

GRADO EN DISEÑO Y PRODUCCIÓN DE VIDEOJUEGOS

DESARROLLO DE UN PROTOTIPO DE JUEGO EN RED

Memoria Intermedia

ERIK ESPUÑES JUBERÓ

TUTOR: JORDI ARNAL MONTOYA

CURSO 2020-2021



ABSTRACT

This project will focus on creating a prototype of an arcade racing video game with online multiplayer.

The prototype will consist of two game modes. The first one will be a single player game mode. And the second one will be multiplayer, that will be the focus of this project.

Additionally, a game server will be developed to make the multiplayer game mode work properly. This server will control and manage the cheaters.

RESUM

Aquest projecte se centrarà a crear un prototip de videojoc de conducció arcade amb multijugador en línia.

El prototip constarà de dos modes de joc. El primer serà un mode un jugador senzill. El segon serà el multijugador, que serà en el que se centrarà aquest treball.

A més, es farà un servidor perquè el multijugador pugui funcionar correctament i així poder gestionar i controlar els jugadors perquè no facin trampes.

RESUMEN

Este proyecto se centrará en crear un prototipo de videojuego de conducción arcade con multijugador en línea.

El prototipo constará de dos modos de juego. El primero será un modo un jugador sencillo. Y el segundo será el multijugador, que será el que se centrará este trabajo.

Además, se hará un servidor para que este segundo modo de juego pueda funcionar correctamente y así poder gestionar y controlar a los jugadores para que no hagan trampas.

ÍNDICE

[ÍNDICE DE FIGURAS V](#_Toc70017675)

[ÍNDICE DE TABLAS VII](#_Toc70017676)

[GLOSARIO DE TÉRMINOS IX](#_Toc70017677)

[CAPÍTULO 1. INTRODUCCIÓN 1](#_Toc70017678)

[CAPÍTULO 2. ANÁLISIS DE REFERENTES 3](#_Toc70017679)

[2.1. REFERENTES DE JUEGOS MULTIJUGADOR EN LÍNEA 3](#_Toc70017680)

[2.1.1. ROCKET LEAGUE 3](#_Toc70017681)

[2.2. REFERENTES DE JUEGOS DE CARRERAS ARCADE 4](#_Toc70017682)

[2.2.1. MARIO KART 8 4](#_Toc70017683)

[2.2.2. TEAM SONIC RACING 8](#_Toc70017684)

[2.2.3. CRASH TEAM RACING NITRO FUELED 10](#_Toc70017685)

[CAPÍTULO 3. MARCO TEÓRICO 15](#_Toc70017686)

[3.1. DEL MULTIJUGADOR LOCAL AL MULTIJUGADOR EN LÍNEA 15](#_Toc70017687)

[3.1.1. EVOLUCIÓN DE LOS JUEGOS MULTIJUGADOR 15](#_Toc70017688)

[3.1.2. MULTIJUGADOR LOCAL 16](#_Toc70017689)

[3.1.3. MULTIJUGADOR EN LÍNEA 17](#_Toc70017690)

[3.2. PROTOCOLOS PARA LA COMUNICACIÓN EN TIEMPO REAL 19](#_Toc70017691)

[3.2.1. ¿CÓMO FUNCIONA INTERNET? 19](#_Toc70017692)

[3.2.2. TRANSPORT CONTROL PROTOCOL (TCP) 23](#_Toc70017693)

[3.2.3. USER DATAGRAM PROTOCOL (UDP) 24](#_Toc70017694)

[3.3. METODOLOGÍAS PARA EL DESARROLLO DE VIDEOJUEGOS EN LÍNEA 26](#_Toc70017695)

[3.3.1. CLIENTE-SERVIDOR 26](#_Toc70017696)

[3.3.2. PEER-TO-PEER (P2P) 28](#_Toc70017697)

[3.3.3. SERVIDOR-SERVIDOR 29](#_Toc70017698)

[3.4. FUNCIONAMIENTO DE UN COCHE 29](#_Toc70017699)

[3.4.1. MOTOR 29](#_Toc70017700)

[3.4.2. SUSPENSIÓN 30](#_Toc70017701)

[3.4.3. DIRECCIÓN 31](#_Toc70017702)

[3.4.4. IMPLEMENTACIÓN EN EL JUEGO 31](#_Toc70017703)

[CAPÍTULO 4. OBJETIVOS 33](#_Toc70017704)

[4.1. OBJETIVOS PRINCIPALES 33](#_Toc70017705)

[4.2. OBJETIVOS SECUNDARIOS 33](#_Toc70017706)

[CAPÍTULO 5. DISEÑO METODOLÓGICO Y CRONOGRAMA 35](#_Toc70017707)

[5.1. HERRAMIENTAS DE DESARROLLO 35](#_Toc70017708)

[5.1.1. HERRAMIENTAS DE DESARROLLO DEL CLIENTE 35](#_Toc70017709)

[5.1.2. HERRAMIENTAS DE DESARROLLO DEL SERVIDOR 36](#_Toc70017710)

[5.2. METODOLOGÍA 37](#_Toc70017711)

[5.2.1. METODOLOGÍA DE LA DOCUMENTACIÓN 37](#_Toc70017712)

[5.2.2. METODOLOGÍA DEL PRODUCTO 38](#_Toc70017713)

[5.3. PLANIFICACIÓN 38](#_Toc70017714)

[CAPÍTULO 6. DESARROLLO 43](#_Toc70017715)

[6.1. DISEÑO 43](#_Toc70017716)

[6.1.1. GDD 43](#_Toc70017717)

[6.1.2. UML DEL SERVIDOR 50](#_Toc70017718)

[6.1.3. BASE DE DATOS 51](#_Toc70017719)

[6.2. CLIENTE 54](#_Toc70017720)

[6.2.1. CONTROLADOR DEL VEHÍCULO 54](#_Toc70017721)

[6.2.2. REGLAS DEL JUEGO 60](#_Toc70017722)

[6.2.3. INTERFAZ GRÁFICA DE USUARIO 63](#_Toc70017723)

[6.2.4. OBJETOS 65](#_Toc70017724)

[6.2.5. INTELIGENCIA ARTIFICIAL 67](#_Toc70017725)

[6.3. SERVIDOR 67](#_Toc70017726)

[6.3.1. BASE DE DATOS 67](#_Toc70017727)

[6.3.2. INSTANCIA DEL JUEGO 70](#_Toc70017728)

[6.3.3. API 70](#_Toc70017729)

[6.4. MODO MULTIJUGADOR 72](#_Toc70017730)

[6.4.1. COMUNICACIÓN CON EL SERVIDOR 72](#_Toc70017731)

[6.4.2. MATCHMAKING 75](#_Toc70017732)

[6.4.3. MOVIMIENTO DE LOS JUGADORES 75](#_Toc70017733)

[6.4.4. LANZAR OBJETOS 75](#_Toc70017734)

[6.4.5. PREDICCIÓN, RECONCILIACIÓN E INTERPOLACIÓN 75](#_Toc70017735)

[6.4.6. BOTS 75](#_Toc70017736)

[CAPÍTULO 7. BIBLIOGRAFÍA 77](#_Toc70017737)

# ÍNDICE DE FIGURAS

[Figura 2.1: Información que proporciona Rocket League de la conexión. 4](#_Toc69497468)

[Figura 2.2: Controles de Mario Kart. 5](#_Toc69497469)

[Figura 2.3: Tabla de probabilidad de los objetos en Mario Kart 6](#_Toc69497470)

[Figura 2.4: Modos de juego de Mario Kart. 7](#_Toc69497471)

[Figura 2.5: Controles de Team Sonic Racing 8](#_Toc69497472)

[Figura 2.6: Algunos wisp que hay en Team Sonic Racing. 9](#_Toc69497473)

[Figura 2.7: Modos de juego de Team Sonic Racing. 10](#_Toc69497474)

[Figura 2.8: Controles Crash Team Racing Nitro Fueled 11](#_Toc69497475)

[Figura 2.9: Todos los objetos de Crash Team Racing Nitro Fueled. 12](#_Toc69497476)

[Figura 2.10: Modos de juego de Crash Team Racing Nitro Fueled. 13](#_Toc69497477)

[Figura 3.1: Imagen del juego de Empire. 15](#_Toc69497478)

[Figura 3.2: Imagen del modo cooperativo de Final Fantasy Chrystal Chronicles. 17](#_Toc69497479)

[Figura 3.3: Imagen del modo competitivo de Call of Duty: Black Ops - Cold War. 18](#_Toc69497480)

[Figura 3.4: Secuencia de los paquetes en el protocolo TCP. 23](#_Toc69497481)

[Figura 3.5: Secuencia de los paquetes en el protocolo UDP. 25](#_Toc69497482)

[Figura 3.6: Topología cliente-servidor 26](#_Toc69497483)

[Figura 3.7: Secuencia de los paquetes en la topología cliente-servidor. 27](#_Toc69497484)

[Figura 3.8: Topología p2p 28](#_Toc69497485)

[Figura 3.9: Ciclo del motor. 30](#_Toc69497486)

[Figura 3.10: Sistema que hace que gire el vehículo. 31](#_Toc69497487)

[Figura 5.1: Logotipos de Unreal Engine y JetBrains Rider. 36](#_Toc69497488)

[Figura 5.2: Logotipos de JetBrains WebStorm y Node.js. 37](#_Toc69497489)

[Figura 6.1: Diagrama ERD de la base de datos del servidor. 49](file:///G:\Desarrollo%20de%20un%20Prototipo%20de%20Videojuego%20en%20Red\Documentación\Entregables\MemoriaIntermedia_ErikEspunesJubero.docx#_Toc69497490)

# ÍNDICE DE TABLAS

[Tabla 3.1: Modelo de capas OSI (ISO/IEC JTC 1, 1994) 19](#_Toc69497502)

[Tabla 3.2: Tabla de una llamada DNS realizada con Packet Tracer. 21](#_Toc69497503)

[Tabla 3.3: Tabla de una respuesta DNS realizada con Packet Tracer. 21](#_Toc69497504)

[Tabla 3.4: Tabla de una llamada HTTP realizada con Packet Tracer. 22](#_Toc69497505)

[Tabla 3.5: Tabla de una respuesta HTTP realizada con Packet Tracer. 22](#_Toc69497506)

[Tabla 5.1: Cronograma del TFG 40](#_Toc69497507)

[Tabla 6.1: Tabla de los objetos y su probabilidad de aparición. 45](#_Toc69497508)

[Tabla 6.2: Tabla de los modos de dificultad. 47](#_Toc69497509)

[Tabla 6.3: Tabla del sistema de puntuación. 47](#_Toc69497510)

[Tabla 6.4: Tabla de la conversión de las monedas. 48](#_Toc69497511)

[Tabla 6.5: Tabla de la API. 56](#_Toc69497512)

# GLOSARIO DE TÉRMINOS

|  |  |
| --- | --- |
| BOTS | Enemigos controlados por una IA en un juego en red. |
| DOTS | Data Oriented Technology Stack |
| ECS | Entity Component System |
| E-SPORTS | Deportes electrónicos. |
| HTML | HyperText Markup Language |
| HUD | Head-Up Display |
| IA | Inteligencia Artificial |
| IDE | Integrated Development Environment |
| LAG | Retardo producido por la comunicación en tiempo real. |
| LAN | Local Area Network |
| Packet Tracer | Es un software de simulación de redes de la empresa CISCO. |
| RFC | Request for Comments, son los documentos oficiales de redes y deben estar aprobados por la Internet Engineering Task Force (IETF). |
| TFG | Trabajo de Fin de Grado |

# INTRODUCCIÓN

Desde hace años los juegos multijugador están en el orden del día. Cada vez se ha ido jugando más de esa forma hasta el punto de llegar a hacer torneos en línea, como son los deportes electrónicos.

En este TFG se hará un juego de conducción arcade centrado en el multijugador. Así que para hacer este producto se separará en tres partes.

Primero se hará la parte del cliente, que será como tener el juego sin multijugador. Allí estará implementado todas las mecánicas que existan en el juego. Por ejemplo, el sistema de conducción, la inteligencia artificial, los objetos, las reglas del juego y el HUD.

Una vez la parte del cliente funciona perfectamente se hará el servidor. Este será el encargado de gestionar la partida multijugador, contendrá una base de datos para poder guardar estadísticas de nuestros jugadores, entre otros. Será un servidor autoritario, por lo que todos los inputs de los jugadores irán a la simulación de la instancia del juego que estará ejecutando el servidor.

Para acabar se hará la integración del cliente con el servidor. Lo que se hará es hacer el modo de juego del multijugador, que hará que el cliente se comunique con el servidor y así poder disfrutar de las partidas contra otros jugadores.

# ANÁLISIS DE REFERENTES

Para analizar los referentes se debe separar en dos partes, los referentes de multijugador en línea y los referentes de juegos de karts.

## REFERENTES DE JUEGOS MULTIJUGADOR EN LÍNEA

En este subapartado, solo se verá los aspectos, en el que el producto que se elaborará se ha tomado como referencias. En este caso solo se ha obtenido de referente el Rocket League.

### ROCKET LEAGUE

Rocket League (Psyonix, 2015) es un juego de fútbol arcade en el que llevas coches. Los partidos están compuestos de dos equipos, entre uno y cuatro jugadores, y el objetivo es anotar más goles que el rival para ganar. Estos tienen una duración de 5 minutos, si uno de los dos equipos gana. En caso de empate, se va a una prórroga y esta no se acaba hasta que uno de los dos equipos marca un gol.

Analizándolo en su modo multijugador, Rocket League es un juego cliente-servidor. Y eso se puede deducir muy bien cuando se entra en la partida, sale un texto que indica que te estás conectando a una región, aparte de que se puede elegir la región en la que el usuario se quiere conectar. Eso es un claro indicador que se trata de una metodología cliente-servidor.

Además, Rocket League hace una cosa muy interesante, que es informarte de los errores o anomalías que puedes tener mientras se está jugando un partido. Como se puede ver en la imagen siguiente.



Figura 2.1: Información que proporciona Rocket League de la conexión.

Para este proyecto Rocket League es el gran referente de multijugador, ya que más o menos se trata de algo parecido al producto a desarrollar. Es decir aunque sean géneros diferentes, los dos usan vehículos para moverse y además de la información que da al usuario en torno a los problemas de conectividad es muy bueno.

## REFERENTES DE JUEGOS DE CARRERAS ARCADE

En este subapartado solo se verán la configuración de los controles, las mecánicas y los modos de juegos que tiene cada juego que se analizará. Se analizarán tres juegos de este género, Mario Kart 8, Team Sonic Racing y Crash Team Racing Nitro Fueled.

### MARIO KART 8

Mario Kart 8 (Nintendo, 2013), que fue lanzado por Nintendo en el 2013, es el octavo juego de la saga Mario Kart. Dicha saga se caracteriza por unas carreras arcades asimétricas de, normalmente, doce jugadores. En dicha carrera solo hay una regla, seguir el trazado, aunque hay algún atajo. En las carreras, también se pueden coger objetos, que, aunque son aleatorios, son una manera muy fácil de ganar posiciones o defender la que tienes.

A continuación, se explicarán la configuración de los controles que tiene el juego, sus mecánicas y para acabar sus modos de juego.

#### CONTROLES



Figura 2.2: Controles de Mario Kart.

Teniendo como referencia el mando de la Switch, Mario Kart tiene los siguientes controles:

* A: Acelerar
* B: Frenar
* L: Lanzar objeto
* R: Derrapar
* Joystick Izquierdo: Girar (aunque también se puede habilitar la opción de girar rotando el mando como si fuese el volante.)

#### MECÁNICAS

En este apartado se analizará la mecánica del movimiento del personaje, como también los objetos que hay en el juego.

Primero de todo el personaje tiene cuatro mecánicas simples, acelerar, frenar, girar y derrapar.

La primera mecánica, la de acelerar, es muy simple, se pulsa el botón de acelerar y el vehículo acelera hasta llegar a su velocidad máxima. Si se deja de acelerar el vehículo llegará a frenarse por completo, pero tardando más que si frenas pulsando el botón de frenar.

El freno es muy parecido a la de acelerar, si se pulsa su botón, este llegará a frenar el coche. Pero si se sigue frenando, una vez el vehículo llega a velocidad cero, este irá marcha atrás. Yendo marcha atrás si dejas de pulsar el botón, el vehículo se frenará por completo, de la misma forma que pasa al acelerar.

