



*ugr* | Universidad  
de Granada

## TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

# Videojuego de carreras de coches

---

Videojuego de carreras, enfocado en Inteligencia Artificial

Autor

Julio Antonio Fresneda García

Tutor

José Manuel Benítez Sánchez



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación

---

Granada, junio de 2019

**Palabras clave:** Inteligencia Artificial, IA, videojuego, coches, carrera, red neuronal, algoritmos genéticos

## Resumen

El proyecto presentado es un videojuego de carreras de coches, donde la mayor parte del desarrollo ha sido relativo a la inteligencia artificial.

El juego debe proporcionar suficiente variedad de contextos (circuitos, trazados, tipos de vehículos, ...). Uno de los aspectos prioritarios en el diseño del videojuego será la incorporación de modelos adaptativos de Inteligencia Artificial para hacer atractivo para jugadores con niveles distintos de habilidad del usuario.

**Keywords:** Artificial Intelligence, AI, videogame, car, race, neural network, genetic algorithm

## Abstract

The project presented is a videogame of racing cars, where most of the development has been relative to artificial intelligence.

The game must provide a variety of contexts (circuits, routes, types of vehicles, ...). One of the priority aspects in the design of the game will be the incorporation of adaptive models of Artificial Intelligence to make it attractive for players with different levels of user ability.

Yo, **Julio Antonio Fresneda García**, alumno de la titulació de grado de Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicació de la Universidad de Granada**, con DNI 49215154F, autorizo la ubicació de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

D. **José Manuel Benítez Sánchez**, Profesor del grado de del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Videojuego de carreras de coches: Videojuego de carreras enfocado en la Inteligencia Artificial*, ha sido realizado bajo su supervisión por **Julio Antonio Fresneda García**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

# Agradecimientos

A mi familia por el apoyo y paciencia.

A mis amigos, por interesarse y darme ánimos en la recta final.

A mi tutor, por todos los consejos y ayuda recibida.

# Contenido de la memoria

1	Introducción .....	9
1.1	En qué consiste este Trabajo de Fin de Grado .....	9
1.2	Motivación .....	9
1.3	Objetivos .....	10
1.4	Estructura de esta memoria.....	10
2	Preliminares .....	13
2.1	Historia de los videojuegos .....	13
2.1.1	Inicios .....	13
2.1.2	La eclosión de los videojuegos, década de los 70s.....	14
2.1.3	La década de los 8 bits: Los 80s.....	14
2.1.4	La revolución de las 3 dimensiones: Década de los 90s.....	15
2.1.5	Época actual: Desde el 2000 hasta ahora .....	16
2.2	I.A. en los videojuegos.....	16
2.2.1	Historia del diseño de la I.A. en los videojuegos .....	16
2.2.2	IA teórica vs IA en videojuegos .....	18
2.3	Videojuegos independientes: Contexto .....	19
3	Ánalisis de requisitos.....	21
3.1	Estudio de viabilidad .....	22
3.2	Obtención y análisis de requisitos.....	22
3.2.1	Descubrimiento de requisitos .....	23
3.2.2	Clasificación y organización de requisitos .....	24
3.3	Especificación de requisitos .....	26
3.4	Validación de requisitos .....	36
4	Planificación y metodología .....	39
4.1	Módulos del videojuego.....	39
4.2	Planificación temporal.....	40
4.3	Metodología .....	41
4.3.1	Unity .....	41
4.3.2	Autodesk 3DS Max .....	42
4.3.3	Adobe Photoshop CC.....	42
4.3.4	Programación .....	43
5	Diseño y desarrollo.....	45
5.1	Gestión de escenas.....	46
5.2	Mecánicas de control del vehículo.....	47
5.2.1	Partes del vehículo .....	48

5.2.2	Características de cada modelo .....	49
5.2.3	Controlador de vehículo.....	51
5.3	Modelado .....	55
5.3.1	Coches .....	56
5.3.2	Circuitos o mapas .....	57
5.3.3	Escenas .....	58
5.3.4	Otros objetos.....	60
5.3.5	Texturas y materiales .....	61
5.4	Interfaces de Usuario .....	62
5.4.1	Botones .....	62
5.4.2	Selección de marca y modelo.....	63
5.4.3	Minimap .....	64
5.4.4	Velocímetro .....	64
5.4.5	Posición de carrera, y número de vuelta .....	64
5.5	Sonido.....	64
5.6	Programación del videojuego .....	64
5.6.1	Menú principal (Gestión de perfiles).....	65
5.6.2	Seleccionar Circuito y Campeonato .....	66
5.6.3	Seleccionar Coche .....	66
5.6.4	Programación en las carreras.....	66
6	Inteligencia Artificial.....	69
6.1	Conceptos básicos .....	69
6.1.1	¿Qué es la Inteligencia Artificial? .....	69
6.1.2	¿Qué es una red neuronal? .....	70
6.1.3	¿Qué es un algoritmo genético? .....	72
6.2	Inteligencia Artificial en el vehículo .....	74
6.2.1	Inteligencia Artificial para el giro del vehículo .....	75
6.2.2	Inteligencia Artificial para el freno y boost del vehículo.....	78
6.3	NEAT .....	79
6.3.1	Introducción .....	79
6.3.2	Codificación genética .....	80
6.3.3	Rastreando genes usando marcas históricas (Número de innovación) .....	81
6.3.4	Protegiendo las innovaciones a través de la especiación .....	82
6.4	Entrenamiento de la Red Neuronal.....	84
6.4.1	Implementación de NEAT.....	84
6.4.2	Uso del algoritmo en nuestro problema .....	84

6.4.3	Hiperparámetros .....	87
6.4.4	Resultados .....	88
7	Jugando al videojuego.....	89
7.1	Iniciando el videojuego .....	89
7.2	Perfiles.....	90
7.3	Carrera Rápida.....	91
7.4	Campeonato .....	94
8	Conclusiones .....	97
9	Bibliografía y referencias.....	99



# 1 Introducción

## 1.1 En qué consiste este Trabajo de Fin de Grado

Este trabajo es un videojuego de carreras de coches con un enfoque arcade. Puesto que soy de la rama de Computación y Sistemas Inteligentes, la parte más destacable de este proyecto es la inteligencia artificial.

## 1.2 Motivación

Hacer un videojuego no es una tarea sencilla. Durante el desarrollo de un videojuego hay que abarcar y tocar muchos campos, algunos totalmente distintos (aparentemente) del mundo de la informática. Programación, modelado, interfaces, diseño del videojuego, sonido, son sólo algunas de las piezas que se necesitan en el desarrollo de un videojuego.

Para tener la motivación necesaria para desarrollar un videojuego, no basta con que te guste la programación. Tampoco con que te guste modelar piezas en tres dimensiones. Para desarrollar un videojuego, necesitas tener motivación e interés en todas y cada una de las partes que componen dicho videojuego, porque si dejas alguna sin trabajar, el videojuego estará cojo, incompleto.

Por supuesto con un equipo de personas con distintos perfiles se puede dividir la tarea, donde nadie tiene que ser experto en todo. Pero en este caso, el desarrollo de un TFG, debo ser yo quien abarque cada una de los campos (o al menos intentarlo, con más o menos éxito).

En mi caso, tengo motivaciones de distinto origen (pero todas útiles) para cada una de las partes de un videojuego. ¿Y qué motivación me lleva a entrar en este mundo? Es un clásico el típico chaval que entra en la carrera de Ingeniería Informática porque le apasionan los videojuegos, los ha jugado desde pequeño, y quiere pasar de jugarlos a hacerlos. Quiere formar parte de este mundo, y trabajar en la empresa que desarrolla su videojuego favorito. No es mi caso.

En mi caso, siempre me ha interesado el desarrollo de videojuegos, pero por otras razones. Hay dos principales motivaciones que han desembocado en un interés por este tema, y curiosamente son las mismas que me empujaron a entrar a esta carrera.

La primera motivación es mi interés por las ciencias exactas. En esto incluyo desde las matemáticas que se usan en el desarrollo del videojuego, hasta la programación. La programación me gusta por la misma razón que las matemáticas, la algorítmica o el ajedrez, en el fondo consisten en resolver un puzzle donde los movimientos o son precisos, sin ambigüedad y tienen sentido, o el puzzle no se resuelve.

Mi segunda motivación es totalmente opuesta, y es mi interés por las tareas que se pueden resolver de forma creativa. Soy malísimo dibujando, a sí que tengo que buscar otras aficiones que me permitan ser creativo, y la informática y el desarrollo de videojuegos son dos de ellas. Pocas ramas (si no ninguna) de la informática es más creativa que la de desarrollar videojuegos, si es que puede considerarse a esto rama de la informática.

Tanto los videojuegos, como la Informática en sí, son la intersección perfecta entre estas motivaciones.

El diseño de un nivel, el aspecto de los personajes, la historia de trasfondo, cómo se mueven los elementos, y todos los casi infinitos aspectos de un videojuego constan de dos fases: Imaginárselos y diseñarlos, los cuales dependen de tu creatividad, y desarrollarlos, que dependen de la programación y el resto de herramientas.

Para imaginarlos y diseñarlos no hay algoritmos ni técnicas exactas, y eso me atrae. Para desarrollarlos sí que hay algoritmos y técnicas exactas, y eso me atrae.

### 1.3 Objetivos

Si no fuese consciente de mis limitaciones, el objetivo sería claro: Desarrollar un videojuego completo listo para publicar. Sin embargo, dado que el desarrollo de un videojuego de calibre puede llevar años, tendré que rebajar mis expectativas.

En el caso de este Trabajo de Fin de Grado, el objetivo no es crear un videojuego completo, si no la base de la cual podría salir ese videojuego soñado. Un punto de partida donde se toquen cada uno de los ámbitos antes mencionados que tiene un videojuego, en la medida de lo posible. Que sea jugable evidentemente, que sea entretenido, y que refleje al menos las ganas e ilusión que tengo por este mundo.

Ya que el tiempo y recursos son limitados, y no puedo desarrollar en profundidad cada una de las partes que componen un videojuego, voy a centrarme en la que mejor se me da (o eso creo), y la que supongo que a un informático más le interesa: La Inteligencia Artificial.

Por tanto, el objetivo de este trabajo de fin de grado será crear un prototipo de videojuego donde la parte que más brille, y la que más en profundidad se tocará, sea la Inteligencia Artificial del videojuego.

### 1.4 Estructura de esta memoria

En los siguientes capítulos de la memoria, se explicará todo lo relacionado con el trabajo de fin de grado.

En el capítulo 2: “Preliminares”, se explicará el punto de partida en el que estamos cuando queremos desarrollar un videojuego. Desde el contexto de los videojuegos a día de hoy, hasta la historia de la I.A. enfocada en este campo, pasando por las herramientas que se han usado en este proyecto.

En el capítulo 3: “Planificación y Metodología”, se explicará la planificación que he seguido a lo largo de este proyecto, y en qué medida se ha cumplido esto, así como la metodología usada.

En el capítulo 4: “Análisis de requisitos”, se verán qué requisitos debe cumplir nuestro videojuego, de forma que el producto final cumpla con unas expectativas deseadas.

En el capítulo 5: “Jugando al videojuego”, se muestra el videojuego en sí y cómo jugarlo.

El capítulo 6: “Diseño y Desarrollo”, será junto con el siguiente, el capítulo más extenso con diferencia, pues se explicará a fondo en qué consiste el videojuego, sus modos de juego, su mecánica, y por supuesto, cómo se ha hecho todo esto. En este capítulo debería ir la inteligencia artificial, pero por su notable importancia en este trabajo, se explicará en el siguiente capítulo.

El capítulo 7: “Inteligencia Artificial en el videojuego”, contendrá todo lo relacionado con la inteligencia artificial usada en el videojuego, tanto su funcionamiento como los métodos usados, así como los problemas a los que me he enfrentado, entre otras cosas.

En el capítulo 8: “Conclusiones”, se plasmarán mis pensamientos finales sobre el proyecto, una vez acabado.

La memoria acabará con un capítulo de bibliografía, donde se citarán las fuentes de todo el material usado, tanto intelectual como imágenes, texturas y sonidos, entre otras cosas.



# 2 Preliminares

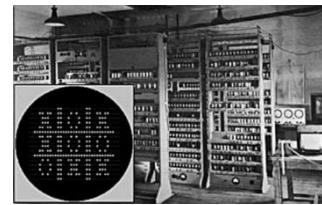
Antes de comenzar a explicar el trabajo realizado, es necesario poner un poco de contexto. Porque a quien esté leyendo esta memoria y no esté relacionado con el mundo de los videojuegos, le vendrá bien saber desde qué punto parte este trabajo.

## 2.1 Historia de los videojuegos

### 2.1.1 Inicios

Los videojuegos, al ser una temática tan extensa y profunda, tiene tantas definiciones que es difícil decidir cuál fue el primer videojuego.

