:14+02:00
4c7527f	Update development packages and .gitignore	2025-05-07T22:48:58+02:00
85cba32	Refactor Render Pipeline names, optimize mesh data for GPU	2025-05-09T17:32:03+02:00
b677a32	Update Rider IDE integration package	2025-05-14T16:52:58+02:00
caaa92e	Update both DevKit and ToolSet	2025-05-28T05:38:52+02:00

Tasks

Aa Task	↗ Blocked by	↗ Blocking	Date	Status
Canvases animations		Canvases transitions	@September 2, 2024 → September 3, 2024	Done
Canvases transitions	Canvases animations	Press start canvas	@September 4, 2024 → September 5, 2024	Done
Press start canvas	Canvases transitions	Main menu canvas	@September 6, 2024 → September 9, 2024	Done
Main menu canvas	Press start canvas	Social Networks links in main menu	@September 10, 2024 → September 18, 2024	Done
Social Networks links in main menu	Main menu canvas	Options canvas	@September 19, 2024	Done
Options canvas	Social Networks links in main menu	Working StartupChecker	@September 20, 2024 → September 25, 2024	Done
Working StartupChecker	Options canvas	Music player (Overlay Canvas)	@September 26, 2024 → September 27, 2024	Done
Music player (Overlay Canvas)	Working StartupChecker	Multiplayer Race canvas	@September 28, 2024	Done
Multiplayer Race canvas	Music player (Overlay Canvas)	Multiplayer+AI race system	@September 29, 2024 → October 20, 2024	Done
Multiplayer+AI race system	Multiplayer Race canvas	Vehicle controller system	@October 21, 2024 → November 17, 2024	Done
— ANTEPROYECTO —			@November 18, 2024 → November 24, 2024	Done
Vehicle controller system	Multiplayer+AI race system	Racetrack building system	@November 25, 2024 → January 13, 2025	Done
Racetrack building system	Vehicle controller system	Basic full gameflow loop	@January 14, 2025 → January 20, 2025	Done
Basic full gameflow loop	Racetrack building system	Settings system	@January 21, 2025 → February 16, 2025	Done
— PROYECTO - ANTECEDENTES —			@February 17, 2025 → March 9, 2025	Done

Aa Task	↗ Blocked by	↗ Blocking	Date	Status
Fix steering system		Fix throttling system	@March 10, 2025 → March 11, 2025	Done
Fix throttling system	Fix steering system	Fix suspension system	@March 12, 2025 → March 13, 2025	Done
Fix suspension system	Fix throttling system	Add reverse gear system	@March 14, 2025 → March 16, 2025	Done
Add reverse gear system	Fix suspension system	Add braking system	@March 17, 2025	Done
Add braking system	Add reverse gear system	Fix and optimize networked ghost	@March 18, 2025 → March 20, 2025	Done
Fix and optimize networked ghost	Add braking system	Design Scene Document	@March 21, 2025 → March 23, 2025	Done
Design Scene Document	Fix and optimize networked ghost	Reimplement initial check system	@March 24, 2025 → March 28, 2025	Done
Reimplement initial check system	Design Scene Document	Create 3D proxy vehicles	@March 29, 2025 → March 30, 2025	Done
Create 3D proxy vehicles	Reimplement initial check system	Create 3D proxy circuit	@March 31, 2025 → April 1, 2025	Done
Create 3D proxy circuit	Create 3D proxy vehicles	Customize GUI with basic assets	@April 2, 2025 → April 4, 2025	Done
Customize GUI with basic assets	Create 3D proxy circuit		@April 5, 2025 → April 11, 2025	Done
— PROYECTO - DESARROLLO —			@April 12, 2025 → June 1, 2025	Done
Proximity voice chat				Pending
Fix people do not get updated when someone leaves the party				Pending
Bug reporting system				Pending
Toast notifications (Overlay Canvas)				Pending
Fix main menu gamepad navigation				Pending

Aa Task	↗ Blocked by	↗ Blocking	Date	Status
Test full gamepad compatibility				Pending
Make Localization profile allow properties per sheet				Pending
Fix GUI selection cannot be recovered				Pending
Steam achievements				Pending
Fix audio spectrum last value is always zero				Pending
Input Map manager				Pending
Fix remaining TODOs				Pending
Fix Rich Presence integration to work for both Steamworks and Discord GameSDK				Pending
Concept art assets				Pending
Music assets				Pending
Statistics system	In-Game settings canvas	Race stats canvas (In-Game)		Pending
Settings system	Basic full gameflow loop	Savedata indicator (Overlay canvas)		Pending
Profile selection canvas				Pending
Race stats canvas (In-Game)	Statistics system	Visual Novel system		Pending
Keybindings system				Pending
Profile stats canvas (Extras)	Opponents AI	Dynamic loading screen		Pending

Aa Task	↗ Blocked by	↗ Blocking	Date	Status
Credits canvas	Splash Scene transitions	Opponents AI		Pending
Opponents AI	Credits canvas	Profile stats canvas (Extras)		Pending
Character voices system (ElevenLabs)				Pending
Visual Novel system	Race stats canvas (In-Game)	Splash Scene transitions		Pending
Splash Scene transitions	Visual Novel system	Credits canvas		Pending
In-Game settings canvas	Loading indicator (Overlay canvas)	Statistics system		Pending
Dynamic loading screen	Profile stats canvas (Extras)			Pending
Watermark (Overlay Canvas)				Pending
Loading indicator (Overlay canvas)	Savedata indicator (Overlay canvas)	In-Game settings canvas		Pending
Savedata indicator (Overlay canvas)	Settings system	Loading indicator (Overlay canvas)		Pending