El giro es bastante simple, se mueve el joystick, o el mando si se ha configurado de tal manera, y el coche gira. Esta mecánica se junta con la siguiente, ya que si se está mucho rato girando el kart empezará a derrapar.

La mecánica de derrape te ofrece hacer un giro con más ángulo que girando de manera normal y además te da un turbo extra que va aumentando en función de la duración del derrape.

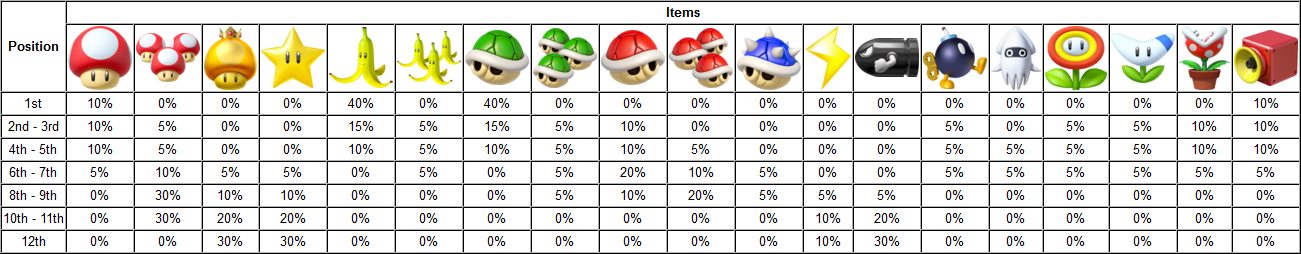


Figura 2.3: Tabla de probabilidad de los objetos en Mario Kart

Los objetos que pueden salir si se recoge una caja misteriosa, varían según la posición que esté el jugador. Como se ve en la figura, hay algunos objetos, los que más ventajas te dan, solo aparecen si estas en las últimas posiciones. En cambio, los objetos más defensivos aparecen cuando estas en las primeras posiciones. El objeto más curioso es el cohete, ya que te lleva hasta cierta posición que el juego cree.

#### MODOS DE JUEGO

Mario Kart tiene cuatro modos de juego.



Figura 2.4: Modos de juego de Mario Kart.

El primero, Gran Prix, se puede jugar tanto contra la máquina o contra otros jugadores y se caracteriza por disputar una copa que está compuesta de cuatro carreras normalmente de tres vueltas. Este modo de juego tiene cinco niveles de dificultad, 50cc, 100cc 150cc, espejo y 200cc. Cada dificultad es más velocidad para el vehículo del jugador, a excepción del modo espejo que tiene una velocidad de 150cc, pero con los mapas invertidos.

El segundo es más para practicar los circuitos, tienes normalmente tres vueltas en un circuito que el jugador elija, con tres turbos que puedes usar cuando quieras de la vuelta contrarreloj, sin la posibilidad de recoger objetos. La comunidad usa este modo de juego para establecer récords mundiales en los circuitos, además el jugador puede elegir si correr solo o contra un fantasma. La contrarreloj solo tiene modo un jugador.

El tercer modo de juego es bastante parecido al primero, pero el jugador puede elegir las reglas de la competición. El jugador elige la dificultad, si quiere que haya dos equipos, la dificultad de la inteligencia artificial, los tipos de vehículo de la máquina y los circuitos que aparecen, el jugador puede poner todos los circuitos del juego. En este modo de juego también hay la posibilidad de jugar contra otros jugadores.

El cuarto y último, también es un modo de juego multijugador. En este debes competir individualmente o por equipos a una serie de competición, por ejemplo, coger el máximo número de monedas, evitar que te golpeen, entre otros. Aquí los mapas que hay son originales para este modo en específico.

### TEAM SONIC RACING

En cuanto a Team Sonic Racing (Sumo Digital, 2019), es un juego de carreras arcade, muy parecido a Mario Kart 8, pero con los personajes del universo Sonic.

Seguidamente se detallará la configuración de los controles, sus mecánicas y para acabar sus modos de juego que hay en Team Sonic Racing.

#### CONTROLES



Figura 2.5: Controles de Team Sonic Racing

Teniendo como referencia el mando de la Switch, Team Sonic Racing tiene los siguientes controles:

* R: Acelerar
* L: Frenar/Derrapar
* B: Lanzar objeto
* A: Ceder Objetos
* X: Megaturbo
* Joystick Izquierdo: Girar

#### MECÁNICAS

En las mecánicas son bastantes parecidas a las de Mario Kart, aunque tiene alguna única. La de acelerar, frenar, girar y derrapar no se ha notado ninguna diferencia.

El cambio en este juego viene con el megaturbo, como se explicará en el apartado siguiente, la mayoría de las carreras en Team Sonic Racing son por equipos de tres, por lo que, si juegas en el modo de un jugador tienes dos IA que van en tu equipo. Cada vez que un miembro del equipo hace algún derrape, o te da algún objeto, ya que los objetos se pueden dar a un compañero de equipo, la barra del mega turbo sube. El mega turbo como bien dice el nombre, da un turbo a todos los integrantes del equipo bastante más largo que los objetos.

En el tema de los objetos, también es bastante parecido a Mario Kart. Aunque en este juego se llaman wisps, aparecen dependiendo de la posición que esté el personaje. El más curioso son el wisp de temblor, que hace aparecer pilares de piedra en el camino



Figura 2.6: Algunos wisp que hay en Team Sonic Racing.

#### MODOS DE JUEGO

Team Sonic Racing tiene cuatro modos de juego de un jugador.

El primero es un modo aventura. Es el modo principal del juego En él se debe hacer una serie de misiones, que son carreras o contrarreloj.



Figura 2.7: Modos de juego de Team Sonic Racing.

Después tiene los típicos modos de un jugador, el gran premio, que es jugar una serie de cuatro carreras en equipo, dos IA, o individual. Además, se puede elegir la dificultad que son tres, fácil, normal y difícil. Al acabar este gran premio, en el caso de ganar, te dan la copa.

El siguiente modo de juego es una carrera normal. Eliges circuito, dificultad y si quieres ir por equipos y juegas una sola carrera.

El último es el modo contrarreloj, que es dar vueltas al circuito para mejorar tu tiempo o practicar.

### CRASH TEAM RACING NITRO FUELED

Para acabar Crash Team Racing Nitro Fueled (Beenox, 2019), remake del mítico juego Crash Team Racing (Naughty Dog, 1999) de PlayStation 1.

#### CONTROLES

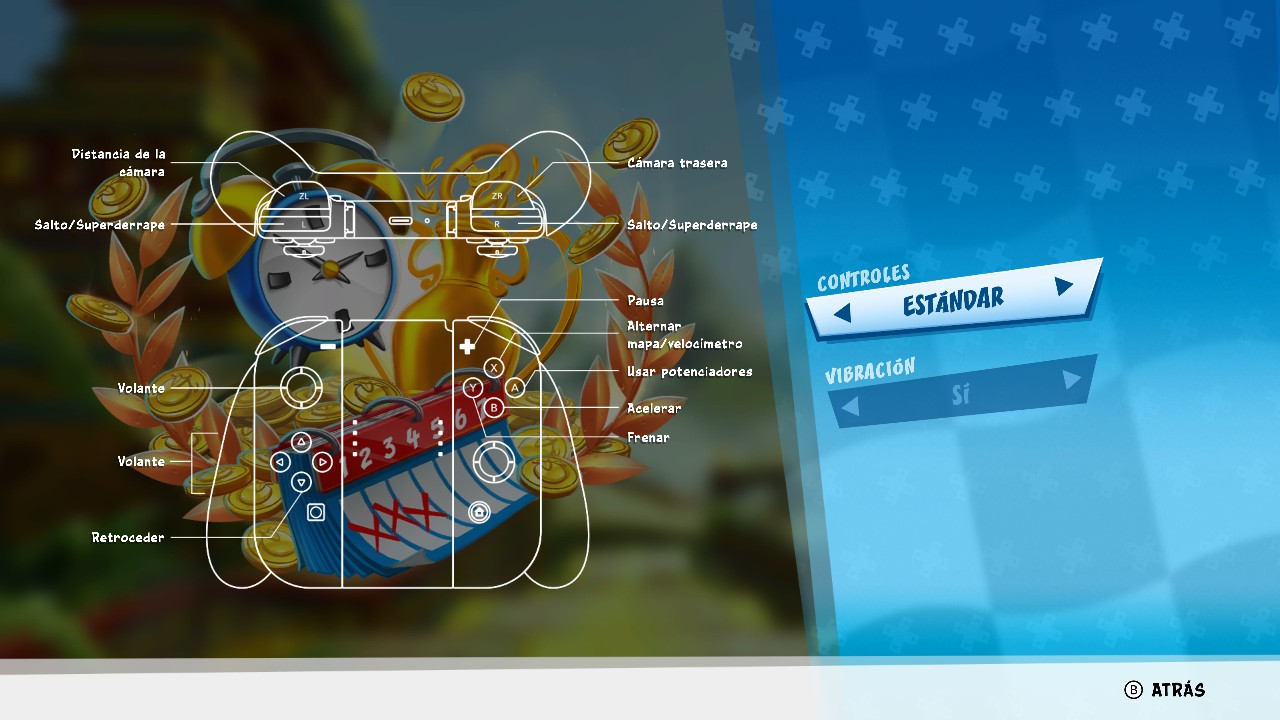


Figura 2.8: Controles Crash Team Racing Nitro Fueled

Teniendo como referencia el mando de la Switch, Crash Team Racing Nitro Fueled tiene los siguientes controles:

* B: Acelerar
* Y: Frenar
* L/R: Derrapar
* A: Lanzar objeto
* X: Habilidad de Equipo
* Joystick Izquierdo: Girar

#### MECÁNICAS

Las mecánicas del movimiento del vehículo son bastantes parecidas a los dos juegos anteriores. Pero el derrape es lo que marca la diferencia. Ya que requiere de habilidad para el jugador. En los otros dos pulsas el botón y ya derrapa solo y te dan el turbo. En Crash una vez se está derrapando se debe pulsar el botón de derrapar, otra vez, en el momento preciso para tener el turbo.



Figura 2.9: Todos los objetos de Crash Team Racing Nitro Fueled.

En cuanto a los objetos, hay algunos que requieren de habilidad del jugador. Tanto en Mario Kart como en Team Sonic Racing, los objetos se lanzan y ya no se debe hacer nada. En Crash hay objetos, como la caja de TNT, que para que no le explote al jugador debe pulsar el botón de lanzar objeto repetidamente para librarse.

En cuanto a la aparición, es bastante aleatorio. Es decir, hay algunos objetos que no saldrán estando en alguna posición específica, pero no parece que haya una probabilidad para que te salgan los objetos.

#### MODOS DE JUEGO

El juego tiene siete modos de juego de un jugador.

Su modo de juego principal es la historia. En este modo de juego hay como un mundo abierto para ir a las misiones o carreras.



Figura 2.10: Modos de juego de Crash Team Racing Nitro Fueled.

Después hay los modos típicos que salen en los otros juegos, carrera, torneo y contrarreloj.

Además, Crash Team Racing Nitro Fueled tiene un modo batalla que en él se debe competir individualmente en diferentes batallas como, capturar la bandera, battle royale, etc. Este modo tiene mapas exclusivos.

La carrera de anillos se debe seguir el camino formado por los anillos que hay en el mapa para ganar puntuación y alargar el tiempo, que si este último llega a cero se acaba la carrera.

La carrera por la reliquia es una contrarreloj, pero mucho más avanzado, es decir, para ganar se debe hacer una vuelta en un tiempo límite, que es bastante difícil.

Para acabar Crash Team Racing Nitro Fueled tiene modos de desafío, que son carreras, pero que para ganar se deben conseguir ciertos objetivos, el CTR que se deben recoger las letras C, T y R que están repartidos en el mapa. El desafío de cristales, en cambio, es en los mapas de batallas y se deben conseguir veinte cristales en un tiempo límite.

# MARCO TEÓRICO

En este apartado se explicará la teoría que hay detrás del producto a desarrollar.

Este apartado se separará en tres subapartados, el primero una evolución de los juegos multijugador. Después vendrán los protocolos que existen para comunicar-se a través de internet. A continuación, se verá que tipo de metodologías existen en los videojuegos para desarrollar un videojuego multijugador en línea. Para acabar se verán los componentes de un coche, y como se desarrollaría esa parte en cualquier motor.

## DEL MULTIJUGADOR LOCAL AL MULTIJUGADOR EN LÍNEA

Desde el inicio de los videojuegos, el multijugador está presente. Así que en este apartado se verá la evolución que ha tenido los videojuegos multijugador y además se verán los dos métodos más usados y se explicará que ventajas e inconvenientes tienen.

### EVOLUCIÓN DE LOS JUEGOS MULTIJUGADOR

El multijugador en los videojuegos existe, prácticamente desde su creación. Tennis for Two (William Higginbotham, 1958), que fue hecho en un osciloscopio, era un juego para dos jugadores, por ello multijugador.

El primer juego multijugador en línea surgió el año 1973, para PLATO (Universidad de Illinois, 1960), que fue una de las primeras máquinas con un sistema de internet. Ese juego se llama Empire (Daleske y Warner, 1973), un juego de estrategia por turnos. Aunque prácticamente surgió a la par que Maze War (Jim Bowery, 1974) un juego de disparos en primera persona en línea.

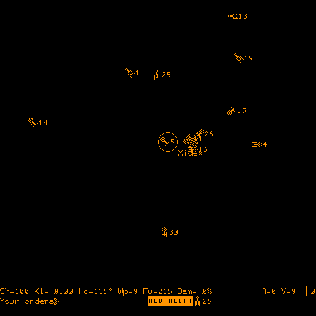


Figura 3.1: Imagen del juego de Empire.

Pero no fue hasta que salió el juego en LAN que hubo un incremento de demanda de este género de juegos. El juego en LAN, según (Glazer y Sanjay Madhav, 2016) se caracteriza ser un multijugador en red, pero requiere de estar en la misma red de internet.

El primero en usar de esa tecnología fue Doom (ID Software, 1993). En él se podía jugar tanto cooperativo como competitivo hasta cuatro jugadores. Con este sistema muchos juegos se prepararon para la era del multijugador en línea.

Con el auge de internet en la década de los 90, los videojuegos empezaron a implementar el multijugador en línea. Uno de los primeros en tener un multijugador en línea fue Epic Game’s Unreal (Epic MegaGames y Digital Extremes, 1998).

Sobre la década de los 2000, las consolas empezaron a implementar el multijugador en línea con la aparición de Xbox Live, PlayStation Network, entre otros.

### MULTIJUGADOR LOCAL

Según (Glazer y Sanjay Madhav, 2016) un juego con multijugador local es aquel que está diseñado para jugar diferentes personas en el mismo ordenador. Un ejemplo de ellos son la saga Mario Kart (Nintendo, 1992), los que salieron antes del 8, ya que el último que introduce el juego en línea, el Final Fantasy Chrystal Chronicles (Square Enix, 2004), entre muchos otros. Estos juegos implementaban un sistema que permitía a diversos jugadores jugar contra ellos, Mario Kart, o en cooperativo siguiendo una historia, Final Fantasy Chrystal Chronicles.



Figura 3.2: Imagen del modo cooperativo de Final Fantasy Chrystal Chronicles.

Este modo tiene alguna ventaja respecto al multijugador en línea, por ejemplo, no hay lag provocado por la conexión a internet.

Por el contrario, para poder disfrutar de las ventajas del multijugador local, necesitas que se esté jugando en la misma consola, por consiguiente, esos jugadores deben estar en el mismo espacio físico o muy cerca geográficamente.

### MULTIJUGADOR EN LÍNEA

Según (Glazer y Sanjay Madhav, 2016) el multijugador en línea es aquel que requiere de, como mínimo, dos dispositivos conectados el uno del otro en la misma sesión de juego. Un ejemplo pueden ser la saga Call of Duty (Activision, 2004), el Minecraft (Mojang, 2011), entre muchos otros.

Mientras que, en uno, Call of Duty, su modalidad principal es jugar en competitivo contra otros jugadores. El otro se caracteriza por la exploración y la cooperación en un mundo que puedes hacer lo que se te pase por la cabeza.