[1] Se podría considerar que el primer videojuego fue el “OXO”, o “Nought and crosses”, desarrollado por Alexander S.Douglas en 1952. Este videojuego no era otra cosa que un tres en raya, donde el jugador competía contra la máquina. Este videojuego se ejecutaba sobre la EDSAC, la cual fue el primer calculador electrónico en el mundo en contar con órdenes internas, aunque no la primera computadora con programas internos (ese honor le corresponde a la [SSEM](#)).



OXO, primer videojuego creado



Tennis for Two

Poco más adelante, en 1958, William Higginbotham creó Tennis for Two, el cual era un simulador de ping pong. Este videojuego se creó gracias a un programa de cálculo de trayectorias y un osciloscopio, y su propósito fue el entretenimiento de los visitantes de la exposición Brookhaven National Laboratory.

A diferencia que OXO, en este videojuego no intervenía la máquina, si no que jugaban dos jugadores humanos.

El primer videojuego en tener relativo éxito, fue Spacewar. Steve Russell, estudiante del instituto de Tecnología de Massachusetts, dedicó seis meses en crear este videojuego, usando gráficos vectoriales. El videojuego consistía en dos naves espaciales que luchaban entre ellas, donde dos jugadores controlaban su dirección y velocidad. El videojuego se ejecutaba sobre un PDP-1, y tuvo relativo éxito en el ambiente universitario.



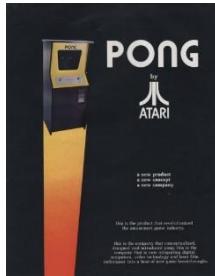
Spacewar



Magnavox Odyssey

Pero el comienzo de los videojuegos domésticos no comenzó hasta que Ted Dabney empezó a desarrollar un proyecto de videojuego llamado Fox and Hounds, dando inicio al videojuego doméstico. Este proyecto evolucionó hasta convertirse en la Magnavox Odyssey, el primer sistema doméstico de videojuegos, el cual se conectaba a la televisión y permitía jugar a varios juegos pregrabados. Este producto salió a la venta en 1972.

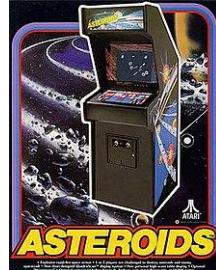
### 2.1.2 La eclosión de los videojuegos, década de los 70s



Anuncio de Pong

Tanto Space War como Tennis for Two tuvieron sus versiones comerciales. Computer Space fue una versión de Space War desarrollada por Nolan Bushnell en 1971, y Pong, la máquina recreativa basada en Tennis For Two, comenzó la ascensión de los videojuegos. El sistema fue diseñado por Al Alcom, en la entonces poco conocida Atari.

Pong se presentó en 1972 y fue la base de la industria de los videojuegos. Durante los años siguientes, con los avances de la época en tecnología (como microprocesadores y chips de memoria) se implantaron numerosos avances técnicos en los videojuegos, dando lugar a la aparición de juegos como Space Invaders o Asteroids en los salones recreativos.



Anuncio de Asteroids

### 2.1.3 La década de los 8 bits: Los 80s

La popularidad de los salones de máquinas recreativas y las primeras videoconsolas de los 70 llevaron a un fuerte crecimiento en el sector de los videojuegos. En cuanto a videoconsolas, aparecieron sistemas como Odyssey 2 (Phillips), Intellivision (Mattel), Colecovision (Coleco), Atari 5200, Commodore 64, TurboGrafx (NEC). En cuanto a máquinas recreativas, triunfaron juegos como el famoso Pacman (Namco), Battle Zone (Atari), Pole Position (Namco), Tron (Midway) o Zaxxon (Sega).



Máquina recreativa de Pacman

Mientras tanto, en Japón, iba a nacer la que ha sido una de las mayores referencias en el mundo de los videojuegos. La empresa de naipes, Nintendo, dio un giro hacia los videojuegos lanzando la Famicom, o Nintendo Entertainment System (NES), en 1983.



Primera NES (1983)

Los norteamericanos adoptaron la NES como su principal sistema de videojuegos, y a lo largo de la década fueron apareciendo nuevos sistemas domésticos, como la Master System (SEGA), el Amiga (Commodore) y el 7800 (Atari), con el ahora clásico Tetris.

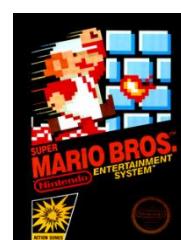
A finales de los 80, comenzaron a aparecer consolas de 16 bits como la Mega Drive de Sega, y mientras tanto los microordenadores fueron lentamente sustituidos por las computadoras personales basadas en arquitecturas de IBM.

En 1985 apareció un punto de inflexión en el desarrollo de videojuegos: Super Mario Bros. Mientras que en la mayoría de juegos anteriores la mecánica era repetir en bucle las mismas pantallas hasta obtener la máxima puntuación posible, el videojuego desarrollado por Nintendo supuso un estallido de creatividad. Por primera vez había un objetivo y una meta en un videojuego. Fue un cambio de estilo, y en años posteriores otras compañías seguirían esta corriente.



Game Boy

En el campo de las recreativas, Japón pasó a ser la mayor productora, y aparecieron videojuegos como Defender, Rally-X, Dig Dug, Bubble Bobble, Gauntlet, Out Run o Shinobi.



Uno de los primeros Super Mario Bros

Por primera vez aparecieron los videojuegos portátiles, pero su evolución definitiva no llegaría hasta que en 1989 se lanzó la Game Boy, de Nintendo.

#### 2.1.4 La revolución de las 3 dimensiones: Década de los 90s

A principios de los años 90 las videoconsolas dieron un importante salto técnico gracias a la competición de la llamada "generación de 16 bits" compuesta por la Mega Drive, la Super Nintendo Entertainment System de Nintendo, la PC Engine de NEC, conocida como TurboGrafx en occidente y la CPS Changer de (Capcom).

Esta generación explotó la cantidad de jugadores, y gracias a los avances técnicos de la época, como la aparición del CD-ROM, hubo una importante evolución dentro de los diferentes géneros de videojuegos.

Diversas compañías comenzarían a desarrollar videojuegos con entornos tridimensionales, principalmente en el campo de los PC, apareciendo obras como Doom. Las consolas de 16 bits empezaron a considerarse antiguas, y su último logro se produciría gracias a la SNES mediante la tecnología 3D de pre-renderizados de SGI, con videojuegos como Donkey Kong Country y Killer Instinct. La competencia de Nintendo, la Mega Drive lanzó el primer videojuego poligonal en consola, el Virtual Racing, que tuvo gran éxito y marcó un antes y un después en los juegos 3D de consola.



Mega Drive

Los videojuegos 3D, dado su gran éxito, se hicieron un importante hueco en el mercado, principalmente gracias a la generación de consolas de 32 bits: Sony PlayStation y Sega Saturn, y la generación de consolas de 64 bits: Nintendo 64 y Atari Jaguar.



Play Station 1

La famosísima PlayStation de Sony, originalmente fue un proyecto en colaboración con Nintendo, denominado SNES PlayStation, el cual consistía en un periférico para SNES con lector de CD. Al final Nintendo rechazó la propuesta, puesto que Sega desarrolló un proyecto parecido con un éxito muy pobre, y Sony se vio obligado a lanzar la PlayStation de forma independiente, con un resultado que nadie se esperaría.

Conforme las consolas y ordenadores aumentaban en demanda, los arcades comenzaron con un lento pero imparable declive.

En el mundo de los videojuegos portátiles, producto de las nuevas tecnologías más poderosas, comenzaron su verdadero auge, uniéndose a la Game Boy máquinas como la Game Gear (Sega), Linx (Atari) o la Neo Geo Pocket (SNK), aunque ninguna pudo hacerle frente a la popularidad de la Game Boy, siendo esta y sus descendientes (Game Boy Pocket, Game Boy Color, Game Boy Advance, Game Boy Advance SP) las dominadoras del mercado.



Game Boy Advance

El final de milenio se acercaba, y la consola más popular era la PlayStation con videojuegos como Final Fantasy VII (Square), Resident Evil (Capcom), Winning Eleven 4 (Konami), Gran Turismo (Polyphony Digital) y Metal Gear Solid (konami).

En PC eran muy populares los FPS (juegos de acción en primera persona) como Quake (id Software), Unreal (Epic Megagames) o Half-Life (Valve), y los RTS (juegos de estrategia en tiempo real) como Command & Conquer (Westwood) o Starcraft (Blizzard). Además, conexiones entre ordenadores mediante internet facilitaron el juego multijugador, convirtiéndolo en la opción predilecta de muchos jugadores, y fueron las responsables del nacimiento de los MMORPG (juegos de rol multijugador online) como Ultima Online (Origin). Finalmente en 1998 apareció en Japón la Dreamcast (Sega) y daría comienzo a la "generación de los 128 bits".

### 2.1.5 Época actual: Desde el 2000 hasta ahora

En el año 2000, Sony lanzó la que sería la consola más vendida de la historia, con más de 155 millones de copias: La clásica PlayStation 2.



Microsoft también entró en escena sacando la Xbox, en 2001. Nintendo lanzó la Gamecube, sucesora de Nintendo 64, y la Game Boy Advance. Ante tal competencia, Sega anunció que ya no produciría hardware, convirtiéndose sólo en desarrolladora de software en 2002.

PlayStation 3, 4, Xbox 360, One, Nintendo DS, 3DS... la cantidad de videoconsolas en el mercado desde los 2000 hasta el día de hoy es inmensa, al igual que la cantidad de videojuegos. El salto en calidad estas dos últimas décadas ha sido estratosférico. Sin embargo, la máquina que más calidad soporta, la más utilizada y la que mejores especificaciones tiene a día de hoy, no es una videoconsola: El ordenador personal.

El ordenador ha llegado a un punto el cual no sólo permite jugar los videojuegos más exigentes desarrollados, si no también crearlos. Gracias al ordenador personal, la comunidad de jugadores es más grande y unida que en toda la historia.



Ordenador Personal

## 2.2 I.A. en los videojuegos

Ya que en este trabajo de fin de grado se da gran importancia a la parte de la inteligencia artificial, no está de más recordar el camino de la I.A. en los videojuegos, desde sus humildes inicios hasta el día de hoy.

### 2.2.1 Historia del diseño de la I.A. en los videojuegos

Los videojuegos nacieron sin IA. Los primeros videojuegos, OXO, Tennis for Two, Spacewar, etc no tenían ningún tipo de inteligencia artificial. Y es que, durante los primeros pasos de los videojuegos, la IA no es una característica necesaria. Esto se debe a que esos juegos eran relativamente simples y, durante la mayor parte del tiempo, se jugaba persona contra persona, no contra la máquina.

En 1970, Atari lanzó su primer videojuego, "Computer Space". Y no fue hasta entonces que los diseñadores de juegos comenzaron a realizar su primer intento de incorporar IA en sus juegos. En esta época, las inteligencias artificiales se diseñaron principalmente para juegos arcade con el fin de garantizar que las personas siguieran jugando el máximo de tiempo posible en las recreativas. Pong, Space Invaders y Donkey Kong estuvieron entre los primeros videojuegos también. Estos juegos se ejecutaban bajo reglas muy simples y acciones guionizadas. Los agentes no tenían la capacidad de tomar decisiones. A veces, las decisiones fueron diseñadas para tomarse al azar, de modo que los comportamientos parecían más impredecibles. Por lo tanto, la llamada Inteligencia fue en realidad precompilada en el juego y no podía actuar en tiempo de ejecución. Por tanto, la primera IA apareció en forma de patrones almacenados. Un ejemplo de una IA tan rígida es el diseño de alienígenas en "Space Invaders". En este juego, el jugador debe disparar a los alienígenas antes de que alcancen la parte inferior de la pantalla. La forma en que se mueven estos alienígenas estaba precompilada en el juego. Tenían un patrón almacenado.

Se crearon muchos más videojuegos basados en este tipo de IA, pero la evolución de la IA acababa de comenzar. La aparición del oponente de la computadora en "Pong" hizo que la gente creyera que la computadora estaba pensando. También fue considerado como la primera inteligencia artificial real en los juegos. La forma en que se desarrolla el juego Pong hace que sea imposible precompilar los comportamientos de objetos no humanos. Las raquetas deben tomar decisiones basadas en las acciones de los jugadores humanos. Las decisiones pueden no ser tan difíciles de tomar: es un cálculo simple de dónde deben ir las raquetas, pero esto hizo que las personas experimenten la misma sensación de jugar contra un jugador humano real.

La influencia que tuvo el diseño de la IA en PacMan es tan significativa como la influencia del videojuego en sí. Este clásico videojuego de arcade hace que el jugador crea que los enemigos en el juego lo están persiguiendo, pero no de una manera burda. Los fantasmas están persiguiendo al jugador (o evadiéndolo) de una manera diferente, como si tuvieran una personalidad individual. Esto le da a la gente la ilusión de que realmente están jugando contra 4 o 5 fantasmas individuales en lugar de copias de un mismo enemigo de computadora.

A finales de la década de 1980, la locura de los videojuegos arcade comenzaba a desvanecerse. Con el desarrollo de la industria de la computación, los ordenadores y consolas domésticos lideraron la nueva dirección del desarrollo de videojuegos. Los nuevos juegos diseñados para estos dispositivos se volvieron más complejos debido a la mayor capacidad de los procesadores modernos. A pesar de que se deben aplicar más recursos a los contenidos y al rendimiento gráfico de mayor calidad, la IA todavía tiene su propia necesidad de desarrollar. Durante la década de 1980, surgieron rápidamente más géneros de juegos. El viejo estilo de diseño de IA estaba desactualizado. Los diseñadores tuvieron que tratar seriamente la IA de los juegos, ya que la IA se ha convertido en una característica estándar de los videojuegos.

Entre los nuevos tipos de juegos, los juegos de estrategia estimularon el desarrollo de la inteligencia artificial debido a que, en estos tipos de videojuegos, la IA es clave. La IA es más importante para los juegos de estrategia y contribuye más al contenido del juego que otros tipos de juegos tales como juegos de rompecabezas o juegos de rol. El juego de estrategia en tiempo real (RTS) se introdujo a finales de los 80 como un nuevo género. Una IA altamente competente y entretenida era lo que este género de juego ofrecía a los jugadores. Pero tal IA era desafiante, por lo que tenía requisitos exigentes. Desde entonces, el diseño de la IA en los juegos de estrategia en tiempo real se ha convertido en una tarea propia y desarrollado como un nuevo campo de investigación.

Los juegos de mesa siempre han sido un clásico en cuanto desarrollo de IA. En 1997, como se explicará en el siguiente apartado, DEEP BLUE consiguió superar al por aquel entonces campeón del mundo Gary Kasparov. En 2016, AlphaGo, desarrollado por Google DeepMind, consiguió derrotar al 18 veces campeón del mundo Lee Sedol, por 4 a 1.

Otros tipos de videojuegos también requerían un desarrollo propio de IA. Half-Life de Valve Software ha recibido grandes elogios por su diseño de IA en el campo de disparos en primera persona. SimCity fue el primero en probar el potencial de los enfoques de vida artificial ("A-LIFE"). Los videojuegos de deportes y carreras también necesitaban de una IA propia, con peculiaridades propias del género.

Pero el desarrollo de IA no ha llegado a su límite. Los videojuegos han recorrido un largo camino desde la década de 1950, al igual que las técnicas de inteligencia artificial que los acompañan. Los últimos años han sido testigos de más y más ideas novedosas, y por tanto también los métodos para los videojuegos que la IA ha unido al proceso de desarrollo del juego.

### 2.2.2 IA teórica vs IA en videojuegos

Normalmente, un programador de IA no necesita una cantidad significativa de conocimiento de Inteligencia Artificial en el campo académico para crear IA de videojuegos de alta calidad. La razón radica en dos hechos. En primer lugar, la inteligencia artificial es una gran rama de la informática. No es fácil adaptar los modelos complejos a los videojuegos. En segundo lugar, el objetivo principal de la IA del videojuego hoy en día es entretener a la gente. Hay trucos u otras formas fáciles de resolver los problemas perfectamente. La gente no estará interesada en cómo funciona la IA en un videojuego. No comprarán un videojuego solo por el hecho de que el diseñador programó complicados algoritmos para resolver un problema matemático a la perfección. En realidad, en la mayoría de los casos, lo que los diseñadores usan para la IA del videojuego no coincide con la rama teórica de IA. Pero siempre viene bien entender los conceptos de IA en el campo académico antes de investigar las técnicas de AI en los videojuegos.

Los conceptos de inteligencia artificial existían mucho antes de que este término fuera utilizado por primera vez. Las raíces intelectuales de la IA pueden remontarse a la mitología griega. Aparecieron artefactos inteligentes en la literatura con algunos dispositivos mecánicos reales que realizan cierto grado de inteligencia. En el siglo V a. C., Aristóteles primero inventó la lógica silogística. También dio la primera definición de inteligencia: la capacidad de poner las cosas en categorías. Pero no fue hasta la historia moderna que la investigación sobre Inteligencia Artificial se hizo más madura. En 1950, A.M. Turing publicó "Computing Machinery and Intelligence", que introdujo la prueba de Turing como una forma de operacionalizar una prueba de comportamiento inteligente. En su ejemplo ilustrativo original, un juez humano se involucra en una conversación en lenguaje natural con un humano y una máquina diseñada para generar un rendimiento indistinguible del de un ser humano. Los conceptos que trajo Turing parecían obvios, pero formaban un concepto esencial en la filosofía de la inteligencia artificial. Para los videojuegos, este concepto también se trata como el objetivo del diseño de IA en videojuegos, naturalmente.

En 1956, el término "Inteligencia Artificial" apareció por primera vez. Fue acuñado por un científico informático llamado John McCarthy. En ese momento, no había videojuegos disponibles. Sin embargo, los videojuegos dedicados a la investigación de la IA en los campos teóricos ya habían existido. En 1950, Claude Shannon publicó su trabajo sobre cómo una computadora puede jugar al ajedrez. Es la primera vez que se utiliza la IA para crear un oponente virtual. Desde entonces, el ajedrez ha sido un pilar en la investigación de la IA. Después de alrededor de medio siglo, en 1997, la computadora DEEP BLUE ganó en una partida de 6 juegos contra el Gran Maestro de ajedrez Gary Kasparov. DEEP BLUE pudo evaluar 200 millones de posiciones por segundo en comparación con 2 por segundo por un jugador humano. Este logro ha demostrado que la investigación de IA tiene sus frutos.

Sin embargo, este hito no repercutió demasiado en la industria del videojuego. Aunque las plataformas de ajedrez online están disponibles para todo el mundo, el significado de la IA del ajedrez en el campo de la investigación es mucho más importante que la forma en que se aplica para entretenecer a los jugadores. En la mayoría de videojuegos, de hecho, se supone que la IA debe dejar que el jugador gane, pero de una manera entretenida.

Por otro lado, invertir demasiado en la inteligencia artificial del videojuego para elevarla a niveles académicos no es realista. Remontándonos a la época en que surgieron los videojuegos arcade, los desarrolladores han tenido dificultades para poner incluso IA simple en los juegos

debido a las limitaciones de RAM. Incluso hoy en día, el porcentaje de ciclos de CPU que quedan para procesar la IA del juego todavía es limitado. Pero la razón por la que la IA del videojuego no tiene que ser tan buena como la AI académica es que la IA en los videojuegos no necesita ser profunda. El diseño de PacMan es evidencia de que el diseño puede no ser tan difícil como sea posible. La forma en que trabaja PacMan es agregar aleatoriedad a la decisión cuando los fantasmas llegan a un cruce, lo que garantiza que los fantasmas no sigan la misma ruta cada vez. Se trata de la variación del mismo algoritmo de búsqueda de ruta, y los programadores no necesitan hacer que el algoritmo sea más complicado para lograr ese efecto. Las IA en muchos videojuegos son exitosas sin ser profundas.

Pero esto no quiere decir que algunas veces no hagan falta herramientas complejas de la IA teórica para resolver problemas en videojuegos. De hecho, se aplican cada vez más técnicas de la IA académica para resolver problemas en videojuegos. El hecho es que la IA de videojuegos ha hecho uso completo de algunas teorías y métodos básicos de la IA real. Los métodos como las Máquinas de Estado Finito (FSM), los Árboles de Decisión y la Lógica Difusa son herramientas simples pero poderosas, y se usan ampliamente para modelar agentes de IA en juegos.

### 2.3 Videojuegos independientes: Contexto

[2] El videojuego desarrollado aquí como trabajo de fin de grado es, sin ninguna duda, un videojuego independiente. Por tanto, no viene mal saber por qué terrenos se mueve el desarrollador independiente a la hora de crear un videojuego.

Los videojuegos independientes o “indie” son videojuegos que no han tenido una base económica para su desarrollo proporcionada por una distribuidora de videojuegos. Estos videojuegos generalmente son diseñados por un equipo de no más de 20 personas, por lo que su desarrollo puede llegar a durar varios años (o horas, depende de la complejidad evidentemente).

Estos videojuegos se pueden obtener principalmente en dos plataformas: PC y dispositivos móviles. Esto es porque para un desarrollador independiente es mucho más fácil obtener las herramientas necesarias para desarrollar en PC y móviles que en videoconsolas, además de que estratégicamente tiene más lógica: La mayoría de jugadores están en estas dos plataformas.

Un desarrollador independiente tiene varias plataformas para exponer su producto al público. La más importante es, sin duda, Steam. Steam tiene aproximadamente 8900 videojuegos bajo la categoría “Indie”, con una valoración media entre el 70% y 77% (fuente: SpySteam). La cantidad de copias vendidas bajo esta categoría se encuentra aproximadamente en 885.800.000 unidades. Esta cantidad de unidades nos da una idea de la importancia de los desarrolladores independientes en el mundo de los videojuegos.

Hay algunos ejemplos de videojuegos indie con un éxito mayor que muchos videojuegos AAA. Por ejemplo, Hotline Miami (2012) ha vendido más de 2 millones de copias, a un precio de 10\$ por copia. Minecraft (2015) ha vendido la increíble cantidad de 100 millones de copias vendidas.

Estos ejemplos son la prueba de que un pequeño equipo, con la idea y ejecución adecuada pueden llegar a crear obras de éxito sobresaliente. Evidentemente son la excepción, no la regla.

Otra plataforma por todos conocida es Google Play Store, la cual permite publicar cualquier videojuego, independientemente del creador.

Uno de los aspectos a considerar son las herramientas específicas para el desarrollo de videojuegos. El creciente del sector de videojuegos tanto desde el punto de vista tecnológico como económico, ha reforzado otros aspectos del área. En particular la aparición y fortalecimiento de herramientas específicas destinadas a su desarrollo. Entre ellas destaca, sobremanera, el denominado “engine”.

Los motores de videojuegos, o “engines” suelen proporcionar un conjunto de herramientas de desarrollo visual y componentes de software que puedan ser reutilizables. Estas herramientas generalmente se proporcionan en un entorno de desarrollo integrado que permiten crear videojuegos de forma rápida y simple a través de una base de datos. En otros casos, los motores se distribuyen con una interfaz de programación de aplicaciones incorporada (o por sus siglas en inglés: API); y otros motores, sin embargo, se distribuyen como un conjunto de herramientas que agilizan y simplifican aún más el desarrollo de un videojuego, como por ejemplo: los entornos de desarrollo integrado, scripts preprogramados, y los middleware (capaces de conectar varios softwares en un solo software). Esto resulta útil a la hora de conseguir una plataforma de software flexible y reutilizable que evite la compra de otros recursos ajenos, lo cual ayuda a tener todo lo necesario para hacerlo funcional de manera inmediata, reducir los costos, complejidades y tiempos de comercialización. Todos estos factores son críticos en una industria altamente competitiva. A partir de 2001, Gamebryo, JMonkeyEngine y RenderWare eran programas de middleware ampliamente utilizados. Actualmente, se pueden citar motores como Torque Game Engine, Unity, Blender, CryEngine y Unreal Engine.

Un motor de videojuego se puede dividir en dos grandes categorías: motor gráfico y motor físico. Los motores gráficos tratan el aspecto visual del videojuego, que generan imágenes sintéticas integrando o cambiando información visual y espacial. Como ejemplo, se puede citar: OGRE 3D, Crystal Space y OpenSceneGraph. Los motores físicos se ocupan de integrar las leyes de la física, siendo responsables de simular acciones reales, a través de variables como la gravedad, la masa, la fricción, la fuerza y la flexibilidad. Como ejemplo, se puede citar: Havok, Bullet y ODE.

A pesar de la especificidad del nombre, los motores de videojuego también se utilizan para crear otros tipos de aplicaciones interactivas denominadas como "juegos serios", tales como visualizaciones arquitectónicas, educación, avances científicos, simulaciones de entrenamiento, herramientas de modelado y simulaciones físicas para recrear animaciones.

Unity es uno de los principales exponentes dentro de las aplicaciones gratuitas para el desarrollo de videojuegos. Este engine permite a los desarrolladores la capacidad de crear contenidos a través de una plataforma simplificada en la que se puede crear videojuegos sin un conocimiento demasiado profundo. Unity se puede usar de forma gratuita siempre y cuando la empresa desarrolladora no genere más de 100,000\$ en ingresos, en caso contrario, habría que pagar una licencia.

Unity es uno de los engines más sencillos y populares dentro del mercado ya que su integración es muy sencilla y varios de sus métodos y librerías son sencillas y accesible. La información para usarse está disponible a lo largo de muchos documentos por parte de los desarrolladores del engine así como información generada por terceros para el mejor uso del engine. Unity permite a los desarrolladores generar videojuegos y contenido a través de su engine mientras éste no se comercialice, todo contenido que sea comercializado deberá ser realizado en sus versiones de paga o generar un porcentaje de ganancias para Unity.