Figura 3.3: Imagen del modo competitivo de Call of Duty: Black Ops - Cold War.

El multijugador en línea, por lo que respecta al multijugador local, puedes jugar con gente de todo el mundo, conocidos o desconocidos.

El principal problema que hay es que requiere de conexión a internet, aunque actualmente casi toda la población mundial tiene de ese servicio. En la era moderna de las consolas, el multijugador en línea requiere de una suscripción de pago.

Además, el multijugador al depender de la conexión a internet puede provocar que haya lag en los juegos. Por lo que, como se explicará en más detalle en la siguiente sección, se pueden perder paquetes que se envían a internet y hacer que la experiencia del juego sea mala. Por lo que los desarrolladores tendrán que innovar para hacer que este problema no cause muchas malas experiencias para el jugador.

## PROTOCOLOS PARA LA COMUNICACIÓN EN TIEMPO REAL

Casi toda la población mundial tiene acceso a internet y para hacer un videojuego en línea es crucial disponer de esa tecnología. Pero para entender cómo funcionan los videojuegos multijugador en línea se debe hacer la siguiente pregunta. ¿Cómo funciona internet?

### ¿CÓMO FUNCIONA INTERNET?

Después de leer un artículo de Rus Shuler (Shuler, 2002) que te explica el funcionamiento de internet. Como bien dice su nombre es una red interconectada de dispositivos y encaminadores (routers). Por lo que todo el mundo está conectado con cables, routers y dispositivos.

Su funcionamiento, aunque tiene algunas complejidades, puede ser fácil de entender. Internet se debe ver como si se tratase un servicio de mensajería. Por ejemplo, un usuario de internet se quiere acceder a una página web. Para poder tener esta información, nuestro navegador enviará un paquete al servidor que esté alojado esa web. En ese paquete habrá toda la información requerida para llegar a ese servidor. Esa información está organizada por un protocolo llamado modelo de capas OSI.

|  |  |
| --- | --- |
| Capa 7 | Aplicación |
| Capa 6 | Presentación |
| Capa 5 | Sesión |
| Capa 4 | Transporte |
| Capa 3 | Red |
| Capa 2 | Enlace |
| Capa 1 | Física |

Tabla 3.1: Modelo de capas OSI (ISO/IEC JTC 1, 1994)

De arriba abajo, la capa siete serían las aplicaciones, del dispositivo, que hacen uso de la red.

La capa seis, la de presentación, es la encargada de hablar en el mismo idioma, por así decirlo. Por ejemplo, si envías una imagen en formato JPG, esta capa es la encargada de decir que esa imagen está en ese formato.

La capa de sesión es la encargada de mantener la sesión activa entre diferentes dispositivos. Se puede usar para realizar llamadas de voz o video a través de internet.

La capa cuatro, es la que se debe encargar de transmitir los datos. Sus protocolos estrella son UDP y TCP.

La capa de red trabaja un único protocolo, llamado Internet Protocol o IP. Si llega un paquete en que la dirección IP no es la misma que la del dispositivo, esta capa descartará el paquete, ya que sabrá que no es para él. La dirección IP está compuesta de números decimales. Esta puede variar dependiendo del tiempo, aunque siempre estará asociada a la red que esté conectado. Un ejemplo de dirección IP es la siguiente 192.168.1.123.

La capa dos, es la encargada de tener la dirección física del dispositivo, MAC, y al igual que la capa de red, ningún paquete pasará a la siguiente capa si la MAC no es la misma que la del dispositivo. La MAC es única en cada ordenador, por lo que no se pueden repetir y es un número hexadecimal. Un ejemplo de una dirección MAC sería esta 00:25:08:55:2B:AB.

Para acabar, la capa uno, es la encargada de convertir toda la trama o paquete generado en señales eléctricas para poderla transmitir por la corriente.

Las capas OSI, cuando se debe generar una trama, siempre empieza en la capa siete hasta abajo. En el caso de leer la trama, va al revés. Primero empieza por la capa uno, hasta llegar a la siete, si llega.

Ahora sí, se va a explicar cómo funcionaria una llamada a una página web.

Primero de todo se establecería la llamada HTTP (HyperText Transfer Protocol) desde la capa 7. Primero miraría si el dispositivo conoce la dirección IP donde se encuentra esa página web. En caso de no conocerlo enviaría un paquete del servicio DNS (Domain Name Service). El DNS es un servicio que contiene una lista de dominios web y sus respectivas IP. Por ejemplo, en esa lista estaría guardado algo parecido a así “google.com = 8.8.8.8”. Por lo que este servicio le envías un dominio web y te devuelve su IP. El paquete que enviaría nuestro dispositivo es parecido al siguiente[[1]](#footnote-1).

|  |  |
| --- | --- |
| Capa 7 | DNS – “google.com” |
| Capa 6 | — |
| Capa 5 | — |
| Capa 4 | UDP |
| Capa 3 | IP ORIG: 10.0.1.66  IP DEST: 172.16.0.3 |
| Capa 2 | MAC ORIG  MAC DEST |
| Capa 1 | — |

Tabla 3.2: Tabla de una llamada DNS realizada con Packet Tracer.

Como se ve en la capa siete, marca que se trata de un servicio DNS y le envía el nombre del dominio. Como no le hace falta ningún protocolo de capa seis o cinco, los omite. En la capa cuatro establece un transporte tipo UDP (más adelante se verá que es). En la capa tres y dos establece una IP origen, IP destino y MAC origen, MAC destino, respectivamente. Este paquete pasaría todos los dispositivos de redes que haya hasta llegar a su destinatario este situado en cualquier parte del mundo, en el caso de que pueda llegar.

|  |  |
| --- | --- |
| Capa 7 | DNS – “8.8.8.8” |
| Capa 6 | — |
| Capa 5 | — |
| Capa 4 | UDP |
| Capa 3 | IP ORIG: 172.16.0.3  IP DEST: 10.0.1.66 |
| Capa 2 | MAC ORIG  MAC DEST |
| Capa 1 | — |

Tabla 3.3: Tabla de una respuesta DNS realizada con Packet Tracer.

El paquete que le enviaría el servidor DNS al dispositivo sería igual, pero en la capa siete en vez de enviar el nombre del dominio llevaría la IP de la página web.

|  |  |
| --- | --- |
| Capa 7 | HTTP — GET (http://google.com) |
| Capa 6 | — |
| Capa 5 | — |
| Capa 4 | TCP |
| Capa 3 | IP ORIG: 10.0.1.66  IP DEST: 8.8.8.8 |
| Capa 2 | MAC ORIG  MAC DEST |
| Capa 1 | — |

Tabla 3.4: Tabla de una llamada HTTP realizada con Packet Tracer.

Como pasa con la llamada DNS, las capas seis, cinco y uno no tienen ninguna función especial. La diferencia que hay entre la llamada HTTP y DNS es que, para empezar en la capa siete cambia el protocolo que usa. En este caso, para visualizar una página web. Después salta la atención que, en la capa tres cambie de protocolo a TCP. Para acabar se ve que la IP destino, en la capa tres, es la del servidor web que se ha pedido antes.

|  |  |
| --- | --- |
| Capa 7 | HTTP - HTML |
| Capa 6 | — |
| Capa 5 | — |
| Capa 4 | TCP |
| Capa 3 | IP ORIG: 8.8.8.8  IP DEST: 10.0.1.66 |
| Capa 2 | MAC ORIG  MAC DEST |
| Capa 1 | — |

Tabla 3.5: Tabla de una respuesta HTTP realizada con Packet Tracer.

Lo que devolvería la llamada HTTP es un archivo HTML, que sería lo que abriría el navegador web.

Toda la información sacada a través de los servicios HTTP o DNS corresponden a sus respectivos documentos oficiales, RFC 2616 (Fielding et al., 1999) y RFC 1035 (Mockapetris, 1987) respectivamente.

### TRANSPORT CONTROL PROTOCOL (TCP)

Según su RFC (Postel, 1981), TCP es un protocolo de capa cuatro. Su función es gestionar el envío o recibo de datos entre dos dispositivos. Solo uno puede enviar archivos y el otro debe recibirlos.

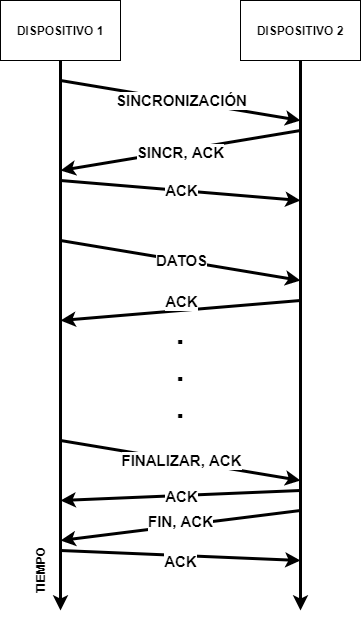


Figura 3.4: Secuencia de los paquetes en el protocolo TCP.

TCP es un protocolo orientado a conexión, así que para poderse comunicar con otro dispositivo debe hacer un establecimiento. En TCP debe hacer un establecimiento a tres bandas. Eso es que el primer dispositivo que quiera hacer el establecimiento deberá enviarle un paquete informándole al otro que quiere enviar o recibir datos. A continuación el otro dispositivo, al recibir ese paquete, le comunicara que él también quiere hacer la conexión, además le enviara un comprobante de que ha llegado su paquete de establecimiento (ACK). Para acabar el primer dispositivo le enviará un nuevo ACK.

Además, en TCP siempre llegan los paquetes, es decir, ni un paquete de datos se pierde, se vuelve a enviar para que llegue al destinatario. Para saber si ha llegado o no, el destinatario debe enviar un paquete ACK. Si al cabo de un tiempo de enviar los datos, ese no recibe la confirmación de llegada, este reenviara los datos.

Para acabar TCP debe cerrar la conexión, para ello hace un cerramiento a cuatro bandas. Primero el dispositivo que quiera acabar la transmisión de datos deberá informal al otro enviándole un paquete de finalización. El otro dispositivo le enviara un ACK y seguidamente le enviara otro paquete informándole de que él también quiere acabar. Para acabar el primero enviará un nuevo ACK y se cerrará la transmisión de datos.

La ventaja de TCP es que siempre llegan los datos, así que es muy fiable.

La desventaja principal es que para enviar o recibir se necesita establecer una conexión, que a veces puede resultar un poco inútil, ya que puede suceder que los datos sean pequeños que todos los establecimientos hechos. Además, es bastante lento.

Este protocolo se usa cuando es necesario asegurar la fiabilidad. Como es el caso de hacer una petición HTTP.

### USER DATAGRAM PROTOCOL (UDP)

Según su RFC (Postel, 1980), UDP es un protocolo de capa cuatro. Por lo que debe poder gestionar el envío de los archivos.

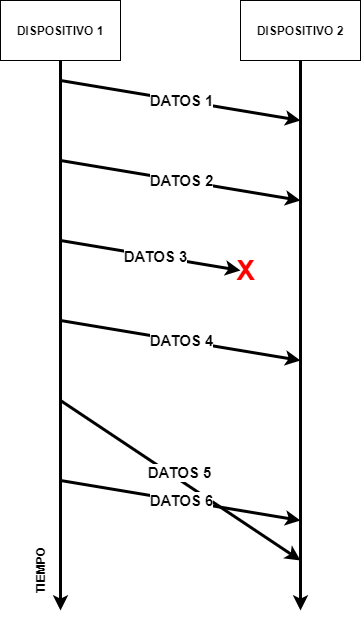


Figura 3.5: Secuencia de los paquetes en el protocolo UDP.

La diferencia principal que se puede observar es que, UDP, no es orientado a conexión. Por lo que el dispositivo que quiera enviar archivos lo podrá hacer con total normalidad, sin que el destinatario le dé permiso. En UDP el destinatario no envía ningún ACK.

Por lo que se puede ver en la figura anterior, puede suceder que por el camino se pierda un paquete de datos, puede suceder que un paquete llegue más tarde que uno más reciente. Esto pasa porque no hay ningún concepto de confirmación de paquetes.

La ventaja de UDP es que es muy rápido, ya que no es necesario hacer ningún establecimiento previo.

La desventaja principal es no es nada fiable y se debe asumir que se pueden perder paquetes, que lleguen desordenados o duplicados.

Este protocolo se usa cuando es no necesario asegurar la fiabilidad. Como es el caso de hacer una petición DNS.

## METODOLOGÍAS PARA EL DESARROLLO DE VIDEOJUEGOS EN LÍNEA

En el desarrollo de videojuegos hay tres metodologías para encarar el multijugador. No hay una que sea mejor que la anterior, pero hay alguna metodología que es más recomendable que otra, dependiendo del género del juego que se haga. Toda la información que se encontrará a continuación ha sido extraída del libro Development and Deployment of Multiplayer Online Games (Ignatchenko, 2017).

### CLIENTE-SERVIDOR

Cliente-Servidor es la metodología más usada en la actualidad. Aunque se ha usado en la mayoría de los multijugadores masivos en línea. Se caracteriza por tener un servidor, ubicado normalmente por regiones, y en él se conectan todos los jugadores de la partida, como se puede ver en la topología siguiente.

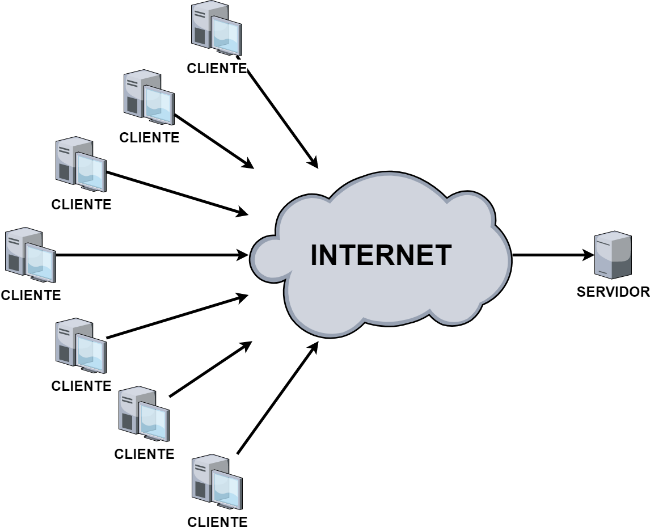


Figura 3.6: Topología cliente-servidor

El funcionamiento del cliente-servidor es sencillo, tanto el cliente como el servidor ejecutan una instancia del juego, cada vez que el cliente hace alguna acción en el juego, esta se envía al servidor y notifica a los demás jugadores de ello.

#### PROBLEMAS DEL CLIENTE-SERVIDOR

El problema principal del cliente-servidor es mucho más costoso, que el peer-to-peer. Ya que, como se verá en el siguiente subapartado, el p2p tiene ningún servidor dedicado. Según (Ignatchenko, 2017), más o menos cada servidor, dependiendo del juego, puede gestionar las partidas entre mil o cinco mil jugadores. El precio de cada servidor puede suponer unos mil euros mensuales. También se puede optar por hacer que un cliente haga también de servidor, a cambio de hacer el servidor no autoritario y por lo tanto más fácil de hacer trampas.

El segundo problema crucial es de rendimiento, ya que al hacerse que todas las acciones hechas por el jugador las gestione el servidor, hace que se deba enviar un paquete por internet a ese servidor. Por lo que tarda un tiempo en hacer esa acción. Por lo que se produce un input lag que es bastante grande, que ronda sobre los trescientos milisegundos. Al ser un problema muy común hay diferentes soluciones.

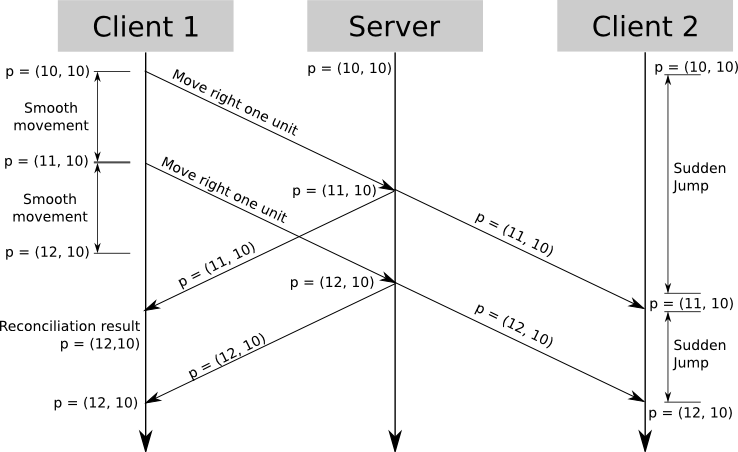


Figura 3.7: Secuencia de los paquetes en la topología cliente-servidor.