# 3 Análisis de requisitos

[3] Los requisitos software de un proyecto son una explicación en detalle del problema a resolver. Un requisito es una descripción que identifica una función, restricción u otra propiedad necesaria que debe realizarse o satisfacerse para cumplir las necesidades propuestas por los usuarios de un sistema informático.

El proceso de ingeniería de requisitos, es un conjunto de actividades ordenado que sirven para obtener los requisitos de nuestro proyecto, los cuales servirán de guía a la hora de construir la solución.

Una definición no demasiado correcta o ambigua de los requisitos software que buscamos en nuestro proyecto puede inducir a problemas, por lo que es conveniente tener claros qué requisitos vamos a cumplir antes de empezar a desarrollar el proyecto.

Una concepción clara de los requisitos nos ayudará a desarrollar el proyecto de forma más fluida y con menos problemas, ya que dispondremos de una guía a seguir.

En este proyecto se van a obtener requisitos siguiendo las directrices de Ian Somerville, el cual divide el proceso de obtención de requisitos en cuatro fases:

1. Estudio de viabilidad
2. Obtención y análisis de requisitos
3. Especificación de requisitos
4. Validación de requisitos

En el estudio de visibilidad, se estudia si las necesidades del cliente (en este caso, el propio desarrollador) se pueden satisfacer usando las tecnologías existentes en el mercado, y en el tiempo disponible. Si se determina que el proyecto es viable, se continuará con el proceso de desarrollo.

En la obtención y análisis de requisitos, se obtienen cada uno de los requisitos. Estos requisitos se pueden obtener de distintas formas, por ejemplo, sabiendo qué buscamos en el videojuego, u observando proyectos similares.

En la especificación de requisitos, toda la información recopilada en los pasos anteriores, será descrita en detalle, donde cada uno de los requisitos serán clasificados dependiendo de unos criterios determinados.

En la validación de requisitos, se verifica que toda la información obtenida de la fase anterior es real, completa y consistente. En este punto es posible que se encuentren errores, por los que debe corregirse.

Estas cuatro fases no tienen por qué seguirse en este orden estricto, y en el caso de este proyecto podemos ser un poco flexibles.

### 3.1 Estudio de viabilidad

En este apartado vamos a ver si nuestra idea de proyecto podría ser viable. El proyecto consiste en un videojuego de carreras de coches de carácter arcade. El videojuego debe ser jugable, y se debe centrar en la inteligencia artificial de los coches.

¿Tenemos las tecnologías necesarias para realizar el proyecto?

En cuanto al engine del videojuego, disponemos de Unity. Unity nos permite desarrollar videojuegos relativamente fácil y gratuitamente en este caso. Para el apartado visual disponemos de Photoshop para imágenes y 3ds Max para el modelado. Además, tenemos algunas páginas con recursos de libre acceso. Por tanto, podríamos decir que en principio tenemos todas las tecnologías necesarias.

¿Se puede realizar el proyecto en el tiempo disponible?

En mi caso personal, solo puedo desarrollar el proyecto desde enero hasta mediados de junio, teniendo las mañanas de enero, marzo, abril ocupadas. Por tanto, suponiendo que de enero a abril puedo invertir 4h diarias, y 8h diarias en mayo y junio, dispondríamos de un total de 720h aproximadamente, lo que serían 90 días trabajando 8h al día.

Dado que el objetivo de este proyecto no es un videojuego de nivel profesional, en principio el proyecto se podría realizar en este periodo de tiempo.

Ya que vemos que disponemos de las tecnologías y el tiempo necesario para llevar a cabo el proyecto, podemos concluir que su desarrollo es viable.

### 3.2 Obtención y análisis de requisitos

En esta fase, trataremos de plasmar toda la información necesaria para las siguientes fases.

Las principales actividades del proceso de obtención y análisis de requisitos, son:

- Descubrimiento de requisitos
- Clasificación y organización de requisitos
- Ordenación por prioridades y negociación de requisitos
- Documentación de requisitos

El descubrimiento de requisitos es la fase donde se obtiene la información en bruto, y a partir de aquí obtenemos los requisitos.

La clasificación y organización de requisitos consiste en ordenar en grupos lógicos los requisitos obtenidos en la fase anterior.

La ordenación y negociación de requisitos trata de ordenar por prioridad los requisitos, así como resolver incidencias o contradicciones entre ellos.

La documentación de requisitos consiste en escribir formalmente los requisitos obtenidos.

Estas fases son iterativas y se retroalimentan entre ellas continuamente. Las dos últimas fases no las he desarrollado en este caso pues no las he visto necesarias.

Normalmente para obtener esta información se usan entrevistas con el cliente, cuestionarios, se estudia documentación, etc. En este caso en particular, como es un proyecto propio, la información la proporcionaré yo mismo.

### 3.2.1 Descubrimiento de requisitos

En este apartado se describirá cómo deberá ser el proyecto, y a partir de aquí se obtendrán los requisitos necesarios.

Menús:

Este proyecto debe ser un videojuego de carreras de coches. En el menú de inicio, debe haber al menos dos modos de juego: Carrera Rápida, y Campeonato. También debe de haber una gestión de perfiles, en la cual cada perfil tiene un nickname, un total de puntos ganados y un total de monedas ganadas.

El modo de Carrera Rápida debe consistir en una carrera única. Cuando seleccionamos este modo, primero debemos seleccionar qué circuito correr, cuantas vueltas y con qué nivel de dificultad de IA jugar. Una vez elegido todo esto, pasaremos a elegir marca y modelo de coche. Debe de haber varias marcas de coches, cada una con distintos modelos, donde todos los modelos tengan ciertas características distintas, haciendo algunos coches mejores que otros para determinados circuitos. En este modo todos los modelos y marcas están desbloqueados. Una vez elegimos coche, comienza la carrera. Una vez acabada la carrera, obtenemos una serie de puntos y monedas según la posición final y según el nivel de dificultad, número de vueltas y modelo elegido, y volvemos a la pantalla principal.

El modo Campeonato consiste en una serie de carreras, una detrás de otra. En este modo se elige dificultad antes de empezar el campeonato, dependiendo de esta selección tendremos mejores o peores premios. Una vez elegido procedemos a correr en todos los circuitos, pudiendo elegir el coche que queremos correr para cada circuito. En este modo no todos los coches están desbloqueados, por lo que desbloquear un coche nos costará monedas. Una vez se desbloquea el coche, se guarda en el perfil que estamos usando. Una vez acabamos todas las carreras, se dictamina la posición final del campeonato, se nos entrega el premio, y volvemos al menú principal.

Gestión de perfiles:

Se debe de poder crear, seleccionar y eliminar perfiles de usuario. Cada perfil de usuario tendrá unos puntos totales, unas monedas que se pueden gastar, y un porcentaje de coches desbloqueados.

Jugabilidad:

El coche debe de poder alcanzar altas velocidades, además de disponer de un dispositivo de boost o nitro, el cual es limitado y se recarga con el paso de los segundos. Este nitro debe darle potencia al coche mientras se usa. El ángulo de giro de la dirección del coche debe de ser inversamente proporcional a la velocidad del coche. Esto hace que se necesite frenar antes de tomar algunas curvas a altas velocidades. Cuando un coche se choque y no pueda seguir, debe de poder respawnear cerca de su ultima posición para poder continuar la carrera. Debe de haber diferencias reales entre distintos modelos de coches, de forma que se note a primera vista la diferencia entre uno de los mejores coches y uno de los peores.

Inteligencia Artificial:

Los coches conducidos por IA deben de poder suponer un reto, no debe de ser fácil ganarles. Estos coches deben ser capaces de acelerar, frenar y girar de forma autónoma para

completar el circuito en el menor tiempo posible. Además, deben ser capaces de adelantar a otros coches.

Físicas:

Al ser un videojuego de enfoque arcade, las físicas no necesitan ser perfectamente realistas. Se busca que los choques entre coches o con los bordes del circuito no obstaculicen demasiado la continuidad de la carrera (por ejemplo, que no empiecen a dar vueltas de campana). Los coches deben derrapar un poco cuando giran a altas velocidades, pero no demasiado. No es necesario simular daños por golpes en los coches.

### 3.2.2 Clasificación y organización de requisitos

A partir del apartado anterior, vamos a obtener todos los requisitos posibles, clasificándolos en requisitos funcionales y no funcionales.

Los requisitos funcionales detallan como el sistema debe actuar, y qué es lo que debe hacer. En la especificación de estos requisitos se suelen especificar las entradas y salidas que necesitan.

Los requisitos no funcionales se refieren a los requisitos que especifican las propiedades emergentes del sistema como disponibilidad, tiempo de respuesta, rendimiento, etc.

En este apartado se obtendrán los requisitos, y en el siguiente apartado (Especificación de requisitos) detallaremos cada requisito con sus entradas, salidas, etc.

Requisitos funcionales

Interfaz de menú:

- Crear, borrar o cambiar de perfil de usuario
- Seleccionar modo de juego (Carrera Rápida o Campeonato)
- Opción de salir del juego

Interfaz en Carrera Rápida:

- Seleccionar circuito
- Seleccionar número de vueltas
- Seleccionar dificultad
- Seleccionar marca y modelo de coche

Interfaz en Campeonato:

- Seleccionar dificultad
- Seleccionar marca y modelo de coche
- Desbloquear un coche

Interfaz dentro de la carrera:

- Disponibilidad de menú de pausa
- Calcular puntos
- Ir al menú

Jugabilidad:

- El coche debe poder acelerar, frenar y girar de forma que pueda completar el circuito
- El coche debe poder alcanzar altas velocidades
- El coche debe tener un boost o nitro
- El nitro debe de gastarse mientras se usa, y recargarse mientras no se usa
- El ángulo de giro de la dirección del coche debe ser inversamente proporcional a su velocidad
- Cuando un coche no pueda continuar, debe respawnear cerca de su última posición
- Debe haber diferencias reales entre distintos modelos de coches

Inteligencia Artificial:

- Los coches conducidos por IA deben ser capaces de acelerar, frenar y girar de forma autónoma para completar el circuito en el menor tiempo posible.
- Los coches conducidos por IA deben poder completar el circuito en un tiempo suficientemente bajo para que suponga un reto para el jugador
- Los coches conducidos por IA deben ser capaces de adelantar a otros coches

Físicas:

- Los choques entre coches o con los bordes del circuito no deben entorpecer bruscamente la carrera
- Los coches deben derrapar al girar bruscamente a altas velocidades, pero no demasiado
- No se deben simular daños por golpes

#### Requisitos no funcionales

- El videojuego debe de consumir pocos recursos, para que un PC de gama media pueda ejecutarlo de forma fluida
- El videojuego debe poder ejecutarse en cualquier momento
- El videojuego debe ser fiable y no tener bugs o crashes espontáneos
- El videojuego debe poder estar abierto a ampliación de contenido o características en un futuro
- El videojuego debe de usar contenido multimedia con derechos que permitan su uso
- La ejecución del videojuego debe no suponer un problema de seguridad para el sistema
- El videojuego debe adaptar su resolución a la del monitor, para que se visualice correctamente

A priori estos serían los requisitos más importantes. En proyectos de videojuegos es muy difícil escribir todos y cada uno de los posibles requisitos, muchos son directamente programados. Sin embargo, se ha intentado que los más importantes estén plasmados antes de la fase de diseño.

### 3.3 Especificación de requisitos

Ya tenemos los requisitos, ahora vamos a especificarlos y aumentar el nivel de detalle.

#### Requisitos funcionales

<b>Nombre</b>	RF-IM1
<b>Resumen</b>	Crear, borrar o cambiar de perfil de usuario
<b>Descripción</b>	Se debe poder crear o borrar un perfil de usuario, así como cambiar de perfil
<b>Entradas</b>	Click en botón de gestión de usuarios
<b>Salidas</b>	Información de usuario actualizada
<b>Procesos</b>	Al agregar o borrar un usuario, se agrega o borra de la base de datos. Al cambiar de usuario, simplemente se cambia de datos de usuario seleccionado
<b>Precondiciones</b>	Debe estar en la pantalla de menú
<b>Postcondiciones</b>	

<b>Nombre</b>	RF-IM2
<b>Resumen</b>	Seleccionar modo de juego
<b>Descripción</b>	Se debe poder seleccionar el modo de juego deseado: Carrera Rápida o Campeonato
<b>Entradas</b>	Click en el botón
<b>Salidas</b>	Carga de la pantalla correspondiente
<b>Procesos</b>	Cuando se hace click en el botón correspondiente, el sistema lo detecta y procede a cargar la pantalla correspondiente
<b>Precondiciones</b>	Debe estar en la pantalla de menú
<b>Postcondiciones</b>	

<b>Nombre</b>	RF-IM3
<b>Resumen</b>	Opción de salir del juego
<b>Descripción</b>	Se debe poder salir del juego pulsando un botón

<b>Entradas</b>	Click en el botón
<b>Salidas</b>	Cerrar el juego
<b>Procesos</b>	Cuando se hace click en el botón correspondiente, el sistema lo detecta y cierra el juego
<b>Precondiciones</b>	Debe estar en la pantalla de menú o de pausa en la carrera
<b>Postcondiciones</b>	

<b>Nombre</b>	RF-ICR1
<b>Resumen</b>	Seleccionar circuito
<b>Descripción</b>	Se debe poder seleccionar el circuito en el cual correr la Carrera Rápida
<b>Entradas</b>	Click en el circuito deseado
<b>Salidas</b>	El circuito clickado se selecciona
<b>Procesos</b>	Cuando se hace click en el circuito, la información del circuito seleccionado se guarda para saber qué circuito cargar
<b>Precondiciones</b>	Debe estar en modo Carrera Rápida, antes de correr
<b>Postcondiciones</b>	Se debe actualizar el circuito seleccionado

<b>Nombre</b>	RF-ICR2
<b>Resumen</b>	Seleccionar número de vueltas
<b>Descripción</b>	Se debe poder seleccionar el número de vueltas a correr en la Carrera Rápida
<b>Entradas</b>	Click en el botón del número de vueltas deseado
<b>Salidas</b>	El número de vueltas se actualiza
<b>Procesos</b>	Cuando se hace click en el botón con el número de vueltas deseado, la información se guarda para saber el número de vueltas en futuras pantallas
<b>Precondiciones</b>	Debe estar en modo Carrera Rápida, antes de correr

<b>Postcondiciones</b>	Se debe actualizar el número de vueltas seleccionado
------------------------	--

<b>Nombre</b>	RF-ICR-ICAMP1
<b>Resumen</b>	Seleccionar dificultad
<b>Descripción</b>	Se debe poder seleccionar la dificultad de la IA
<b>Entradas</b>	Click en el nivel de dificultad deseado
<b>Salidas</b>	El nivel clickado se selecciona
<b>Procesos</b>	Cuando se hace click en el nivel de dificultad deseado, la información del nivel seleccionado se guarda para saber qué circuito cargar
<b>Precondiciones</b>	Se debe seleccionar antes de correr la carrera rápida o antes de comenzar el torneo
<b>Postcondiciones</b>	En modo torneo, se debe actualizar los premios finales

<b>Nombre</b>	RF-ICR-ICAMP2
<b>Resumen</b>	Seleccionar modelo y marca de coche
<b>Descripción</b>	Se debe poder seleccionar modelo de coche deseado para correr la carrera rápida o el campeonato
<b>Entradas</b>	Click en el modelo deseado
<b>Salidas</b>	El modelo clickado se selecciona
<b>Procesos</b>	Cuando se hace click en el modelo deseado, se marca como seleccionado y se guarda la información
<b>Precondiciones</b>	Si es modo Campeonato, el coche debe estar desbloqueado
<b>Postcondiciones</b>	Se debe actualizar el modelo seleccionado

<b>Nombre</b>	RF-ICAMP1
<b>Resumen</b>	Desbloquear modelo de coche

<b>Descripción</b>	Se debe poder desbloquear un modelo de coche bloqueado
<b>Entradas</b>	Click en el modelo deseado
<b>Salidas</b>	El modelo clickado se desbloquea
<b>Procesos</b>	Cuando se hace click en el modelo deseado, se desbloquea para el perfil actual, y se podrá usar siempre
<b>Precondiciones</b>	Debe estar en modo Campeonato, el modelo debe estar previamente bloqueado, se deben tener suficientes monedas
<b>Postcondiciones</b>	El coche se desbloqueará permanentemente para ese perfil, se restará el precio del coche a las monedas del perfil

<b>Nombre</b>	RF-ICARRERA1
<b>Resumen</b>	Disponibilidad de menú de pausa
<b>Descripción</b>	Se debe de tener un menú de pausa disponible, con botones para ir al menú, escritorio o continuar en el juego. Mientras está el menú de pausa, el juego puede continuar
<b>Entradas</b>	Presionar Esc
<b>Salidas</b>	Se abre una pantalla de pausa
<b>Procesos</b>	Cuando se pulsa Esc, se abre un menú de pausa con botones funcionales
<b>Precondiciones</b>	Se debe estar dentro de una carrera
<b>Postcondiciones</b>	Se debe abrir una ventana

<b>Nombre</b>	RF-ICARRERA2
<b>Resumen</b>	Calcular puntos
<b>Descripción</b>	Se calculan los puntos y monedas ganados en función del modelo, dificultad, etc deseados
<b>Entradas</b>	
<b>Salidas</b>	
<b>Procesos</b>	Al finalizar la carrera, se calculan los puntos y monedas ganados

<b>Precondiciones</b>	Debe haber finalizado la carrera
<b>Postcondiciones</b>	Las monedas ganadas se deben añadir al perfil actual

<b>Nombre</b>	RF-ICARRERA3
<b>Resumen</b>	Ir al menú
<b>Descripción</b>	Al finalizar la carrera, volvemos al menú
<b>Entradas</b>	
<b>Salidas</b>	
<b>Procesos</b>	Al finalizar la carrera, después de calcular los puntos, volvemos al menú
<b>Precondiciones</b>	Debe haber finalizado la carrera y se deben haber calculados los puntos
<b>Postcondiciones</b>	Se debe cargar la pantalla de menú

<b>Nombre</b>	RF-JCARRERA1
<b>Resumen</b>	El coche debe poder acelerar, frenar y girar de forma que pueda completar el circuito
<b>Descripción</b>	El coche debe poder acelerar, frenar y girar de forma que pueda completar el circuito
<b>Entradas</b>	Teclas de Input
<b>Salidas</b>	Reacción en el coche
<b>Procesos</b>	Se detectan las teclas de input pulsadas, y en correspondencia se efectúan los cambios necesarios en el coche
<b>Precondiciones</b>	Debe estar en carrera
<b>Postcondiciones</b>	Debe haber un cambio en el coche

<b>Nombre</b>	RF-JCARRERA2
<b>Resumen</b>	El coche debe poder alcanzar altas velocidades
<b>Descripción</b>	El coche debe poder alcanzar altas velocidades en carrera
<b>Entradas</b>	

<b>Salidas</b>
<b>Procesos</b>
<b>Precondiciones</b>
<b>Postcondiciones</b>

<b>Nombre</b>	RF-JCARRERA3
<b>Resumen</b>	El coche debe tener un boost o nitro
<b>Descripción</b>	El coche debe disponer de una opción de nitro, que le de un impulso temporal al coche
<b>Entradas</b>	Tecla Input de nitro
<b>Salidas</b>	Impulso temporal
<b>Procesos</b>	Se genera un impulso mientras la tecla input esté pulsada
<b>Precondiciones</b>	La tecla Input de nitro debe estar pulsada, y debe quedar suficiente nitro
<b>Postcondiciones</b>	Se impulsa el coche

<b>Nombre</b>	RF-JCARRERA4
<b>Resumen</b>	El nitro debe de gastarse mientras se usa, y recargarse mientras no se usa
<b>Descripción</b>	Mientras se usa el nitro, éste debe ir gastándose hasta que no se pueda seguir usando. Mientras no se usa, se recarga
<b>Entradas</b>	Tecla Input de nitro (o no pulsada)
<b>Salidas</b>	Resta al nitro restante si se está usando nitro, suma al nitro restante si no se está usando nitro
<b>Procesos</b>	Se resta el nitro que se está usando si se está usando, se suma si no se está usando
<b>Precondiciones</b>	El nitro se debe estar usando para gastarlo, y no usando para recargarlo
<b>Postcondiciones</b>	Debe haber menos nitro (si se está usando) o más (si no se está usando)

<b>Nombre</b>	RF-JCARRERA5
---------------	--------------

<b>Resumen</b>	El ángulo de giro de la dirección del coche debe ser inversamente proporcional a su velocidad
<b>Descripción</b>	El ángulo de giro de la dirección del coche debe ser inversamente proporcional a su velocidad. Esto es para evitar giros bruscos a altas velocidades
<b>Entradas</b>	Velocidad
<b>Salidas</b>	Angulo máximo de giro
<b>Procesos</b>	Se calcula el ángulo máximo con una función cuyo input es la velocidad
<b>Precondiciones</b>	Debemos estar en carrera
<b>Postcondiciones</b>	Se debe actualizar el ángulo máximo. Si el nuevo ángulo es menor que el ángulo en el que las ruedas están en ese momento, las ruedas se centran hasta cumplir con el ángulo máximo

<b>Nombre</b>	RF-JCARRERA6
<b>Resumen</b>	Cuando un coche no pueda continuar, debe respawnear cerca de su última posición
<b>Descripción</b>	Cuando un coche no pueda continuar, debe respawnear cerca de su última posición. Esto es para evitar que algún coche se quede “pillado” en un muro, por ejemplo
<b>Entradas</b>	Velocidad del coche
<b>Salidas</b>	Nueva posición del coche
<b>Procesos</b>	Se lleva el coche al último checkpoint pasado, reseteando su velocidad
<b>Precondiciones</b>	El coche lleva más de x segundos a menos de y velocidad
<b>Postcondiciones</b>	El coche tendrá reseteadas sus propiedades físicas, como velocidad o velocidad angular

<b>Nombre</b>	RF-JCARRERA7
---------------	--------------

<b>Resumen</b>	Debe haber diferencias reales entre distintos modelos de coches
<b>Descripción</b>	Los modelos de coches deben de crear de tal forma que haya diferencia notable entre coches tanto en manejo y aceleración como en tiempo de completar una vuelta
<b>Entradas</b>	
<b>Salidas</b>	
<b>Procesos</b>	
<b>Precondiciones</b>	
<b>Postcondiciones</b>	

<b>Nombre</b>	RF-IA1
<b>Resumen</b>	Los coches conducidos por IA deben ser capaces de acelerar, frenar y girar de forma autónoma para completar el circuito en el menor tiempo posible
<b>Descripción</b>	Los coches conducidos por IA deben ser capaces de acelerar, frenar y girar de forma autónoma para completar el circuito en el menor tiempo posible
<b>Entradas</b>	Posición del coche respecto al circuito, velocidad actual del coche
<b>Salidas</b>	Aceleración/freno, giro necesario para continuar completando la vuelta
<b>Procesos</b>	La IA dictamina cuánto acelerar, frenar o girar para completar la vuelta en el menor tiempo posible
<b>Precondiciones</b>	El coche debe estar controlado por IA no por el jugador
<b>Postcondiciones</b>	El coche debe completar la vuelta en el menor tiempo posible

<b>Nombre</b>	RF-IA2
<b>Resumen</b>	Los coches conducidos por IA deben poder completar el circuito en un tiempo

<b>Descripción</b>	suficientemente bajo para que suponga un reto para el jugador
<b>Entradas</b>	Los coches conducidos por IA deben poder completar el circuito en un tiempo suficientemente bajo para que suponga un reto para el jugador
<b>Salidas</b>	
<b>Procesos</b>	
<b>Precondiciones</b>	
<b>Postcondiciones</b>	

<b>Nombre</b>	RF-IA3
<b>Resumen</b>	Los coches conducidos por IA deben ser capaces de adelantar a otros coches
<b>Descripción</b>	Los coches conducidos por IA deben ser capaces de adelantar a otros coches, minimizando las colisiones en la medida de lo posible
<b>Entradas</b>	Posición del resto de coches
<b>Salidas</b>	Cambios en la aceleración/freno y giro
<b>Procesos</b>	La IA, a partir de la posición del resto de coches respecto al coche actual, dictamina los cambios en aceleración/freno y giro
<b>Precondiciones</b>	El coche debe ser conducido por IA
<b>Postcondiciones</b>	El coche debe modificar su trayectoria para adelantar satisfactoriamente

<b>Nombre</b>	RF-F1
<b>Resumen</b>	Los choques entre coches o con los bordes del circuito no deben entorpecer bruscamente la carrera
<b>Descripción</b>	Los choques entre coches o con los bordes del circuito no deben entorpecer bruscamente la carrera. Por ejemplo, no pueden dar vueltas de campana, o entorpecer a terceros coches, en la medida de lo posible.