Como bien explica el señor Gambetta (Gambetta, s. f.), el cliente deber hacer una predicción de la acción, que viene a ser hacer una predicción del estado del juego en el servidor.

La segunda solución que debe hacer el cliente es reconciliarse con el servidor. Como se ve en la imagen anterior, cuando el jugador se ha movido dos unidades, el servidor le da el estado del servidor del primer movimiento. Así que el cliente debe analizar esos datos y ver si actualiza su estado del mundo o lo descarta.

La última solución es la interpolación de los demás jugadores. Esto se debe a que el movimiento hecho por el jugador tardara un tiempo en llegar a los demás jugadores. Así que el cliente debe interpolar las posiciones o acciones de los demás jugadores para que no haya saltos y se vea mal el juego.

### PEER-TO-PEER (P2P)

La otra gran metodología para los juegos multijugador en red es el peer-to-peer o p2p. Esta metodología se basa en que todos los clientes se comunican entre ellos sin la necesidad de un servidor de por medio.

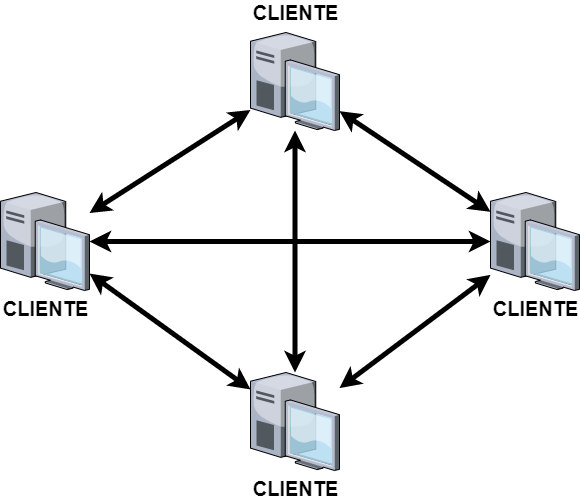


Figura 3.8: Topología p2p

Como se ve en la figura anterior, cada cliente se comunica con el que tenga que hacerlo para enviarle los datos del juego. Así que es una metodología bastante barata, ya que no requiere de tener alojado un servidor.

Pero hace que nuestro juego no sea seguro y sea muy fácil de hacer trampas en él.

### SERVIDOR-SERVIDOR

La metodología de servidor-servidor se usa para los juegos orientados a reglas. Estos son los juegos de casino, juegos sociales, etc. Normalmente son los juegos que no requieren de arte 3D.

Como es una metodología que para el juego que se desarrollara no es útil, tampoco se desarrollará su explicación.

## FUNCIONAMIENTO DE UN COCHE

En el juego el elemento principal es la conducción del vehículo y aunque no es un juego de simulación realista tendrá elementos muy parecidos al vehículo en la vida real. En esta sección no se explicará nada sobre la evolución de los coches, solo se centrará en explicar su funcionamiento de manera sencilla del motor, suspensión y dirección.

### MOTOR

El motor es uno de los ejes principales del coche, sin él el vehículo no se podría mover. El motor está compuesto de cilindros y pistones.

Lo que hace el motor es, mediante combustible, generar energía para que el vehículo pueda moverse. Para hacer esto el motor internamente debe hacer los siguientes movimientos.

Según (NSW Department of Education and Training, 2008), primero el pistón se comprime dentro del cilindro, mientras se comprime se inyecta el combustible dentro del cilindro. Cuando ya esté comprimido, se genera una explosión del combustible para que exista el movimiento, ya que al explotar el pistón baja, y entonces se vuelve a hacer el ciclo otra vez.

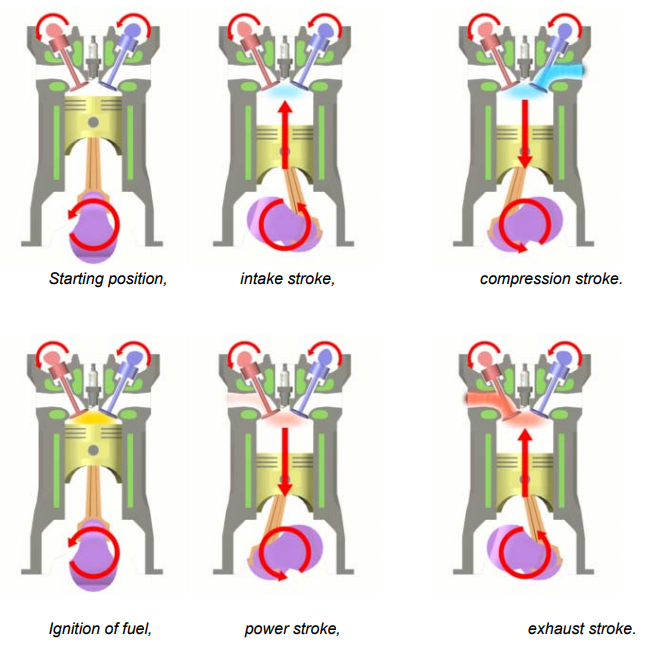


Figura 3.9: Ciclo del motor.

### SUSPENSIÓN

Otro pilar fundamental es la suspensión.

Como bien dice Konieczny y Burdzik (Konieczny y Burdzik, 2017), sin ella el vehículo solo podría ir plano, es decir que cualquier bache en la carretera o desnivel se podrían hacer daño los ocupantes del vehículo.

Para ello, la suspensión se basa en la ley física de Hook. Que su fórmula es la siguiente: , donde k es la constante elástica del muelle, x es la longitud del muelle aplicando la fuerza y x0 es la longitud del muelle sin aplicar la fuerza.

Por lo que cada rueda tiene un muelle y si hay algún desnivel, la suspensión balancea el vehículo para que no sea un problema para los ocupantes.

### DIRECCIÓN

La dirección del vehículo es lo que hace poder girarlo con el volante. Su sistema de giro es un poco complejo.

Según Schnubel (Schnubel, 2018), Primero el conductor debe girar el volante para que el vehículo gire. Cuando el conductor ha hecho esa acción un sistema formado por un unas bolas de metal y un eje hace que gire un engranaje y este hace que las ruedas delanteras, traseras o ambas giren.

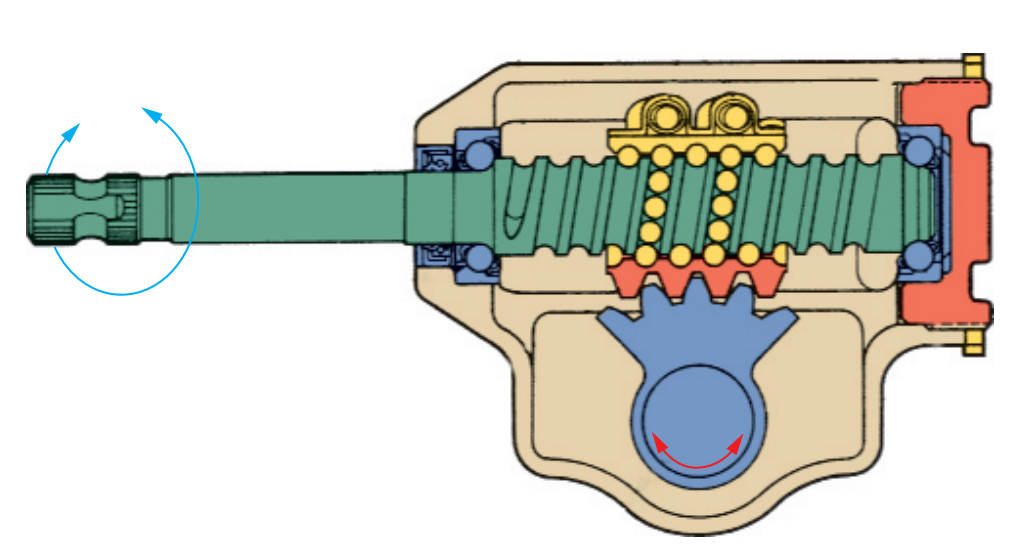


Figura 3.10: Sistema que hace que gire el vehículo.

### IMPLEMENTACIÓN EN EL JUEGO

Para la implementación de las físicas del vehículo en cualquier motor de juego se ha mirado un video de desarrollo de un juego llamado Obliteracers (Space Dust Studios, 2016). El video es una teórica de como esta empresa ha implementado las físicas del vehículo en su juego.

Este video empieza explicando la suspensión, que es el elemento principal de las físicas del vehículo. En él se explica que para simular una suspensión se debe hacer un raycast desde cada rueda del vehículo hasta el suelo. Para que este raycast pueda saber la distancia que tiene cada rueda con el suelo y recomienda guardar la posición donde colisiona, la normal y la ratio de compresión. Para poder hacer la simulación de la suspensión y que del efecto que está rebotando se usa la ley de Hook.

A continuación, explica cómo hacer el movimiento. Para hacer la velocidad menciona que se debe hacer con fuerzas en la dirección donde mira el vehículo y desde el suelo. Enseña que, si pones el centro de masa del vehículo en una posición bastante inferior, provocara que el vehículo, si vuelca, se dé la vuelta automáticamente. Otra mención interesante que hay es que, si se aplica la fuerza un poco hacia delante del vehículo, dará el feedback visual de acelerar o frenar.

Para acabar falta el giro, este giro se hace aplicando un par de torsión. Así que el vehículo, físicamente girará muy bien, pero no tendrá agarre en el suelo. Eso significa que parecerá que esté derrapando todo el rato. Así que para que el vehículo gire de manera adecuada se debe aplicar una fuerza inversa y proporcional a la dirección que se está girando.

# OBJETIVOS

Los objetivos que ahora se mencionarán se separarán en el objetivo principal y los secundarios. Los principales serán, los requisitos necesarios para que se pueda llegar a la conclusión de que el proyecto ha sido un éxito. Los secundarios serán los que ayudarán a saber si la conclusión del proyecto será sobresaliente o, por el contrario, suficiente.

## OBJETIVOS PRINCIPALES

1. Hacer que un multijugador usando Unreal Engine como motor de juego y Node.js como servidor.

Actualmente desarrollar un multijugador en línea usando Unreal Engine es bastante sencillo, ya que el mismo motor te proporciona todo tipo de ayudas para la gestión del juego en línea, como puede ser el servidor. Así que para que este trabajo fuese más completo se ha optado por diseñar el servidor independientemente del motor de juego y así aprender y buscar información de la gestión del juego y servidor.

1. Evitar y gestionar a los jugadores que hagan trampas.

Como se intentará hacer un servidor autoritario, el objetivo principal es que nadie haga trampas. Así que se tendrá mano dura para los jugadores que hagan trampas de manera reiterada.

## OBJETIVOS SECUNDARIOS

1. Hacer que la conducción sea divertida.

Este objetivo podría ser principal, pero se ha considerado que es secundario, ya que es muy subjetivo. Un sistema que sea divertido para alguien puede no serlo para otro. Aunque se intentará que la conducción sea divertida para la mayoría de gente que lo pruebe.

1. Hacer que los objetos estén bien balanceados.

La gracia de los juegos de conducción arcade, aparte de la conducción, son los objetos que se pueden lanzar a los jugadores. Si estos aparecen en una posición incorrecta o el poder que tiene es desproporcionado, el jugador sentirá frustración y dejará de jugar a nuestro juego. Así que, dentro de lo posible, se intentará que estos estén bien balanceados.

1. Tener un sistema para que el jugador siempre pueda jugar multijugador.

Se intentará que, cuando un jugador quiera jugar el modo multijugador, el juego le permita. Pero si no hay más jugadores o no hay los suficientes para empezar la carrera se añadirán bots.

# DISEÑO METODOLÓGICO Y CRONOGRAMA

En este capítulo se verán toda la metodología y planificación inicial que se tiene del proyecto. Para empezar, se verán cuáles son las herramientas del desarrollo del producto que se hará. A continuación, se explicarán los hitos que tiene el proyecto, así como la metodología y nomenclatura que se usará para elaborar el mismo. Para acabar se verá la planificación que se tiene del TFG y cuáles serán las fechas claves para que este proyecto salga adelante.

## HERRAMIENTAS DE DESARROLLO

Este proyecto, como ya se ha comentado varias veces, dispone de dos partes. Cada una de las partes tendrá unas herramientas distintas. Así que a continuación se verán cuáles son y por qué se han elegido esas herramientas.

### HERRAMIENTAS DE DESARROLLO DEL CLIENTE

El cliente lo primero que se debe saber es que motor de juego se usará. Unreal Engine 4 (Epic Games, 2014). La versión será la última que ha salido que es la 4.26 y el motivo principal es por motivaciones personales del autor de este TFG. Ya que la mejor manera de aprender C++ es entrar en el desarrollo de algún software con ese lenguaje. Así que el autor después de barajar la idea de usar Unity Engine (Unity Technologies, 2005). Se descartó usarlo Unity Engine, ya que, valoró muy positivamente las nuevas opciones que ofrece, o ofrecerá, Unreal Engine, como MetaHumans o su próxima versión Unreal Engine 5 que se espera para este mismo año.

La programación será completamente en C++, y es otro de los motivos que se apostó por Unreal Engine, y solo se usará la programación Blueprints en casos en que hacerlo en C++ resulte muy dificultoso. Por ejemplo, el HUD estará hecho en Blueprints, ya que es bastante más sencillo que hacerlo en C++.

El motivo principal de usar C++ como lenguaje principal es porque la mayoría de las ofertas de trabajo que hay en el sector piden este lenguaje como principal. Así que se ha pensado que este proyecto puede ser un buen ejemplo para lanzarse a utilizar ese lenguaje.

El IDE que se usará es una versión preview del IDE de la empresa JetBrains que se llama Rider. Es decir, Rider es un IDE que se usa para desarrollar elementos del framework .NET y además se puede usar para programar los scripts de Unity Engine. Hace realmente poco Rider sacó su versión de IDE para poder programar exclusivamente en C++ y orientado a Unreal Engine y actualmente se encuentra en modo preview. El motivo de usar Rider es porque esta empresa, JetBrains, tiene desarrollados muchos IDEs para distintos lenguajes de programación y es bastante fácil de usar y es sencillo familiarizarse con sus controles e interfaz.

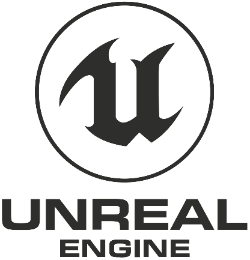


Figura 5.1: Logotipos de Unreal Engine y JetBrains Rider.

### HERRAMIENTAS DE DESARROLLO DEL SERVIDOR

Para programar el servidor, se usará básicamente un entorno de trabajo, que es Node.js. Node.js, según su página web (Dahl, 2009), es un entorno de trabajo basado en JavaScript y usado para programar servidores. Este entorno está basado en eventos asíncronos, así que para hacer un servidor para un juego se necesitará un complemento llamado Socket.io, que lo que ofrece es la gestión de eventos en tiempo real usando websockets. El motivo de usar este entorno es por su comunidad que tiene detrás y el soporte para aprendizaje que hay por la red.

Para acabar el IDE usado será también de la empresa JetBrains y en este caso es su IDE para JavaScript llamado WebStorm.

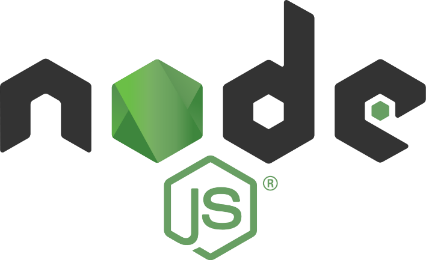


Figura 5.2: Logotipos de JetBrains WebStorm y Node.js.

## METODOLOGÍA

Se intentará usar una metodología ágil. Para ello en el proyecto de Git, se crearán unas ramas de master, develop¸y documentation como brancas por defecto. Después para cada tarea se le pondrá las siglas del hito a que corresponde, y a continuación el nombre de lo que se va a desarrollar. Por ejemplo, si se está haciendo el controlador del personaje y justamente se está haciendo el controlador del vehículo, el nombre de esa rama se llamaría c\_vehicleController.

Este trabajo debe separarse en dos partes principales. La primera es la elaboración de la documentación. Y la segunda es la creación y desarrollo del producto. Así que primero se explicará la metodología de la documentación.

### METODOLOGÍA DE LA DOCUMENTACIÓN

La documentación tiene previstas dos hitos principales, que son las entregas puestas antes de empezar este trabajo. Así que el primer hito es elaborar el anteproyecto. En él la metodología es simple, se trabajará dos horas diarias desde noviembre, mientras se solapa con el diseño y comienzo del desarrollo de la parte del cliente.

A continuación, está la entrega de la memoria final. Para poder explicar cómo se hará esta documentación, se debe saber que, una vez entregado el anteproyecto, la memoria final empezará su producción. Para ello se dedicarán dos horas semanales para el desarrollo de esta, hasta las dos últimas semanas antes de la fecha límite del hito. Esas dos semanas, se supondrá que el producto ya estará acabado, y por lo tanto se dedicaran las veinte horas para acabar de pulir y dejar perfecto este documento.

### METODOLOGÍA DEL PRODUCTO

El producto tendrá de tres hitos. Se entiende que la fase de preproducción del prototipo estará con el diseño del juego y está incluido en el primer hito.

El primero tiene fecha límite en el 28 de febrero, y se deberá tener hecho la parte del cliente. Así que este hito corresponde a ocho sprints, de una semana cada uno. Además, antes de poder hacer ese hito se debe haber hecho un pequeño GDD en donde se explique cuáles son las mecánicas y cómo funciona el juego.

El segundo hito, es de cinco sprints y corresponde con el servidor. La fecha límite es el 2 de abril y para ello se deberá tener hecho el servidor entero.

El tercer y último hito es de nueve sprints y en él se hará la integración entre el cliente y el servidor. Su fecha límite es el 4 de junio, y se debe tener hecho el multijugador del juego.

## PLANIFICACIÓN

A continuación, se mostrará la planificación del proyecto. Para poder elaborar el cronograma, se ha estimado cuanto tiempo se tardará para elaborar cada funcionalidad del proyecto.

|  |  | DOCS | | | DISEÑO | | | | CLIENTE | | | | | SERVIDOR | | | MULTIJUGADOR | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | M | P | GDD | UMLC | UMLS | ERD | CV | RJ | GUI | IA | O | BD | IJ | API | CAPI | MM | MJ | LO | SE | B |
| NOV | S1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DICIEMBRE | S2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ENERO | S7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FEBRERO | S11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MARZO | S15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S18 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S19 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ABRIL | S20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S21 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S22 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S23 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MAYO | S24 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S25 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S26 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S27 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| JUNIO | S28 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S29 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S31 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| S32 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| JUL | S33 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Tabla 5.1: Cronograma del TFG

Para que se pueda entender un poco el cronograma que se acaba de ver, las letras que se ven en la cabecera de la tabla corresponden a:

* Docs (Documentación)
  + A: Anteproyecto ≈ 90 h
  + M: Memoria ≈ 52 h
  + P: Presentación ≈ 35 h
* Diseño:
  + GDD: Game Design Document ≈ 10 h
  + UMLC: UML del Cliente ≈ 4 h
  + UMLS: UML del Servidor ≈ 6 h
  + ERD: Esquema relacional de la base de datos ≈ 4 h
* Cliente:
  + CV: Controlador del vehículo ≈ 15 h
  + RJ: Reglas del juego ≈ 15 h
  + GUI: Interfaz gráfica de usuario ≈ 15 h
  + IA: Inteligencia artificial ≈ 63 h
  + O: Objetos ≈ 30 h
* Servidor:
  + BD: Base de datos ≈ 24 h
  + IJ: Instancia del juego ≈ 72 h
  + API: Crear la API ≈ 24 h
* Multijugador:
  + CAPI: Comunicarse con la API ≈ 24 h
  + MM: Matchmaking ≈ 10 h
  + MJ: Movimiento de los jugadores ≈ 38 h
  + LO: Lanzar objetos ≈ 24 h
  + SE: Solucionar errores ≈ 96 h
  + B: Bots ≈ 15 h

# DESARROLLO

En este apartado se explicará toda la producción del producto que se elaborará en este TFG. Este apartado se ha separado en cuatro subapartados. Estos corresponden a las cuatro grandes tareas para hacer este prototipo, diseño, cliente, servidor e integración del cliente y servidor.

## DISEÑO

Antes de poder empezar con el desarrollo del juego, se debe saber que se hará. Así que se ha hecho un diseño de cómo será el juego, GDD y cómo será el servidor, diagramas UML y de la base de datos.

### GDD

Para poder hacer este desarrollo más ameno, se ha elaborado un pequeño GDD con el fin de tener diseñado con anterioridad los elementos del juego. En este GDD se han descrito, de manera principal, los objetos, su función y porcentaje de aparición.

#### OBJETOS

Estos objetos son los que harán el juego divertido, sin contar la conducción, ya que son los que dan aleatoriedad e incerteza. Para poder diseñar se han mirado los juegos analizados y se han adecuado a una temática. En este caso se ha cogido una temática de moda en el momento de la escritura de este documento que es la mitología nórdica.

Para poder explicar mejor los objetos se ha elaborado una tabla con el nombre del objeto, su función y porcentaje de aparición. Para que se entienda mejor la función de algunos objetos, el estado de “golpe”, es que el jugador afectado se parará por completo y no podrá volver a acelerar hasta que pasen tres segundos.

| OBJETO | FUNCIÓN | % APARICIÓN | | | |
| --- | --- | --- | --- | --- | --- |
| POS | % | POS | % |
| MJÖLNIR | Se lanza el martillo de Thor dirigido al vehículo que hay de delante. No hace falta que tengas visión del contrincante. Esto infringirá un estado de “golpe” al rival.  Parecido al Caparazón Rojo de Mario Kart | 1 | 25 | 7 | 15 |
| 2 | 40 | 8 | 15 |
| 3 | 35 | 9 | 10 |
| 4 | 30 | 10 | 5 |
| 5 | 25 | 11 | 5 |
| 6 | 20 | 12 | 5 |
| ESCUDO DE RUNAS | Se pone un escudo alrededor del vehículo, que hace que le hace inmune de los objetos de los rivales. Una vez activado solo se quita si usan un objeto contra él o el jugador lo utiliza. En el último caso el objeto solo se eliminará.  Parecido al Escudo de Fuerza de Crash Team Racing Nitro Fueled. | 1 | 60 | 7 | 0 |
| 2 | 20 | 8 | 0 |
| 3 | 15 | 9 | 0 |
| 4 | 10 | 10 | 0 |
| 5 | 5 | 11 | 0 |
| 6 | 5 | 12 | 0 |
| VELOCIDAD DE SLEIPNIR | Le da al jugador que lo usu un boost de velocidad durante tres segundos. Este boost será un 50% de la velocidad máxima que tenga en ese momento.  Parecida a la Seta de Mario Kart. | 1 | 5 | 7 | 20 |
| 2 | 15 | 8 | 15 |
| 3 | 20 | 9 | 10 |
| 4 | 25 | 10 | 5 |
| 5 | 25 | 11 | 5 |
| 6 | 25 | 12 | 5 |
| LANZAS DE LAS VALQUIRIAS | El jugador tendrá tres lanzas que podrá lanzar justo en línea recta hacia adelante. Si la lanza lanzada impacta en el escenario, será eliminada. En cambio, si da con algún rival, esta infringirá un estado de “golpe” al vehículo impactado.  Parecido al Cohete de Sonic Racing Team. | 1 | 5 | 7 | 10 |
| 2 | 15 | 8 | 10 |
| 3 | 20 | 9 | 10 |
| 4 | 15 | 10 | 5 |
| 5 | 15 | 11 | 5 |
| 6 | 10 | 12 | 5 |
| MANZANA DE IÐUNN | Durante diez segundos el jugador que use el objeto tendrá un boost de velocidad, protección y ataque. La velocidad subirá un 33% de su velocidad máxima. Le hará inmune de los objetos rivales. Y para acabar, si un rival colisiona con él, el rival recibirá un estado de “golpe”.  Parecido a la Estrella en Mario Kart. | 1 | 0 | 7 | 5 |
| 2 | 0 | 8 | 10 |
| 3 | 0 | 9 | 10 |
| 4 | 0 | 10 | 15 |
| 5 | 0 | 11 | 10 |
| 6 | 3 | 12 | 10 |
| TRUCO DE  LOKI | Durante diez segundos, los jugadores que haya delante del poseedor del objeto se le invertirán los controles. Así que de repente girar a la derecha será ir a la izquierda y acelerar será frenar y viceversa.  No se ha encontrado ningún objeto parecido. | 1 | 0 | 7 | 10 |
| 2 | 0 | 8 | 10 |
| 3 | 0 | 9 | 10 |
| 4 | 0 | 10 | 15 |
| 5 | 0 | 11 | 15 |
| 6 | 2 | 12 | 15 |
| OJO DE ODÍN | Una vez usado el objeto, el jugador se teletransportará aleatoriamente a alguno de los checkpoints que haya entre él mismo y el líder de la carrera.  Parecido al Cohete en Mario Kart. | 1 | 0 | 7 | 10 |
| 2 | 0 | 8 | 10 |
| 3 | 0 | 9 | 15 |
| 4 | 0 | 10 | 20 |
| 5 | 0 | 11 | 30 |
| 6 | 0 | 12 | 39 |
| MORDISCO DE JÖRMUNDGANDER | Los jugadores que vayan en las tres primeras posiciones recibirán el estado de “golpe”.  Parecido al Caparazón Azul en Mario Kart. | 1 | 0 | 7 | 10 |
| 2 | 0 | 8 | 15 |
| 3 | 0 | 9 | 20 |
| 4 | 5 | 10 | 25 |
| 5 | 5 | 11 | 20 |
| 6 | 10 | 12 | 15 |
| RUGIDO DE FENRIR | Durante diez segundos el rival que tengas delante y detrás reducirán su velocidad 25%. Además, si algún jugador choca con ellos, estos también se le reducirá la velocidad hasta que se les acabe el efecto.  No se ha encontrado ningún objeto parecido. | 1 | 5 | 7 | 20 |
| 2 | 10 | 8 | 15 |
| 3 | 10 | 9 | 15 |
| 4 | 15 | 10 | 10 |
| 5 | 25 | 11 | 10 |
| 6 | 25 | 12 | 5 |
| RAGNARÖK | Cuando es usado el objeto, se reiniciará la carrera desde la salida, pero en la última vuelta. La posición de los jugadores se invertirá. Así que, si el primero será el último, etc.  Solo podrá suceder una vez por carrera y será muy difícil de que salga.  No se ha encontrado ningún objeto parecido. | 1 | 0 | 7 | 0 |
| 2 | 0 | 8 | 0 |
| 3 | 0 | 9 | 0 |
| 4 | 0 | 10 | 0 |
| 5 | 0 | 11 | 0 |
| 6 | 0 | 12 | 1 |

Tabla 6.1: Tabla de los objetos y su probabilidad de aparición.

#### MECÁNICAS Y CONTROLES

Como se ha visto en el análisis de referentes, los juegos de este género tienen cinco mecánicas básicas. Se ha elaborado una tabla con las mecánicas, su función y el botón para usar esa mecánica.

| MECÁNICAS DEL VEHÍCULO | |  |
| --- | --- | --- |
| MECANICA | FUNCIÓN | BOTON[[2]](#footnote-2) |
| ACELERAR | El vehículo acelerará hasta llegar a la velocidad máxima. Si el jugador deja de acelerar, la velocidad, lentamente se irá reduciendo. | X |
| FRENAR | El vehículo reducirá su velocidad hasta que llegue a 0, si el jugador sigue frenando, el vehículo irá marcha atrás. En el caso de que el jugador acelere a la vez que frena, la velocidad se quedará en 0. | O |
| GIRAR | El vehículo efectuará un giro en un ángulo constante. Si esta más de dos segundos girando, el vehículo empezará a derrapar. | JOYSTICK  IZQUIERDO |
| DERRAPAR | El vehículo empezará a hacer un giro con incremento de ángulo de manera exponencial, hasta llegar a su máximo.  Cuanto más tiempo este derrapando, más velocidad tendrá cuando deje de derrapar. Lo que significa, se le dará un boost de velocidad de entre uno y dos segundos. Y el incremento del boost será de como máximo de un 33% de la velocidad máxima. | L1/R1 |
| REBUFO | Si se permanece justo detrás de un vehículo durante dos segundos, este otorgará una velocidad extra, de un 10% de la velocidad máxima.  Esa velocidad extra durará hasta que el jugador frene. Si el jugador gira o derrapa solo dejará el efecto siempre y cuando deje de estar detrás del vehículo. | - |
| USAR OBJETO | Si el jugador posee un objeto, usara ese objeto haciendo el efecto que tenga cada objeto. | L2/R2 |

Tabla 6.2: Tabla de las mecánicas del juego.

#### DIFICULTAD

A continuación, se explicará la dificultad. Esta se separa en dos partes, primero la dificultad de la IA y después esta la habilidad del jugador. La dificultad de la IA, son los rivales que tendrá el jugador para competir con el campeonato final. La dificultad de la habilidad es solo que eligiendo una de las tres dificultades, la velocidad del vehículo cambiará, yendo más lento en la dificultad más fácil y mucho más rápido en la difícil.

En el modo multijugador, solo habrá la dificultad de habilidad, ya que la intención es que, mayoritariamente, haya más personas que IA jugando.

A continuación, se mostrará una tabla con las tres dificultades que habrá en el juego.

|  |  |
| --- | --- |
| MODOS DE DIFICULTAD | |
| 50 KM/H | Solo hay un contrincante que lucha por la victoria. |
| 100 KM/H | Hay tres contrincantes que luchan por la victoria. Uno de ellos lo hace prácticamente todo perfecto. |
| 150 KM/H | Hay cinco contrincantes que luchan por la victoria. Tres de ellos lo hacen prácticamente todo perfecto. |

Tabla 6.3: Tabla de los modos de dificultad.

#### PUNTUACIÓN

Para que el juego tenga alguna satisfacción para el jugador, se tiene pensado en dar una puntuación, en función de la posición final. Esta puntuación es solo para las carreras multijugador en línea.

La puntuación se reparte con la siguiente manera:

| SISTEMA DE PUNTUACIÓN | |
| --- | --- |
| VOLANTE DE ORO | Se consigue cada vez que se gana una carrera. |
| VOLANTE DE PLATA | Se consigue cada vez que se quede segundo en una carrera. |
| VOLANTE DE BRONCE | Se consigue cada vez que se quede tercero en una carrera. |
| VOLANTE DE MADERA | Se consigue cada vez no se quede entre los tres primeros en una carrera. |

Tabla 6.4: Tabla del sistema de puntuación.

#### INTELIGENCIA ARTIFICIAL

La inteligencia artificial, o IA, está pensada, principalmente, para el juego solitario. Esta tendrá diferentes niveles de dificultad. La dificultad se determinará en sí esa IA, tiene la trazada óptima, hace uso bueno de los objetos, o si sabe derrapar bien.

La IA, tendrá cuatro estados, el IDLE, que prácticamente será una transición al estado de Start o Racing. El estado de Start, que será el que haga cada vez que tenga que hacer una salida de inicio, es decir, que empieza la partida. El estado de Racing, que es el estado principal, en él hará uso de todas las mecánicas del juego. Para acabar tendrá el estado de Hit, que es el estado de “golpe”, en este se bloqueará, para que la IA, no pueda hacer nada, hasta que acabe el estado.

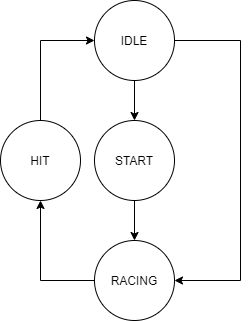


Figura 6.1: Diagrama de los estados de la Inteligencia Artificial.

### UML DEL SERVIDOR

Antes de explicar el UML del servidor, se hará una breve explicación de porque no hay un diagrama UML de clases del cliente. Ese motivo es porque como se generaría una tabla muy grande, se ha preferido separar este UML en las tareas que tiene el cliente. Así que cada trozo del diagrama se encontrará dentro de los sub apartados que haya en el sub apartado 6.2.