**Entradas****Salidas****Procesos****Precondiciones****Postcondiciones****Nombre**

RF-F2

**Resumen**

Los coches deben derrapar al girar bruscamente a altas velocidades, pero no demasiado

**Descripción**

Los coches deben derrapar al girar bruscamente a altas velocidades, pero no demasiado

**Entradas****Salidas****Procesos****Precondiciones****Postcondiciones****Nombre**

RF-F3

**Resumen**

No se deben simular daños por golpes

**Descripción**

No se deben simular daños por golpes

**Entradas****Salidas****Procesos****Precondiciones****Postcondiciones**

## Requisitos no funcionales

### *Rendimiento*

- El videojuego debe de consumir pocos recursos, para que un PC de gama media pueda ejecutarlo de forma fluida
- El videojuego debe adaptar su resolución a la del monitor, para que se visualice correctamente

### *Seguridad*

- La ejecución del videojuego debe no suponer un problema de seguridad para el sistema

### *Fiabilidad*

- El videojuego debe ser fiable y no tener bugs o crashes espontáneos, y debe tener el menor número de bugs posibles

### *Disponibilidad*

- El videojuego debe poder estar disponible para ejecutarse en cualquier momento

### *Mantenibilidad*

- El videojuego debe poder estar abierto a ampliación de contenido o características en un futuro

### *Portabilidad*

- Tanto el videojuego compilado como el proyecto deben tener la posibilidad de ser ejecutados en cualquier PC Windows

### *Propiedad Intelectual*

- El videojuego debe de usar contenido multimedia con derechos que permitan su uso

## 3.4 Validación de requisitos

En esta cuarta y última fase se comprueba que el conjunto de requisitos es correcto, definen correctamente el sistema y satisfacen las necesidades que el videojuego tiene para cumplir su objetivo.

Durante esta fase de validación de requisitos, se han realizado un conjunto de comprobaciones en el documento de especificación de requisitos, los cuales son los siguientes:

- Verificación de validez: Todos los requisitos obtenidos son válidos y viables en su cumplimiento
- Verificación de consistencia: Ninguno de los requisitos obtenido se contradice con algún otro requisito
- Verificación de completitud: En la verificación de completitud se busca verificar que los requisitos contemplan todas las tareas y limitaciones en el videojuego. Este apartado es muy difícil de cumplir en su totalidad. Dada la naturaleza de los videojuegos, la lista de

tareas, limitaciones, interacciones entre elementos, etc. es interminable, y conseguir todos los requisitos posibles sería una tarea casi utópica.

- Verificación de realismo: Todos los requisitos pueden ser implementados utilizando la tecnología existente, y en el tiempo disponible.
- Verificabilidad: Todos los requisitos descritos son comprobables mediante pruebas del sistema, y podemos ver fácilmente si se cumplen o no en el proyecto final.

Hay que tener en cuenta, que, dada la naturaleza del desarrollo de videojuegos, se debe ser muy flexible con la especificación de requisitos a priori, pues los cambios en el videojuego durante su desarrollo son constantes.



# 4 Planificación y metodología

En este capítulo se va a explicar la planificación en el desarrollo del videojuego. Para ello, es conveniente dividir el proyecto en partes o módulos, donde cada módulo ha tenido su propia planificación de desarrollo. También se han asignado los tiempos que se deberían haber empleado a cada módulo.

También se explica qué metodología se ha usado. En este apartado explicaremos tanto el modelo de proceso software utilizado como las herramientas que se han empleado.

Antes de nada, al videojuego hay que ponerle un título. Para este videojuego, ha sido “Hell’s Driver”.

## 4.1 Módulos del videojuego

Puesto que el desarrollo de un videojuego es un proceso en el cual se deben tocar muchos campos muy distintos, por comodidad a la hora de desarrollar vamos a especificar estos campos y agruparlos en distintos módulos. De esta forma tendremos una visión clara y global del desarrollo del videojuego.

### Interfaces de usuario

En este módulo incluiremos el desarrollo de las distintas interfaces de usuario del videojuego (menú, selección de coche, marcador de velocidad y posición en carrera, etc.). La tarea de diseñar interfaces de usuario no es exclusivamente visual, ya que hay que integrarla con el videojuego, de forma que las interfaces sean interactivas (por ejemplo, que los botones funcionen). En este módulo por lo tanto no se incluye solo el diseño visual de las IU, sino que también su funcionalidad.

### Gestión de escenas

Llamamos escena a una fase o “escenario” del videojuego. Una escena es una fase aislada del videojuego donde ocurren una serie de cosas concretas. Por ejemplo, el menú es una escena. Cuando corremos en un circuito, estamos en otra escena. Cuando estamos eligiendo coche, estamos en otra escena. Por tanto, las escenas contienen los elementos de IU, modelos, etc. que componen el videojuego. Este módulo se centra tanto en la jerarquía de escenas como en el contenido de éstas. En el capítulo de diseño se verá de forma mucho más clara este concepto.

### Modelado

Este módulo consiste en el modelado de todos los elementos 3D que se van a usar en el videojuego, incluyendo coches y circuitos. Aunque todos los coches y circuitos se han modelado desde cero, hay algún que otro objeto (árboles y rocas) que se han obtenido de paquetes gratuitos. Estos objetos también se incluyen en este módulo.

Puesto que los modelos necesitan al menos un material adjunto para que sean visibles, la gestión de materiales y texturas también están incluidas en este módulo.

#### Mecánicas de control del vehículo

Uno de los módulos más importantes del proyecto. En este apartado se define cómo funciona el vehículo: Cómo funciona la aceleración, cómo funciona el giro, cómo funciona el boost, etc. También se definen las propiedades físicas del vehículo, como la masa, el agarre, la potencia, la potencia de frenado, la posición del centro de masa, etc. En este módulo incluimos también los sensores que tiene cada vehículo para obtener información del exterior.

Este módulo no incluye la inteligencia artificial del vehículo. A lo que respecta con esta parte del proyecto, de dónde le vengan las órdenes como acelerar o girar (IA o jugador) le es indiferente.

#### Inteligencia Artificial

El módulo más importante y extenso del proyecto. En este módulo se define tanto qué sistema de inteligencia artificial es elegido para los vehículos, como el proceso de entrenamiento de este sistema.

#### Sonido

Incluimos tanto los sonidos de los coches como la música de fondo.

#### Programación del videojuego

Conforme se van desarrollando el resto de módulos, hay que cohesionarlos para que el videojuego como tal vaya avanzando. Esto incluye, desde programar en qué posición está cada coche en la carrera, hasta la gestión de perfiles de usuario. Por tanto, es un módulo fundamental y que requiere mucho tiempo y trabajo detrás.

## 4.2 Planificación temporal

Ya tenemos claro los distintos módulos o tareas a desarrollar, ahora se explicará la planificación pensada para desarrollar el proyecto antes de empezarlo.

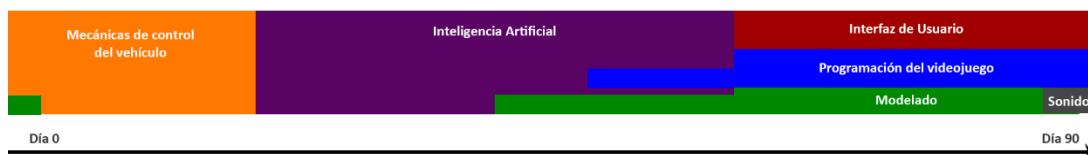
En el caso de este proyecto, el desarrollo se ha llevado a cabo desde el 1 de enero hasta el 18 de junio, teniendo en enero, febrero y marzo disponibles sólo las tardes. Por tanto, vamos a asumir que tenemos disponibles para el desarrollo 4h cada día desde enero hasta marzo, y 8h cada día mayo y junio. Esto hace un total aproximado de 420h, lo cual serían 90 días de trabajo, trabajando 8h cada día. La distribución de tiempos se ha planificado teniendo en cuenta esta escala de 90 días, para hacerlo más intuitivo.

La planificación sería la siguiente.

- ➔ Días 1-20: Se modelan un circuito y un coche de prueba, y se desarrolla el módulo de Mecánicas de control del vehículo. También se modelan algunos objetos de prueba para coger práctica a la tarea de modelar.

- Días 21-60: Se desarrolla el módulo de Inteligencia Artificial. En estos días realiza tanto la tarea de elegir el sistema de IA a implementar, como la implementación y entrenamiento. Paralelamente se modelan algunos coches y los cuatro circuitos, ya que son necesarios para entrenar la IA.
- Días 60-90: Se lleva a cabo casi la totalidad del modelado, del diseño de interfaces de usuario, de gestión de escenas, sonido y de programación de videojuego. Aunque bien algunos módulos como el modelado se tocaron bastante antes, la mayor carga de trabajo en modelado, IU y programación de videojuego se ha realizado en este tramo de tiempo.

Gráficamente, el desarrollo de los distintos módulos llevado a cabo podría representarse así:



La planificación que se pensó antes de comenzar el proyecto se ha llevado a cabo sin apenas cambios. Con el proyecto acabado, puedo decir que he cumplido con la planificación antes descrita. Cabe decir que los distintos módulos han estado más entrelazados, y por algunas circunstancias relacionadas con la IA, ésta se ha alargado algo más de lo esperado, pero en general se ha cumplido con lo planificado.

### 4.3 Metodología

Cada módulo desarrollado necesita de sus propias herramientas. En este apartado voy a explicar las principales herramientas usadas.

#### 4.3.1 Unity

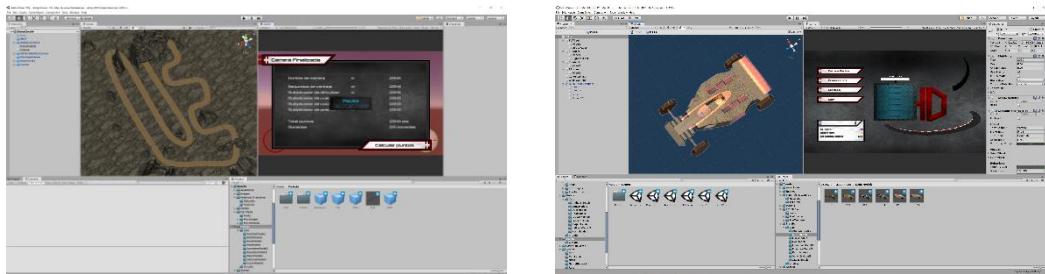
Unity es una “engine” o motor de videojuegos multiplataforma, creado por Unity Technologies. Unity se puede usar tanto en Windows como OS X y Linux, y tiene soporte de compilación para infinidad de plataformas: Desde Windows, OS X y Linux hasta Android, iOS o consolas como Xbox.

Unity está enfocado a desarrolladores independientes que no pueden crear su propio motor de juego, ofreciendo herramientas para desarrollar un videojuego gratis y potentes.

El motor gráfico utiliza OpenGL y Direct3D (Windows). El scripting viene a través de Mono, la implementación en código abierto de .NET Framework. Por tanto, los programadores pueden usar UnityScript, C# o Boo. En el caso de este proyecto se ha usado C#.

La versión de Unity usada es Unity 2019.3.

Unity tiene una gran cantidad de tipos de licencias, pero en nuestro caso sólo usaremos la licencia Unity Personal, la cual nos ofrece todas las prestaciones del motor siempre y cuando no superemos un tope de ingresos de 100.000 dólares.



*Imágenes de Unity, engine que utilizaremos en este proyecto*

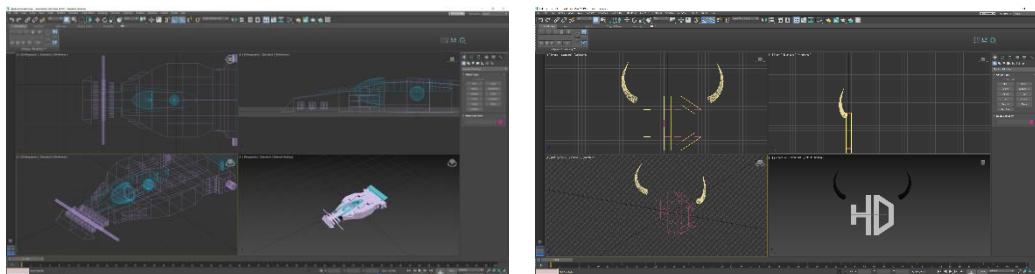
#### 4.3.2 Autodesk 3DS Max

Autodesk 3DS Max es un programa de creación de gráficos y animación en tres dimensiones. 3DS Max es uno de los programas de modelado y animación más utilizado, especialmente para la creación de videojuegos, pero también para anuncios de televisión, películas, arquitectura, etc.

En este proyecto, el uso de este software ha sido exclusivamente para modelar objetos como coches y circuitos.

Autodesk 3DS Max no es software libre, sin embargo, he podido usarlo porque dispongo de licencia de estudiante.

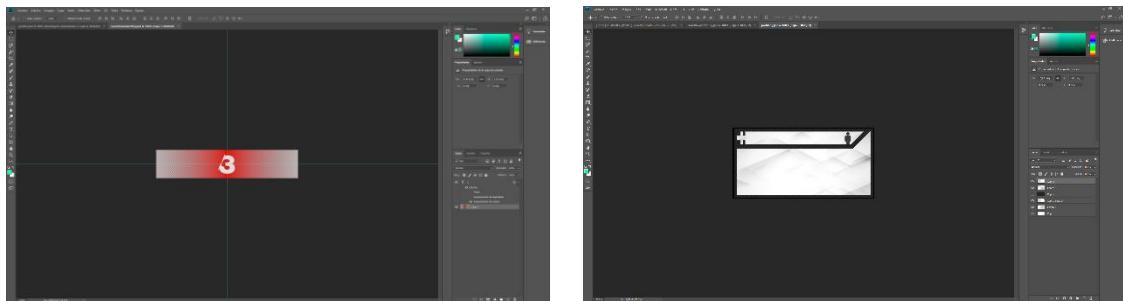
La versión utilizada ha sido Autodesk 3ds Max 2019.