El servidor está diseñado para gestionar dos funciones.

Primero las llamadas a la base de datos. Estas llamadas solo se harán cuando el cliente lo pida, así que el servidor solo trabajará si el cliente se lo pide, solo para el aspecto de las llamadas de la base de datos. Estas llamadas se gestionarán a través de una API que se enviarán directamente del cliente.

La segunda función es que gestionara el juego en línea, como era evidente. Para ello, se creará un thread para cada carrera. Este thread será el que se comunicará con el cliente, es decir que recibirá los datos de cada cliente y reenviará esos datos a los clientes de nuevo, para así, que el jugador pueda verlo. Además, cada vez que se reciban los datos del cliente, se comprobará de que no haya ninguna anomalía, eso significa que se mirará que las velocidades del vehículo sean las adecuadas en cada momento, o que el jugador no se teletransporte. Esta segunda función, una vez se haya creado una sala de juego, funcionará automáticamente. Así que no será una funcionalidad “vaga” como la primera, sino que será el elemento principal del juego multijugador en línea.



Figura 6.2: Diagrama UML de clases del servidor.

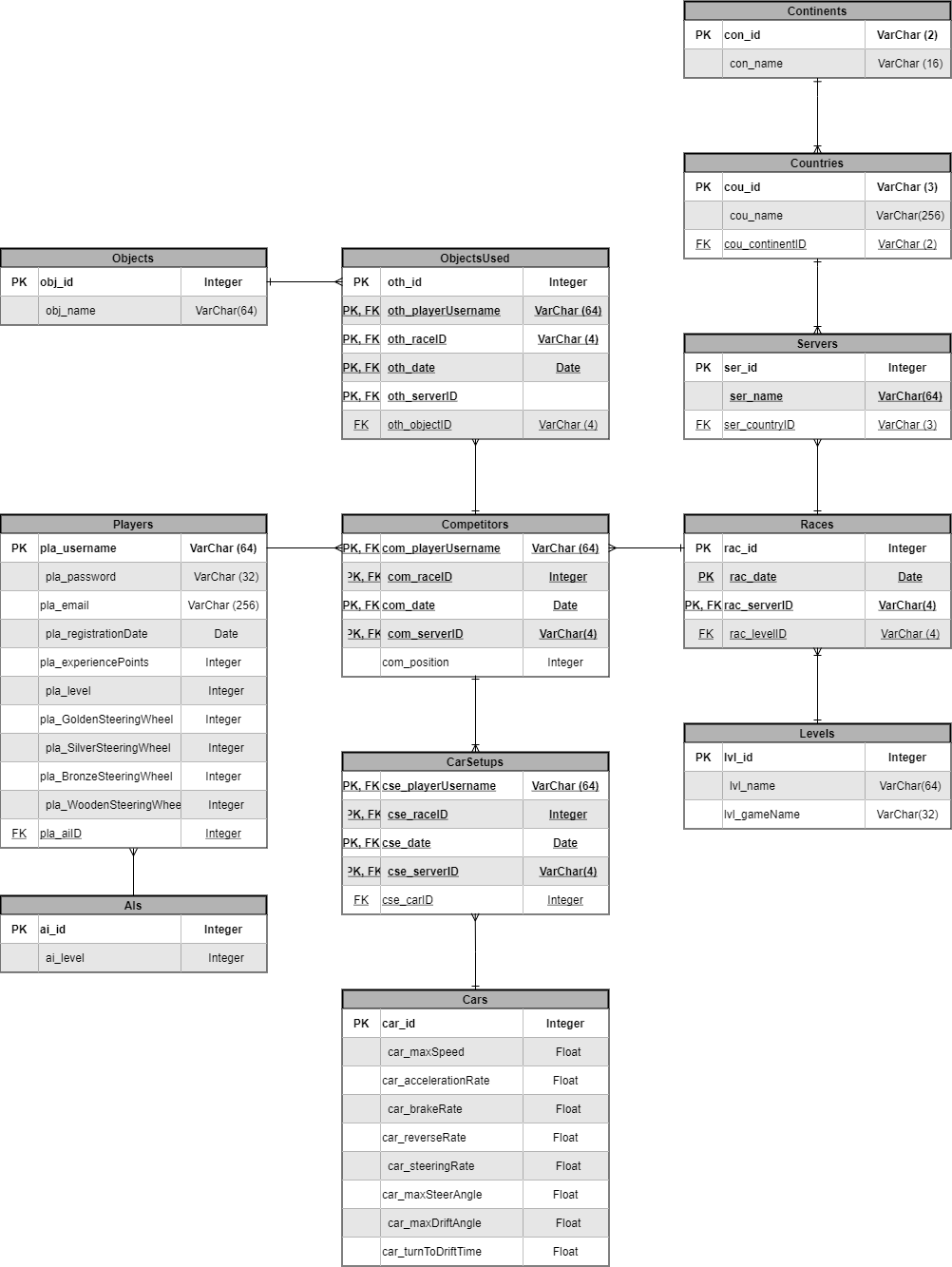
### BASE DE DATOS

Para poder gestionar bien el juego, se debe tener un control de los jugadores. Para ello se ha elaborado un diagrama relacional de la base de datos. El encargado de usar esta base de datos, como se ha visto en el apartado anterior, es el servidor.

Esta base de datos, aparte de controlar a los jugadores, evita, un poco, las trampas que puedan hacer los jugadores. Para ello, como se puede ver en la siguiente figura, hay tres tablas de control del juego. Esas tablas tienen guardadas las variables de los objetos, de las diferentes dificultades de las IA y de los diferentes vehículos que puede haber en el futuro en el juego.

Además, se guardará, en la base de datos, todos los objetos lanzados en una partida, el coche que hayan elegido y el circuito. Todos estos datos se guardarán una vez finalizada la carrera, ya que, si no se puede perder rendimiento del servidor, para hacer unas tareas que no son muy útiles en ese momento.

Para acabar, esta base de datos tiene constancia de que puede haber diferentes servidores que hay en el juego. Así que, si en un futuro el juego tiene diferentes servidores, no se debería cambiar mucho es parte de la base de datos.

Figura 6.3: Diagrama ERD de la base de datos del juego.

## CLIENTE

Cuando ya se ha diseñado el juego, se empezará a desarrollar el cliente del juego. Para ello hay una serie de tareas que hacer. Estas tareas estarán explicadas en subapartados

### CONTROLADOR DEL VEHÍCULO

El controlador del vehículo se ha desarrollado siguiendo las pautas del apartado 3.4.4. Es el elemento principal del juego, ya que sin un buen controlador el jugador no se divertirá jugándolo.

Este es el diagrama UML del controlador del vehículo.

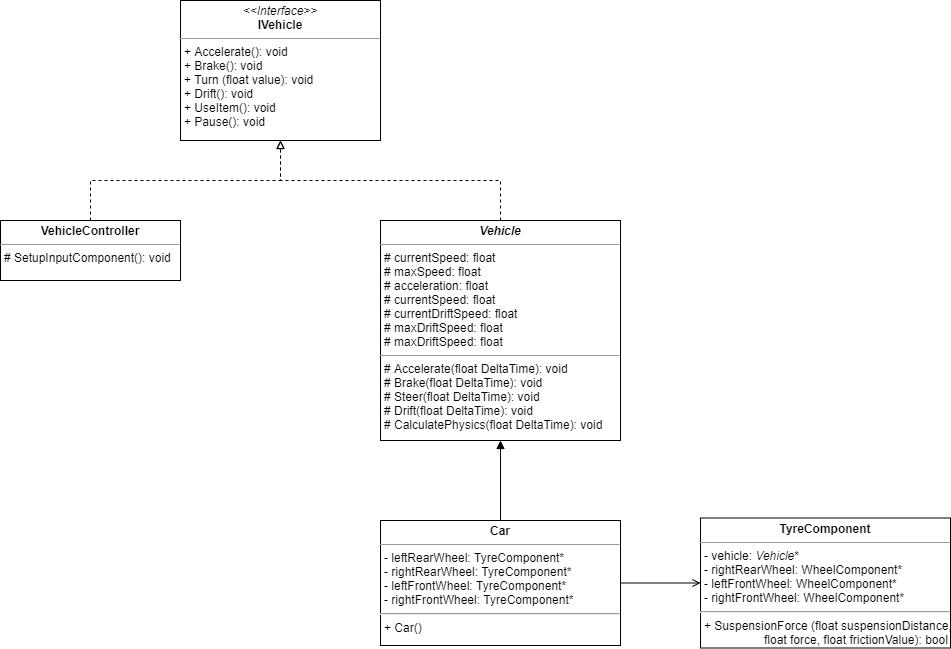


Figura 6.4: Diagrama UML del controlador del vehículo.

#### BLUEPRINT

Para empezar con el controlador se debe saber cómo será su jerarquía en el blueprint. Para ello se ha pensado que la mesh del vehículo, el chasis del coche, sea la raíz del blueprint. Así que después de tener el chasis, debe tener asignadas las posiciones y el componente de la rueda. Estas ruedas están separadas del chasis, no está en un skeletal mesh, ya que así era más sencillo de controlarlas y asignarles el componente de la rueda, que es el que hará la suspensión.



Figura 6.5: Jerarquía de los elementos del blueprint del controlador del vehículo.

#### TYRE COMPONENT

Seguidamente se ha empezado a programar y testear la suspensión. Para ello se ha creado el componente TyreComponent y se ha empezado ha hacer la suspensión usando la fórmula del muelle.

Para hacer eso se debe hacer un raycast, line trace en Unreal Engine, para saber si la rueda está tocando el suelo. Si lo está haciendo, se aplica la fórmula del muelle y se añade una fuerza hacia arriba de la rueda para que el vehículo tenga la sensación de que hay una suspensión. Si no toca el suelo, directamente no hace nada, reinicia el valor de la última suspensión y deja que la gravedad actúe.

El método que se ve a continuación requiere de que se pase la distancia máxima de la suspensión, la fuerza máxima que se puede aplicar, y la constante de la fricción, que hace que la suspensión sea más dura o blanda.

1. bool UTyreComponent::SuspensionForce(float suspensionDistance, float force, float frictionValue)
2. {
3. FHitResult hit;
4. FVector position = GetComponentLocation();
5. FVector end = position - (GetUpVector() \* suspensionDistance);
7. GetWorld()->LineTraceSingleByChannel(hit, position, end, ECC\_WorldStatic);
9. if (hit.bBlockingHit)
10. {
11. //FÓRMULA DEL MUELLE
12. float currentSuspensionCompression = 1 - (hit.Distance / suspensionDistance);
13. float springForce = currentSuspensionCompression \* force;
14. float frictionForce = (currentSuspensionCompression - suspensionCompression);
15. frictionForce /= GetWorld()->DeltaTimeSeconds;
16. frictionForce \*= frictionValue;
17. springForce += frictionForce;
19. //SE APLICA LA FUERZA
20. vehicleMesh->AddForceAtLocation(GetUpVector() \* springForce, position, NAME\_None);
22. suspensionCompression = currentSuspensionCompression;
24. return true;
25. }
26. else {
27. suspensionCompression = 0;
29. return false;
30. }
31. }

#### VEHICLE

Una vez ya funciona la suspensión se puede empezar a jugar con el movimiento del vehículo. Este movimiento será hecho exclusivamente con físicas. El vehículo está hecho con, al inicio, una interfaz, IVehicle. Esta interfaz lo que hace es tener los métodos comunes para la clase que gestiona los inputs, VehicleController, i la clase que ejecuta los inputs, Vehicle. La clase Vehicle, es una clase abstracta, y lo que hace es hacer los métodos comunes en todos los vehículos, menos las ruedas que pueda tener el vehículo, ya que en el futuro puede haber vehículos con menos ruedas, motos, bicis, triciclos, etc.

##### ACELERACIÓN

Para poder acelerar el vehículo, primero se debe tener un atributo de la velocidad máxima, maxSpeed, y el tiempo que tarda a llegar a ella, acceleration. Como se puede saber la aceleración no es lineal, sino que hace una curva exponencial, hasta llegar a la velocidad máxima. Así que cuando se añade la fuerza se hace el cálculo de esa aceleración utilizando una variable que hace de contador de tiempo y la función estática Exp de la clase Math.

Para que de un efecto visual de que el vehículo este acelerando, se ha hecho que la fuerza se ejecute un poco hacia adelante del centro de gravedad del chasis. Así cuando se acelere, el vehículo automáticamente hará el efecto de acelerar.

1. void AVehiclePawn::PerformAcceleration()
2. {
3. carMesh->SetLinearDamping(1.f);
4. if (currentSpeed < maxSpeed)
5. {
6. carMesh->AddForceAtLocation(
7. carMesh->GetForwardVector() \* acceleration \* FMath::Exp(accelerationTimer) \* accelerationRate, GetCenterOfMass());
8. lastVelocity = currentSpeed;
9. accelerationTimer += GetWorld()->DeltaTimeSeconds;
10. }
11. }

#### FRENO

El freno se ha hecho de la misma manera que la aceleración, es decir, el freno es una aceleración negativa. Así que se utiliza la fórmula exponencial para ejecutar esa aceleración. De esta manera, a diferencia de la aceleración, este método debe hacer una gestión de saber si debe frenar o si debe ir marcha atrás. Así que, si se sigue frenando y la velocidad ha llegado a cero, el vehículo directamente empezará a ir marcha atrás.

1. void AVehiclePawn::PerformBraking(float& currentVelocity)
2. {
3. carMesh->SetLinearDamping(1.f);
4. currentVelocity = lastVelocity < currentVelocity ? - currentVelocity : currentVelocity;
5. if (currentVelocity < maxSpeed / 20 && currentVelocity > reverseSpeed)
6. {
7. carMesh->AddForceAtLocation(-carMesh->GetForwardVector() \* acceleration \* FMath::Exp(reverseTimer) \* accelerationRate, GetCenterOfMass());
8. lastVelocity = currentVelocity;
9. brakeTimer = 0;
10. reverseTimer += GetWorld()->DeltaTimeSeconds;
11. }
12. else {
13. carMesh->AddForceAtLocation(-carMesh->GetForwardVector() \* acceleration \* FMath::Exp(brakeTimer) \* brakeRate, GetCenterOfMass());
14. lastVelocity = currentVelocity;
15. reverseTimer = 0;
16. brakeTimer += GetWorld()->DeltaTimeSeconds;
17. }
18. }

#### GIRO

El giro, en vez de añadir fuerzas, se usa una función llamada torque. Esta función solo rota el vehículo, así que el giro se debe hacer con la ayuda de la aceleración. Lo que hará que este giro pare, de la misma forma que se hace en el freno y la aceleración, es el valor fricción que ya da Unreal Engine. Así que para hacer un efecto de más derrape, se le asigna una fricción menor y para más agarre se puede asignar una fricción mayor.

Para acabar el vehículo cuando gira debe tener alguna velocidad, es decir, no puede girar con una velocidad de cero. Así que para ello el ángulo de giro máximo varía en función de la velocidad. Cuando más gira es cuando la velocidad es menor, ya que cuanto más lento vas más puedes girar un vehículo.

1. void AVehiclePawn::PerformSteering(float currentVelocity, float currentAngular)
2. {
3. driftValue = 0;
4. carMesh->SetAngularDamping(3.f);
5. carMesh->SetPhysicsMaxAngularVelocityInDegrees((maxSteerAngle + (maxDriftAngle - maxSteerAngle) \* (1 - (currentVelocity / maxSpeed))));
6. if (FMath::Abs(steerValue) == 0)
7. {
8. if (currentAngular <= maxSteerAngle \* frictionDecelerationRate)
9. {
10. lastTurnValue = 0;
11. }
12. }
13. else {
14. carMesh->AddTorqueInDegrees(carMesh->GetUpVector() \* steeringRate \* steerValue \* currentVelocity / maxSpeed,NAME\_None, true);
15. lastTurnValue = FMath::Sign(steerValue);
16. }
17. }

#### DERRAPE

El derrape sigue la misma lógica que el giro, pero añadiéndole menos fricción.