*Imágenes de 3DS Max, software de modelado con el que diseñaremos objetos como coches y circuitos*

#### 4.3.3 Adobe Photoshop CC

Adobe Photoshop es un editor de gráficos desarrollado por Adobe. Aunque su principal uso es el de retoque de fotografías, su versatilidad hace que se puedan crear todo tipo de imágenes desde cero.



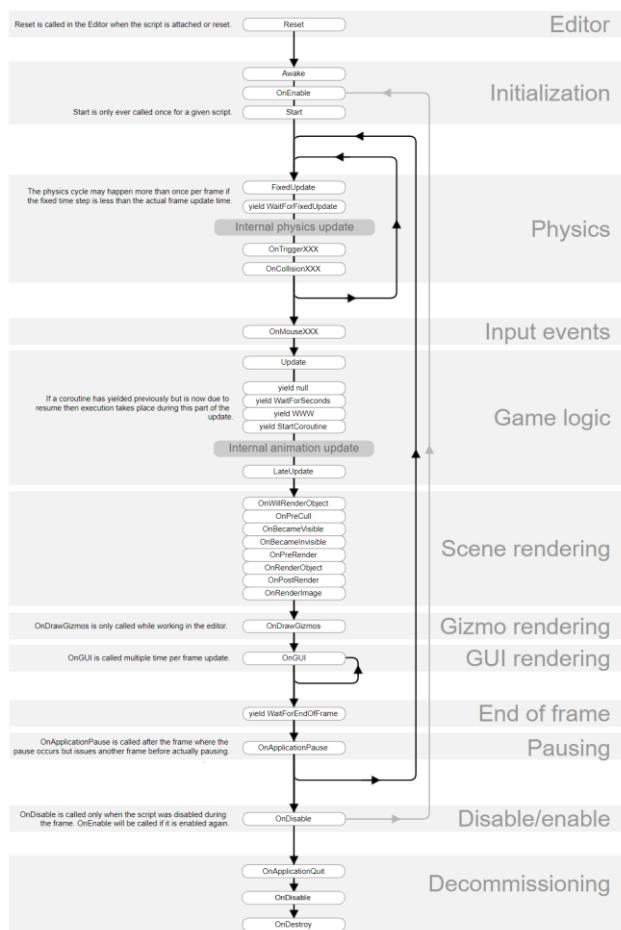
Imágenes de Photoshop CC, software de edición fotográfica que usaremos para diseñar las interfaces

En este proyecto Adobe Photoshop ha sido usado principalmente para el desarrollo de Interfaces de Usuario.

#### 4.3.4 Programación

El lenguaje de programación utilizado ha sido C#. En Unity, cada uno de los objetos que tenemos en la escena pueden tener adjuntos los llamados Script, el cual sirve para que programemos la funcionalidad de ese objeto. Cada Script de cada objeto tiene funciones que se ejecutan cada frame, pero esto no es obligatorio.

En cada frame, por tanto, cada Script llama a estas funciones, en este orden:



Este orden es muy importante de tenerlo claro a la hora de programar, para evitar bugs y errores que pueden presentarse de forma muy fácil si desconocemos esta información.

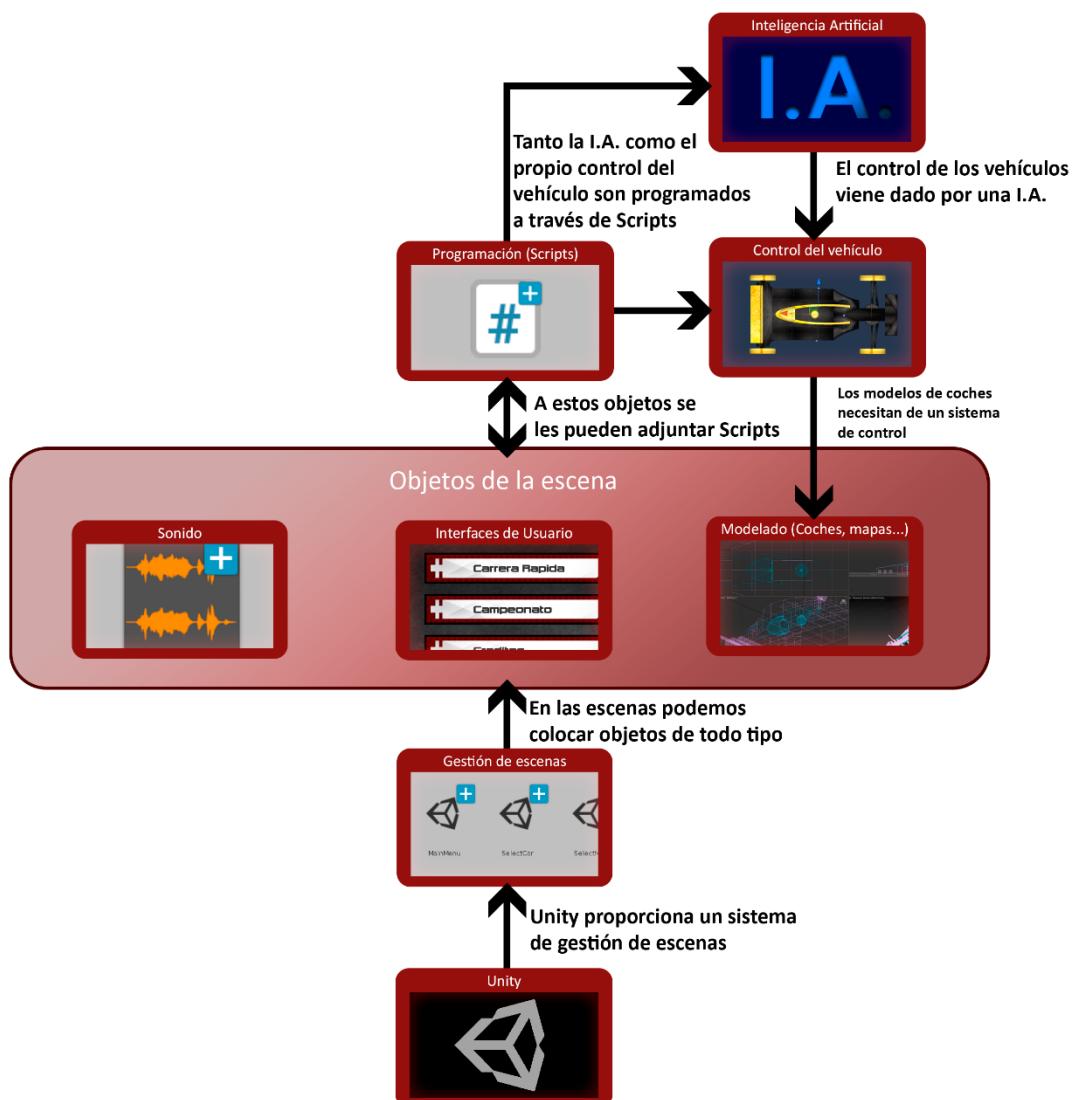
El IDE utilizado ha sido Visual Studio 2019 Community Edition. Visual Studio está completamente integrado con Unity, por lo que el uso de este IDE posee muchas ventajas respecto al resto de IDEs.

# 5 Diseño y desarrollo

Antes de nada, hagamos un resumen de lo que hemos visto en la memoria hasta ahora. Hemos comenzado con una breve introducción, explicando muy por encima en qué consiste este proyecto, así como mi motivación personal y objetivos. Se ha hablado de la historia de los videojuegos, y de la historia de la inteligencia artificial en éstos. Se ha comenzado a hablar del proyecto en sí haciendo un análisis de requisitos, y se ha hecho una planificación del desarrollo del proyecto. Y en el capítulo anterior, se ha enseñado cómo jugar al videojuego, para que veamos el resultado.

En este capítulo, vamos a empezar a hablar de cómo se ha desarrollado el proyecto. En “Diseño y desarrollo”, vamos a englobar todos los módulos antes planificados y explicar con gran nivel de detalle cómo se han llevado a cabo, exceptuando Inteligencia Artificial, la cual dada su importancia merece un capítulo adicional en esta memoria.

La siguiente imagen es un esquema de lo que sería la cohesión entre módulos a alto nivel.



Como vemos en la imagen, todo nace a partir del engine Unity. Con Unity podemos crear varias escenas. En cada escena podemos implementar objetos de todo tipo, desde objetos en tres dimensiones como son los circuitos, vehículos, árboles, etc. como sonidos, luces, skybox, o interfaces de usuario. A cada uno de estos objetos podemos adjuntarles Scripts. En los Scripts definimos las acciones que realizan estos objetos, y cómo interaccionan con el resto. Con Scripts definimos tanto la funcionalidad de estos objetos, como las mecánicas de control del vehículo, así como la Inteligencia Artificial.

Los módulos se van a explicar en un orden coherente para que se entienda todo lo mejor posible. Por tanto, se comenzará explicando el módulo de Gestión de escenas. Seguidamente se explicarán las mecánicas de control del vehículo. Después se explicarán la, el modelado, la interfaz de usuario y sonido, en este orden. Con todos estos módulos explicados, dejamos para el final el módulo de programación del videojuego.

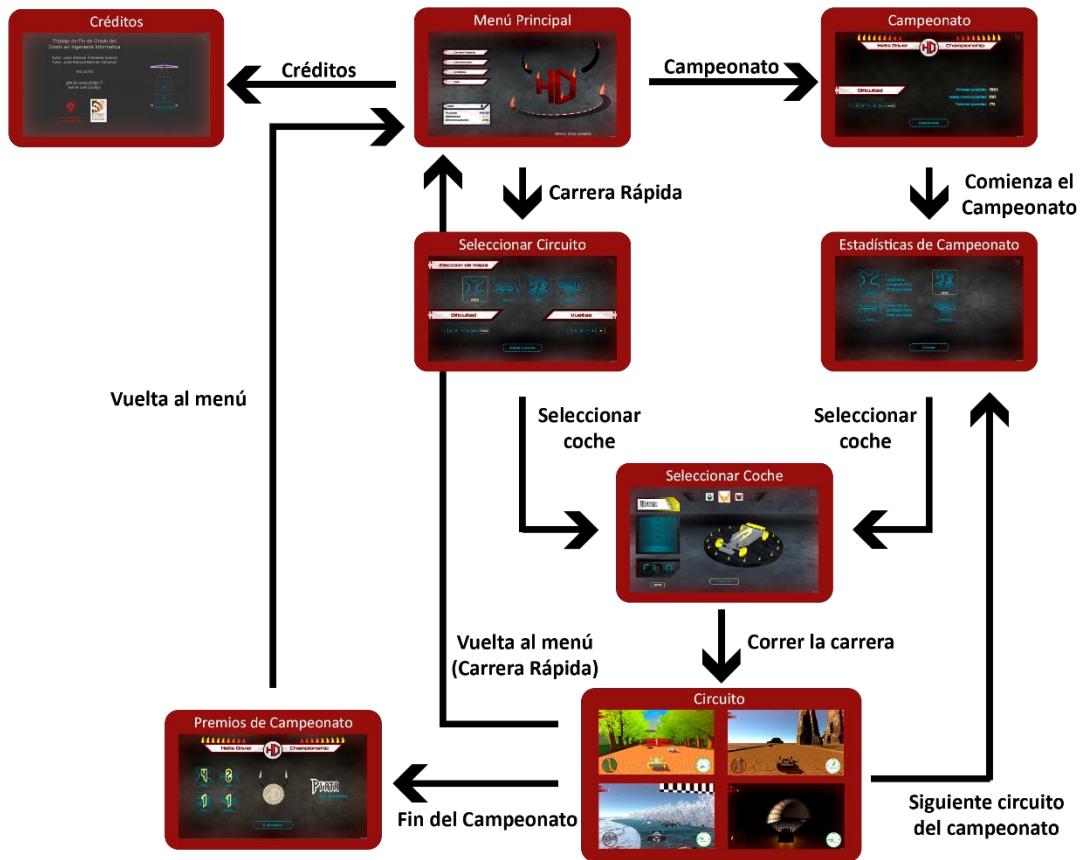
## 5.1 Gestión de escenas

Como ya se explicó anteriormente, una escena es cada uno de los niveles o pantallas del videojuego. Cuando creamos una escena, tenemos un lienzo en blanco. A este lienzo podemos colocarle todo tipo de modelos en tres dimensiones, así como interfaces de usuario, como otros muchos objetos. Además, podemos adjuntarle Scripts a cada uno de los objetos, para así desarrollar la tarea para la cual la escena ha sido creada. Las escenas que se han creado son las siguientes:

- Menú Principal: Primera pantalla que vemos al iniciar el juego. En esta pantalla tenemos los botones de Carrera Rápida y Campeonato, así como Créditos y Salir. También tenemos la gestión de perfiles.
- Seleccionar Circuito: Primera escena que se abre al pulsar el botón de Carrera Rápida del Menú Principal. En esta escena se elige el circuito, la dificultad y el número de vueltas.
- Seleccionar Coche: Escena que se abre una vez hemos elegido circuito en caso de Carrera Rápida, o después de la escena de Estadísticas de Campeonato. En esta escena elegimos modelo de coche para competir.
- Campeonato: Primera pantalla que se abre al pulsar el botón Campeonato del Menú Principal. Nos permite elegir la dificultad del campeonato.
- Estadísticas de Campeonato: Escena que vemos antes de elegir coche para correr. En esta escena vemos los datos de las carreras que hemos completado en el campeonato. Se abre después de la escena Campeonato, y después de cada carrera.
- Premios de Campeonato: Escena que vemos al acabar todas las carreras. Nos dice el premio obtenido. Después de esta escena, volvemos al menú principal.
- EightCircuit: Escena que contiene el circuito de nombre “Eight”. Se abre cuando hemos elegido coche para correr, y el circuito que tenemos que correr es este. En esta escena se desarrolla la carrera.
- DizzyCircuit: Escena que contiene el circuito de nombre “Dizzy”. Se abre cuando hemos elegido coche para correr, y el circuito que tenemos que correr es este. En esta escena se desarrolla la carrera.
- WhirlCircuit: Escena que contiene el circuito de nombre “Whirl”. Se abre cuando hemos elegido coche para correr, y el circuito que tenemos que correr es este. En esta escena se desarrolla la carrera.