1. void AVehiclePawn::PerformDrift(float currentVelocity)
2. {
3. carMesh->SetAngularDamping(1.5f);
4. if (driftSign == 0 || FMath::Abs(currentVelocity) < maxSpeed / reverseRate || turnTimer >= turnToDriftSeconds && FMath::Abs(steerValue) <= 0)
5. {
6. isDrifting = false;
7. turnTimer = 0;
8. return;
9. }
10. carMesh->SetPhysicsMaxAngularVelocityInDegrees( CalculateMaxDriftValue());
12. carMesh->AddTorqueInDegrees(carMesh->GetUpVector() \* steeringRate \* driftSign, NAME\_None, true);
13. }

Aunque, a diferencia del giro, cuanto más aplicas el derrape hacia una dirección, derecha o izquierda, más girara con el tiempo. Para ello la velocidad de derrape máximo aumenta en el tiempo de manera exponencial y si se está aplicando la misma dirección que se quería al inicio. Es decir, si se empieza derrapando hacia la derecha, y se va girando el joystick hacia la derecha, empezara a girar más.

1. float AVehiclePawn::CalculateMaxDriftValue()
2. {
3. if (driftValue == 0)
4. driftValue = maxSteerAngle;
5. if (FMath::Abs(steerValue) >= 1 && FMath::Sign(steerValue) == driftSign)
6. {
7. driftTimer += GetWorld()->DeltaTimeSeconds;
8. driftValue += FMath::Exp(driftTimer);
9. }
10. else driftTimer = 0;
11. return driftValue >= maxDriftAngle ? maxDriftAngle : driftSign \* driftValue;
12. }

### REGLAS DEL JUEGO

Una vez acabado el controlador del vehículo, se empezó a programar las reglas del juego.

Este es el diagrama UML de las reglas del juego.

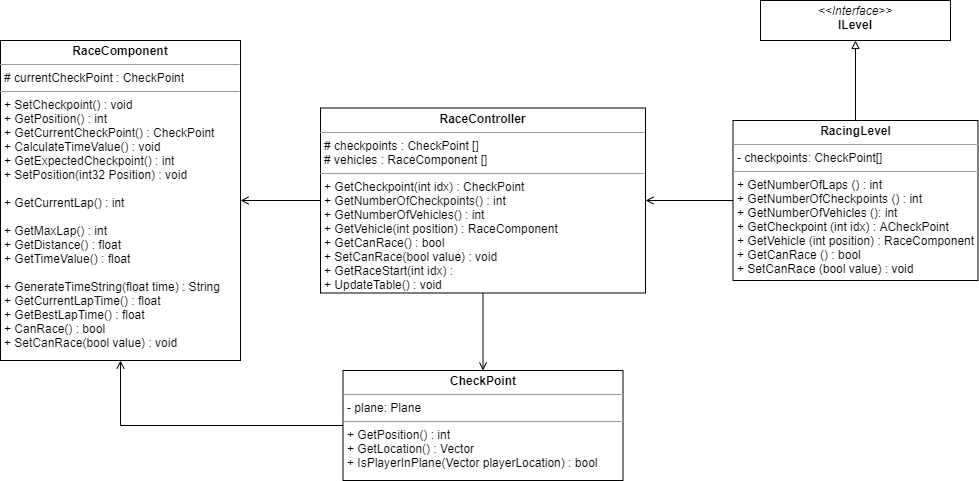


Figura 6.6: Diagrama UML de las reglas del juego.

#### CHECKPOINT

Lo primero que hay que tener en cuenta es que el vehículo debe recorrer todo el circuito, de momento no hay ningún sistema de atajos, para que cuente una vuelta.

Para poder hacer eso se ha optado por poner unos checkpoints, que el jugador no verá. Así que para que se pueda contar una vuelta se debe pasar por cada uno de los checkpoints.

De manera inicial estos checkpoints fueron implementados por triggers. Pero los triggers tienen un problema, y es que si durante el juego, tanto en línea como en solitario, el jugador, por problemas de rendimiento del ordenador, puede ser que, aunque siga la trazada del circuito, no se cuente el checkpoint, ya que el vehículo no ha pasado por encima del trigger.

Así que, para evitar problemas, se ha optado por que cada checkpoint tenga una variable que sea un plano. En el caso de Unreal Engine, un FPlane. Los planos no tienen el problema anterior, ya que tienen una función para saber si una posición en el mundo 3D esta delante o detrás del plano.

Para ello en el método BeginPlay, se ha creado el plano cogiendo la posición en el mundo y su dirección.

1. plane = FPlane(GetActorLocation(), GetActorForwardVector());

Una vez se tiene el plano, ya solo queda crear un método que llame a esa función mencionada anteriormente, PlayerDot.

1. bool ACheckPoint::IsPlayerInPlane(FVector playerLocation)
2. {
3. return plane.PlaneDot(playerLocation) < 0;
4. }

#### RACE CONTROLLER

Para poder gestionar la carrera, lo hace el actor RaceController. Aparte de otras cosas es el encargado de hablar con el nivel para saber las vueltas máximas, tener los checkpoints y vehículos que compiten en la carrera.

Otra función que tiene es generar la clasificación. Lo hace en función del tiempo, esta variable solo sirve para cuando un vehículo ha acabado la carrera y así poder generar el tiempo final. Si los vehículos no han acabado la carrera toma de referencia un valor que le da el mismo RaceComponent, llamado distancia. Para elaborar esta distancia tiene en cuenta la vuelta y el checkpoint que está y la distancia que tiene respecto al siguiente checkpoint.

1. void ARaceController::UpdateTable()
2. {
3. for (URaceComponent\* raceComponent : vehicles)
4. {
5. raceComponent->CalculateTimeValue();
6. }
7. URaceComponent\* aux;
8. for (int i = 1; i < vehicles.Num(); i++)
9. {
10. for (int j = vehicles.Num() - 1; j >= i; j--)
11. {
12. if (vehicles[j]->GetTimeValue() > vehicles[j - 1]- >GetTimeValue())
13. {
14. aux = vehicles[j];
15. vehicles[j] = vehicles[j - 1];
16. vehicles[j - 1] = aux;
17. }
18. else if (vehicles[j]->GetTimeValue() == vehicles[j - 1]->GetTimeValue())
19. {
20. if (vehicles[j]->GetDistance() < vehicles[j - 1]->GetDistance())
21. {
22. aux = vehicles[j];
23. vehicles[j] = vehicles[j - 1];
24. vehicles[j - 1] = aux;
25. }
26. }
27. }
28. }
29. for (int i = 0; i < vehicles.Num(); i++)
30. {
31. vehicles[i]->SetPosition(i + 1);
32. }
33. }

#### RACE COMPONENT

El tercer elemento para las reglas del juego es el Race Component. Dicha clase se encarga de generar el tiempo hecho en cada vuelta y, además, de comprobar con el próximo checkpoint si lo ha pasado o no.

Para hacer las funciones mencionadas en el párrafo anterior, se hace usando el método TickComponent, que es un método que se ejecuta en cada frame. Para hacer el contador de tiempo, se suma el valor del DeltaTime al contador total y al contador por vuelta. Para comprobar si se puede cambiar de checkpoint, se hace llamando el método IsPlayerInPlane que tiene el Checkpoint, si sale positivo, se hará el cambio.

1. void URaceComponent::TickComponent(float DeltaTime, ELevelTick TickType, FActorComponentTickFunction\* ThisTickFunction)
2. {
3. Super::TickComponent(DeltaTime, TickType, ThisTickFunction);
4. if (CanRace())
5. {
6. currentLapTime += DeltaTime;
7. generalLapTime += DeltaTime;
8. if (currentCheckPoint->IsPlayerInPlane( GetOwner()->GetActorLocation()))
9. SetCheckpoint();
10. }
11. }

### INTERFAZ GRÁFICA DE USUARIO

Para hacer la GUI, se ha elaborado dos blueprints específicos de la UI.

#### MENU PRINCIPAL

La primera interfaz que hay es el menú principal. El blueprint que gestiona eso se llama WBP\_MainMenu. En la parte del cliente, esta interfaz solo hace la función de entrar al modo de un jugador, así que más adelante se mostrará como se ha hecho el inicio de sesión en el apartado de la integración del cliente con el servidor, que es su función más importante.



Figura 6.7: Diseño de la GUI del menú principal.

#### IN GAME

Esta interfaz es la encargada de informar al jugador de los hechos que ocurren en la carrera. El blueprint en el que esta esa interfaz se llama WBP\_HUD. Así que en esta interfaz habrá:

* Posición: el número que está abajo a la izquierda de la pantalla
* Mejor tiempo de una vuelta: el rectángulo blanco, el que es más pequeño, que está arriba a la derecha. En ese momento pone “NO TIME”.
* Tiempo de la vuelta: el rectángulo naranja, el que es más grande, que está arriba a la derecha.
* Vueltas máximas: el cuadrado naranja, que es más pequeño, ubicado arriba a la derecha. En la imagen marca “2”.
* Vuelta actual: el cuadrado blanco, que es más grande, ubicado arriba a la derecha. En la imagen marca “1”.



Figura 6.8: Diseño de la GUI dentro del juego.

Para pasar los valores, mencionados anteriormente se ha usado la programación gráfica de Unreal Engine. Todos esos datos están en el componente RaceComponent que tiene cada vehículo.

Un ejemplo de un trozo del código de esa parte es la siguiente imagen. En ese método, se coge el blueprint en el que está el jugador, se coge el componente RaceComponent y para acabar se envía al HUD.

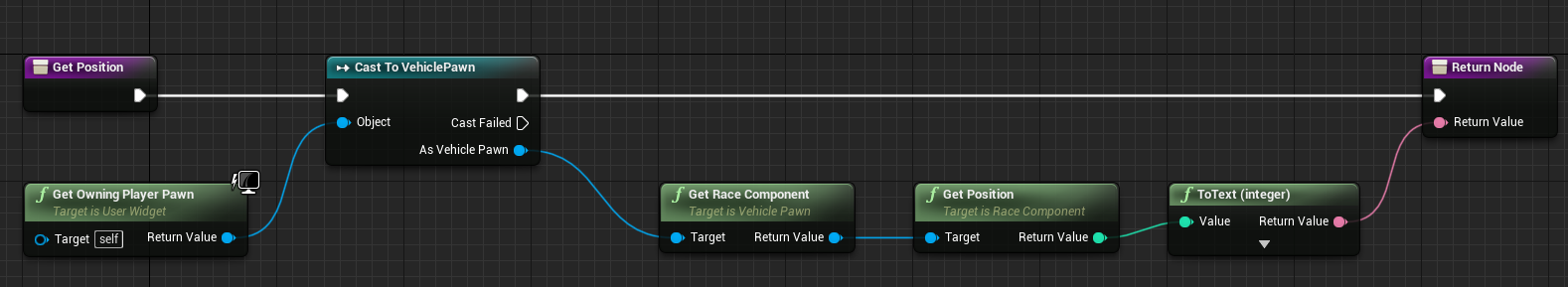


Figura 6.9: Método de GetPosition del WBP\_HUD.

### OBJETOS

Después del controlador del vehículo, los objetos es el elemento más importante del juego. Para ello se ha optado por crear una interfaz IObject, que tenga todos los métodos que deba tener cada objeto. Esa interfaz la implementa la clase abstracta ObjectBase, e implementará los métodos de forma común en todos los objetos. Para acabar de esa clase tendrá unas clases hijas que lo que harán será hacer las funcionalidades específicas de los objetos.

Este es el diagrama UML de los objetos.

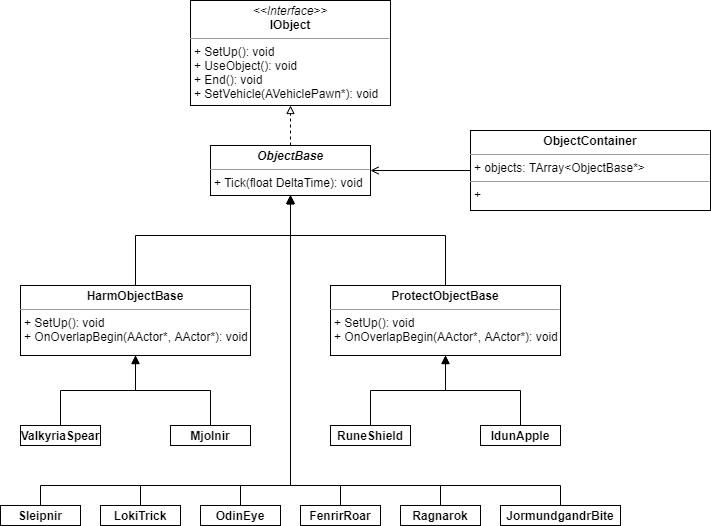


Figura 6.10: Diagrama UML de los objetos.

A continuación, se verán las implementaciones de las clases comunas de los objetos que pueden “herir” al jugador y de los que lo protegen.

#### HARM OBJECT BASE

La función que tiene este objeto es que cuando la mesh colisione con el jugador, le haga ponga el estado de golpe y se autodestruya. De esta manera los objetos como pueden ser las Lanzas de las Valquirias y el Mjölnir, solo se deben centrar en moverse. La lanza solo se centrará en hacer un movimiento hacia adelante y el Mjölnir a seguir el camino hasta llegar al jugador objetivo.

1. void AHarmObjectBase::SetUp()
2. {
3. Super::SetUp();
4. OnActorBeginOverlap.AddDynamic(this, &AHarmObjectBase::OnOverlapBegin);
5. }
7. void AHarmObjectBase::OnOverlapBegin(AActor\* ownerActor, AActor\* otherActor)
8. {
9. AVehiclePawn\* vehiclePawn = Cast<AVehiclePawn>(otherActor);
10. if (vehiclePawn)
11. {
12. if (used && vehiclePawn != vehicle)
13. {
14. vehiclePawn->Damage();
15. Destroy();
16. }
17. }
18. }

#### PROTECT OBJECT BASE

A diferencia del HarmObjectBase, esta clase tiene el objetivo de defender al jugador. Así que de la misma manera que el HarmObjectBase, si un objeta colisiona con el trigger de defensa, directamente destruye el objeto enemigo y hace la función que deba hacer los objetos como la Manzana de Iðunn o el Escudo de Runas.

La manzana directamente no hará nada si elimina un objeto, ya que ese objeto solo se elimina con el tiempo. En cambio, el escudo se autodestruirá si un objeto colisiona con él, ya que es de un solo uso.

1. void AProtectObjectBase::SetUp()
2. {
3. Super::SetUp();
4. OnActorBeginOverlap.AddDynamic(this, AProtectObjectBase::OnOverlapBegin);
5. }
7. void AProtectObjectBase::OnOverlapBegin(AActor\* ownerActor, AActor\* otherActor)
8. {
9. AHarmObjectBase\* object = Cast<AHarmObjectBase>(otherActor);
10. if (object)
11. {
12. object->Destroy();
14. UseObject();
15. }
16. }

### INTELIGENCIA ARTIFICIAL

#### RACING STATE

## SERVIDOR

Una vez acabado el cliente, se ha empezado a hacer el servidor del juego con Node.js. A continuación, se explicará cómo se ha hecho esta parte del desarrollo.

### BASE DE DATOS

Primero se ha configurado una base de datos con PostgreSQL. Así que se ha cogido el diagrama ERD de la base de datos y se ha convertido en unas comandas SQL. A continuación, se mostrará un ejemplo de la tabla Players.

1. DROP TABLE IF EXISTS Players CASCADE;
2. CREATE TABLE Players
3. (
4. pla\_username VARCHAR(64),
5. pla\_password VARCHAR(32) NOT NULL,
6. pla\_email VARCHAR(256) NOT NULL,
7. pla\_registrationDate DATE,
8. pla\_level INTEGER,
9. pla\_experiencePoints INTEGER,
10. pla\_goldenSteeringWheel INTEGER,
11. pla\_silverSteeringWheel INTEGER,
12. pla\_bronzeSteeringWheel INTEGER,
13. pla\_woodenSteeringWheel INTEGER,
14. pla\_aiID VARCHAR(4),
15. PRIMARY KEY (pla\_username),
16. FOREIGN KEY (pla\_aiID) REFERENCES AIs (ai\_id)
17. );

Una vez creada la base de datos, se ha empezado a crear el código en JavaScript que se comunique con la base de datos. Para poder hacer eso, se debe importar la función pg , que es la encargada de comunicarse con bases de datos PostgreSQL. Después se debe añadir los datos de la base de datos, que se han guardado en un script de configuración llamado DatabaseConfiguration.

1. const {Pool} = require('pg')
2. (...)
3. const pool = new Pool({
4. user: databaseConfig.user,
5. host: databaseConfig.host,
6. database: databaseConfig.database,
7. password: databaseConfig.password,
8. port: databaseConfig.port,
9. ssl: {
10. rejectUnauthorized: false
11. }
12. })

Una vez se tiene acceso a la base de datos ya se puede empezar a hacer llamadas a la base de datos.

Para ello se verán dos métodos, el que hace inicio de sesión y el que registra un jugador a la base de datos.

El método de inicio de sesión requiere que se le pase por parámetros el usuario y la contraseña. Lo que hará el método es buscar la contraseña del usuario en específico. Si encuentra una fila con el nombre de usuario deseado, devolverá la contraseña, se comprobará que sea igual a la contraseña recibida y si está correcto devolverá un código 202[[3]](#footnote-3).

Si ha ocurrido algún problema enviara un código 404 si no ha podido encontrar el usuario en la base de datos. Enviará un código 403 si la contraseña no es la correcta. Y si ha sucedido algún otro tipo de error, enviara un código 503.

1. exports.login = function (req, res) {
2. const {username, password} = req.body;
3. pool.query('SELECT pla\_password FROM Players WHERE pla\_username=$1', [username], (error, results) =>
4. {
5. if (error) {
6. res.sendStatus(404)
7. } else {
8. try {
9. if (results.rows[0]['pla\_password'] === password)
10. {
11. playersLoggedIn.push(username)
12. console.log(playersLoggedIn)
13. res.sendStatus(202)
14. } else
15. res.sendStatus(403)
16. } catch (e) {
17. res.sendStatus(503)
18. }
19. }
20. })
21. };

El siguiente método es para añadir un jugador en la base de datos. Para ello será necesario el nombre de usuario, la contraseña y el correo electrónico.

De la misma forma que el método anterior, la respuesta del servidor dependerá de si la operación se ha hecho correctamente o ha habido un error. En este caso, si se ha podido crear correctamente el jugador se enviará un código 201. Pero si ha habido algún problema se devolverá el código 401.

1. exports.register = function (req, res) {
2. const {username, password, email} = req.body;
4. let dataToInsert = [username, password, email, new Date().toISOString(), 1, 0, 0, 0, 0, 50];
6. pool.query('INSERT INTO Players VALUES($1, $2, $3, $4, $5, $6, $7, $8, $9, $10)', dataToInsert, (error) => {
7. if (error) {
8. res.sendStatus(401);
9. return
10. }
11. res.sendStatus(201);
12. })
13. };

### INSTANCIA DEL JUEGO

### API

Una vez ya está hecho todo el backend del servidor, solo queda que desde el cliente pueda comunicarse con el servidor. Para ello, la clase que es la raíz del proyecto, index, debe gestionar las llamadas a este servidor.

A continuación, se mostrará una tabla explicando cada ruta que hay la API, mostrando que parámetros de envío hay y cómo será la respuesta, si existe alguna. Todas las rutas que tengan al final dos puntos, significa que ese parámetro va incluido en la ruta, por ejemplo /:username/. En cambio, si no tiene ninguna notación parecida a la anterior, significa que esos parámetros van dentro de un JSON.

|  |  |  |  |
| --- | --- | --- | --- |
| **TIPO** | **RUTAS** | **PARÁMETROS DE ENVÍO** | **RESPUESTA** |
| POST | /login/ | Nombre de usuario | — |
| Contraseña |
| POST | /logout/ | Nombre de usuario | — |
| POST | /player/ | Nombre de usuario | — |
| Contraseña |
| Correo electrónico |
| GET | /player/:username/ | Nombre de usuario | Todos los datos del jugador |
|
| POST | /golden/ | Nombre de usuario | Valor actualizado |
| Valor |
| POST | /silver/ | Nombre de usuario | Valor actualizado |
| Valor |
| POST | /bronze/ | Nombre de usuario | Valor actualizado |
| Valor |
| POST | /wood/ | Nombre de usuario | Valor actualizado |
| Valor |
| POST | /race/add/ | Nombre de usuario | Identificador del websocket |
| Identificador del nivel |
| GET | /race/all/:username/ | Nombre de usuario | Todas las carreras hechas por el jugador |
| GET | /race/remove/:username/ | Nombre de usuario | — |
| GET | /car/:id/ | Identificador del coche | Todos los datos del coche |
| GET | /car/all/ | — | Todos los coches |
| GET | /ai/:id/ | Identificador de la IA | Todos los datos de la IA |
| GET | /ai/all/ | — | Todas las IA |
| GET | /object/:id/ | Identificador del Objeto | Todos los datos del objeto |
| GET | /object/all/ | — | Todos los objetos |
| GET | /level/:id/ | Identificador del nivel | Todos los datos del nivel |
| GET | /level/all/ | — | Todos los niveles |

Tabla 6.6: Tabla de la API.

Para poder implementar esas rutas en el servidor se necesita usar el framework express para Node.js. Este framework facilita las interacciones web de manera que sea muy sencillo generar una ruta para que gestione las peticiones GET o POST. Para poder hacer eso hay que crear los métodos correspondientes y justo al final del código, asignarle un puerto para que escuche el servidor. Además, al servidor se le debe decir que gestione JSON, ya que por defecto no lo hace.

1. const express = require('express'),
2. app = express(),
3. port = process.env.PORT || 3000,
4. bodyParser = require("body-parser");
6. app.use(bodyParser.json());
7. (...)
8. app.listen(port, () => {
9. playerDB.createServer()
10. console.log(`Server created at http://localhost:${port}`)
11. })

Para crear las rutas, sigue una forma parecida a la siguiente:

1. app.post('/api/login/', (req, res) => {
2. playerDB.login(req, res)
3. })

Se llama a la función post o get, se introduce la ruta, sin la primera parte del dominio, y después se pone los parámetros req (request), que es el JSON o datos de envío, y res, que es para enviar al cliente los datos que se necesiten. A continuación, ya solo queda poner la funcionalidad del servidor una vez un usuario acceda a esa ruta. En este caso se redirige al método correspondiente en la DatabaseController.

## MODO MULTIJUGADOR

Una vez ya se tiene hecho el cliente y el servidor, solo queda empezar a desarrollar el modo multijugador.

### COMUNICACIÓN CON EL SERVIDOR

Lo primero que se debe hacer establecer una conexión con el servidor. Para ello en este subapartado se ha desarrollado el inicio y cierre de sesión, el registro y el perfil.

Estas funciones se hacen en el menú principal, en la sección de multijugador. Así que lo primero que se hizo es diseñar esa parte del menú.

Figura 6.11: Diseño de la GUI del menú principal en el modo multijugador.

Para poder gestionar bien esta parte, se cambió la clase que heredaba el WBP\_MainMenu, por una clase propia, llamada MainMenuWidget. Así directamente se podía unificar el código en C++ con la programación gràfica.

A continuación se mostrará el incio de sesión en C++, que se verá como se hace la llamada a la API del servidor.

Para hacer una llamada a la API Unreal Engine necesita dos métodos. El primero es el encargado de hacer el inicio de sesión. El segundo es el que, una vez llegue la respuesta, podrà gestionar los datos recibidos.

El primer método, que es el que se verá a continuación, es el que hace las llamadas al servidor. Para ayudar ha hacer esas llamadas se ha creado una clase, HTTPCommunication, que será quien acabe de rellenar toda la URL y los datos que queden para que el envío sea correcto.

Así que en este método, solo hará falta crear el JSON, para enviarle al servidor y enviar la petición por la ruta especifica.

1. void UMainMenuWidget::Login(FString username, FString password, bool save)
2. {
3. responseCorrect = false;
4. responseRecieved = false;
6. FString json;
7. FLoginStruct loginStruct;
8. loginStruct.username = username;
9. loginStruct.password = password;
10. FJsonObjectConverter::UStructToJsonObjectString(FLoginStr uct::StaticStruct(), &loginStruct, json, 0, 0);
12. TSharedRef<IHttpRequest, ESPMode::ThreadSafe> Request = FHttpModule::Get().CreateRequest();
13. Request->OnProcessRequestComplete().BindUObject(this, &UMainMenuWidget::OnResponseReceived);
15. controller->SendPost(Request, "/api/login", json);
16. }

La segunda clase, solo mira cuál ha sido el código de respuesta del servidor y activa un atributo bool.

1. void UMainMenuWidget::OnResponseReceived(FHttpRequestPtr Request, FHttpResponsePtr Response, bool bWasSuccessful)
2. {
3. if (controller->IsSuccessful(Response->GetResponseCode()))
4. {
5. responseCorrect = true;
6. }
7. responseRecieved = true;
8. }

Una vez hechos esos métodos, se ha programado una función en el blueprint del menú principal. Lo importante que tiene esa función es que, al ser una llamada asíncrona, se debe esperar a que lleguen los datos. Para hacer eso se ha hecho una espera activa, así que, cuando ya se ha enviado la petición y no se haya recibido una respuesta, se esperará 0.1 segundos, así continuamente hasta que llegue la respuesta.

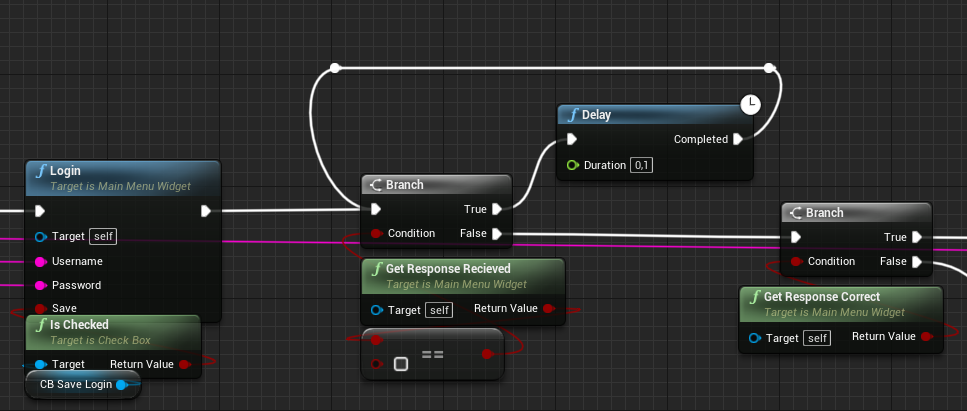


Figura 6.12: Sección del código del método Login de WBP\_MainMenu.

Las otras funcionalidades son bastante simétricas, ya que no reciben ningún dato, solo informan al servidor. Aunque la única diferencia que hay es en el método que recibe los datos cuando se pide un perfil.

Así que, en vez de usar el método anterior, OnResponseRecieved, usa el siguiente.

Ese método, aparte de hacer la función del método anterior, crea un struct con los datos del jugador y los guarda. Ya que de momento esos datos solo lo puede consultar el propio jugador.

1. void UMainMenuWidget::OnResponseUserReceived(FHttpRequestPtr Request, FHttpResponsePtr Response, bool bWasSuccessful)
2. {
3. if (controller->IsSuccessful(Response->GetResponseCode()))
4. {
5. responseCorrect = true;
6. TArray<FPlayerStruct> data;
7. FJsonObjectConverter::JsonArrayStringToUStruct(Response-> GetContentAsString(), &data, 0, 0);
9. if (data.Num() > 0)
10. {
11. playerLoggedIn = new FPlayerStruct(&data[0]);
12. }
13. }
14. responseRecieved = true;
15. }

### MATCHMAKING

### MOVIMIENTO DE LOS JUGADORES

### LANZAR OBJETOS

### PREDICCIÓN, RECONCILIACIÓN E INTERPOLACIÓN

### BOTS

# BIBLIOGRAFÍA

Activision. (2004). *Call of Duty*. https://www.youtube.com/watch?v=PrvX0sanW\_E

Beenox. (2019). *Crash Team Racing Nitro Fueled*. https://www.youtube.com/watch?v=51fqnxwQ1kg

Dahl, R. L. (2009). *Node.js*. https://nodejs.org/en/

Daleske, J., y Warner, S. (1973). *Empire*. https://www.youtube.com/watch?v=Yo8mrNPr\_3o

Epic Games. (2014). *Unreal Engine 4*. https://www.unrealengine.com/en-US/

Epic MegaGames y Digital Extremes. (1998). *Epic Game’s Unreal*. https://www.youtube.com/watch?v=WTRmZ6aFEWM

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., y T. Berners-Lee. (1999). *Hypertext Transfer Protocol—HTTP/1.1*. Internet Official Protocol Standards. https://tools.ietf.org/html/rfc2616

Gambetta, G. (s. f.). *Fast-Paced Multiplayer*. https://www.gabrielgambetta.com/client-server-game-architecture.html

Glazer, J. y Sanjay Madhav. (2016). *Multiplayer Game Programming*. Addison-Wesley.

ID Software. (1993). *Doom*. https://www.youtube.com/watch?v=iFnOLFd\_ByQ

Ignatchenko, S. (2017). Development and deployment of multiplayer online games. ITHare.com.

ISO/IEC JTC 1. (1994). 7. Detailed description of the resulting OSI architecture. En *ISO/IEC 7498-1:1994* (1st Edition, p. 59). https://www.iso.org/standard/20269.html

Jim Bowery. (1974). *Maze War*. https://www.youtube.com/watch?v=9wX5L03r-qA

Konieczny, Ł., y Burdzik, R. (2017). Modern suspension systems for automotive vehicles and their test methods. *Vibroengineering PROCEDIA*, *14*, 233-237. https://doi.org/10.21595/vp.2017.19238

Mockapetris, P. (1987). *Domain Names—Implementation and Specification*. Internet Official Protocol Standards. https://tools.ietf.org/html/rfc1035

Mojang. (2011). *Minecraft*. https://www.youtube.com/watch?v=srd17aD7P3Y

Naughty Dog. (1999). *Crash Team Racing*. https://www.youtube.com/watch?v=OTLREpMM9tE

Nintendo. (1992). *Super Mario Kart*. https://www.youtube.com/watch?v=AlAmXXNz5ac

Nintendo. (2013). *Mario Kart 8*. https://www.youtube.com/watch?v=BR0l84gjEXg

NSW Department of Education and Training. (2008). *Inspect & Service Cooling Systems*.

Postel, J. (1980). *User Datagram Protocol*. Internet Official Protocol Standards. https://tools.ietf.org/html/rfc768

Postel, J. (1981). *Transmission Control Protocol*. Internet Official Protocol Standards. https://tools.ietf.org/html/rfc793

Psyonix. (2015). *Rocket League*. https://www.rocketleague.com/

Schnubel, M. (2018). Classroom manual for automotive suspension & steering systems (Seventh edition). Cengage Learning.

Shuler, R. (2002). *How Does the Internet Work?* https://web.stanford.edu/class/msande91si/www-spr04/readings/week1/InternetWhitepaper.htm

Space Dust Studios. (2016). *Obliteracers*. https://www.youtube.com/watch?v=LG1CtlFRmpU

Square Enix. (2004). *Final Fantasy: Crystal Chronicles*. https://www.youtube.com/watch?v=pOA0U6yWJqo

Sumo Digital. (2019). *Team Sonic Racing*. https://www.youtube.com/watch?v=s7I\_SI4pRzw

Unity Technologies. (2005). *Unity Engine*. https://unity.com/es

Universidad de Illinois. (1960). *PLATO*. https://distributedmuseum.illinois.edu/exhibit/plato/

William Higginbotham. (1958). *Tennis for Two*. https://www.youtube.com/watch?v=s2E9iSQfGdg

1. El paquete está muy simplificado. En verdad estaría compuesto de valores binarios. [↑](#footnote-ref-1)
2. Los botones tendrán como referencia el mando de Play Station [↑](#footnote-ref-2)
3. El código mencionado corresponde a los códigos del estado del protocolo HTTP. Siempre son tres dígitos. Si el primero empieza por 1, significa que es una información. Si empieza por 2 significa que el envío ha sido exitoso. Si empieza por 3 significa que es de redirección. Si empieza por 4 es porque ha habido un error en el cliente. Para acabar si empieza por 5 significa que ha sido un error del servido. [↑](#footnote-ref-3)