- SubwayCircuit: Escena que contiene el circuito de nombre “Subway”. Se abre cuando hemos elegido coche para correr, y el circuito que tenemos que correr es este. En esta escena se desarrolla la carrera.

La estructura de las escenas es más fácil de entender si la vemos visualmente:



## 5.2 Mecánicas de control del vehículo

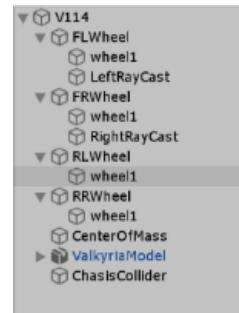
Uno de los módulos más importantes del proyecto. En este apartado se va a explicar a detalle cómo funciona el vehículo: Cómo funciona la aceleración, cómo funciona el giro, cómo funciona el boost, etc. También van a explicar las propiedades físicas del vehículo, como la masa, el agarre, la potencia, la potencia de frenado, la posición del centro de masa, etc.

En este apartado no se explicará la inteligencia artificial del vehículo.

### 5.2.1 Partes del vehículo

Cada vehículo se compone de los objetos que podemos ver en la imagen. Dos ruedas delanteras, dos ruedas traseras, un centro de masa, el modelado que le da forma, y un Box Collider. Vamos a explicar qué contiene cada una de las partes, y en apartados siguientes se explicarán a fondo.

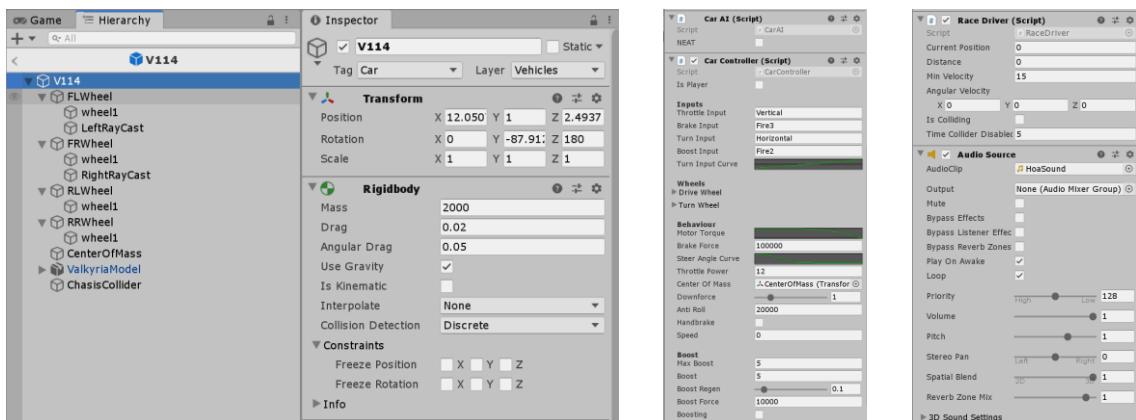
#### Objeto raíz



El objeto raíz (V114 en la imagen, nombre del modelo de vehículo) es el objeto que contiene al resto de objetos. Al ser el objeto base, este objeto contiene la propiedad de “Rigidbody”. Esta propiedad es propia de Unity, y gracias a ella, podemos darle al vehículo las propiedades físicas que deseemos. Podemos asignarle una masa (en kg), podemos darle una fuerza de arrastre y arrastre angular, podemos decir si al objeto le afecta la gravedad, etc. En el caso de este videojuego, todos los vehículos tienen un arrastre o “Drag” de 0.02 y un arrastre angular o “Angular Drag” de 0.05. La masa variará según el modelo, pudiendo ser 1500 kg, 2000 kg o 2500 kg.

Al Rigidbody se le asigna el centro de masa. Esto influye en el comportamiento del vehículo, sobre todo en los giros.

Este objeto también tiene adjuntado el sistema de IA, el “Car Controller”, el “Race Driver” y la fuente de audio. Todas estas partes se explicarán más adelante.



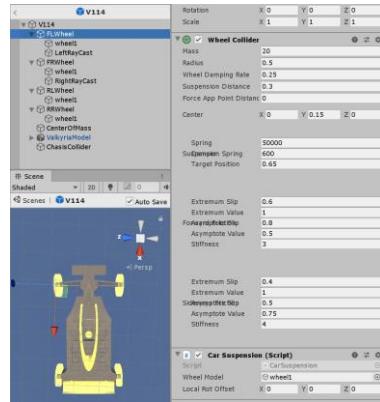
#### Ruedas

Cada una de las cuatro ruedas tiene adjuntado un “WheelCollider”, y un sistema de suspensión. Las cuatro ruedas tienen un hijo el cual es el modelo visible de la rueda (“wheel1” en la imagen). Adicionalmente, las dos ruedas delanteras tienen un sistema de raytracing para detectar el borde del circuito (“LeftRayCast” y “RightRayCast”).

El WheelCollider es un colisionador especial de Unity para vehículos de tierra. Tiene una detección de colisión integrado, física de ruedas, y un modelo de deslizamiento basado en la fricción de la llanta. Puede ser utilizado para objetos que no sean ruedas, pero es específicamente diseñado para vehículos con ruedas.

Los WheelColliders son los objetos que nos permite enviarle a Unity la señal de que queremos acelerar, frenar o girar las ruedas del vehículo.

Cada rueda delantera, además de un WheelCollider, tiene como objeto hijo un sistema de RayTracing para detectar bordes. Este sistema se explicará en el capítulo de Inteligencia Artificial.

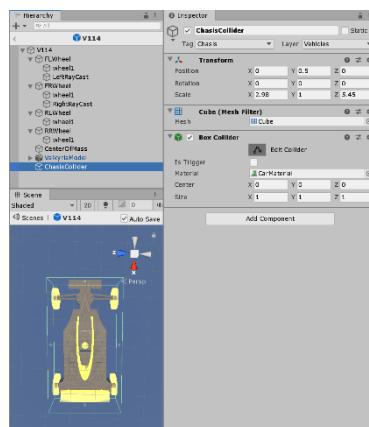


### Centro de masa

No contiene nada, excepto las coordenadas de su posición. Se le asigna al Rigidbody. Dependiendo de la posición del centro de masa respecto al vehículo, éste actuará de forma diferente en los giros. Por ejemplo, un centro de masa demasiado adelantado puede hacer que el vehículo empiece a dar vueltas en un giro brusco. Un centro de masa demasiado alto, puede hacer que el vehículo de vueltas de campana en las curvas.

### Colisionador de Chasis (Chassis Collider)

Caja invisible que delimita los choques de otros vehículos. Sin este colisionador, los vehículos podrían traspasarse entre ellos.



#### 5.2.2 Características de cada modelo

En total tenemos 54 modelos, pues tenemos 6 modelos por marca. Cada uno de los modelos se diferencia en una serie de características:

- Potencia: Puede ser entre 6 y 14. La potencia, como más adelante se verá, es clave en la aceleración del vehículo.
- Masa: Puede ser de 1500kg, 2000kg o 2500kg. Menor masa implica más aceleración, pero también es más difícil de controlar.
- Agarre: Cada WheelCollider tiene una serie de características modificables, de las cuales una de ellas es el “Stiffness” o rigidez. En los modelos, esta rigidez, que se traduce en agarre, puede tener un valor de 3 o de 4. La diferencia entre estos valores es muy notable en una carrera, es por ello que un modelo exactamente igual que otro pero con menor agarre, tendrá un multiplicador de puntuación mucho mayor.

Las combinaciones de características en cada una de las marcas no son en absoluto aleatorias:

Las marcas Duck, Audidas y Hoa tienen, cada una, dos modelos con potencia 6, dos con potencia 7 y dos con potencia 8.

Las marcas Valkyria, XLynx y DJED tienen, cada una, dos modelos con potencia 9, dos con potencia 10, y dos con potencia 11.

Las marcas Raijin, Poseidon y Leviathan tienen, cada una, dos modelos con potencia 12, dos con potencia 13 y dos con potencia 14.

Todos los modelos de las marcas Duck, Valkyria y Raijin tienen una masa de 1500kg.

Todos los modelos de las marcas Audidas, XLynx y Poseidon tienen una masa de 2000kg.

Todos los modelos de las marcas Hoa, DJED y Leviathan tienen una masa de 2500kg.

En cada marca, para cada par de modelos con la misma potencia, uno tiene agarre de 3 y otro agarre de 4.

Ya que se ha descrito en detalle las características de cada modelo, vamos a explicar cómo calculamos el multiplicador de puntuación de modelo (cada modelo tiene un multiplicador de puntuación, que se multiplica con los puntos finales de cada carrera. Esto favorece la elección de coches no demasiado buenos a favor de más monedas y puntos).

La fórmula es la siguiente.

$$\text{multPotencia} = 2.25f - (\text{potencia} - 6) * 0.25$$

$$\text{multAgarre} = 1 + 0.5 * (4 - \text{agarre})$$

$$\text{multMasa} = \begin{cases} 0.75 & \text{si masa} = 1500\text{kg} \\ 1 & \text{si masa} = 2000\text{kg} \\ 1.5 & \text{si masa} = 2500\text{kg} \end{cases}$$

$$\text{multiplicador} = \text{multPotencia} * \text{multAgarre} * \text{multMasa}$$

### 5.2.3 Controlador de vehículo

El elemento llamado “CarController” que está adjunto con el objeto raíz del vehículo, es el que se encarga de transmitir las señales del jugador o IA a las WheelColliders.



El funcionamiento es el siguiente.

En cada frame del videojuego, se reproduce el siguiente proceso, dando lugar a un bucle infinito que nos permite manejar el vehículo:

#### Obtención de señales

En primer lugar, se calculamos la velocidad del vehículo. Esta velocidad será la que se nos muestre en el velocímetro. El Rigidbody tiene un atributo de velocidad en unidades por segundo. Como en este videojuego consideramos cada unidad un metro, simplemente pasamos este valor a km/h multiplicándolo por 3.6.

Una vez hecho esto, obtenemos los Inputs o señales que nos proporciona la IA o el jugador. Estas señales son la aceleración, el freno, el boost, y el giro.

En el caso del jugador, las señales de aceleración, freno, boost y giro se obtienen desde el teclado. Todas estas señales tienen un valor de entre 0 y 1.

En el caso de la IA, las señales se obtienen desde el objeto “CarAI” adjunto al objeto raíz que vimos en el apartado anterior. El valor de aceleración usando la IA es siempre 1, el valor de freno está entre 0 y 2, y el de boost y giro entre -1 y 1.

Estas señales las obtenemos en bruto, y hay que transformarlas en valores que el controlador sepa manejar.

#### Tratado de señales

El controlador transforma estas cuatro señales en tres: Acelerador, cuyo valor se encuentra entre [-1,1]; dirección, cuyo valor se debe encontrar entre los límites del ángulo de giro (si el ángulo es 20 por ejemplo, entre [-20,20]), boosting, el cual es un valor booleano, por lo que puede ser 0 (si no se está usando el boost) y 1 (si se está usando el boost).

Como las señales obtenidas por el jugador o por la IA son distintas, hay que tratarlas de forma distintas.

## *Jugador*

En el caso del jugador:

El acelerador se calcula restando la señal de aceleración y la de freno, resultando en 1 si la aceleración es total (1) y el freno es nulo (0), y en -1 si no hay aceleración (0) y el freno es total (1).

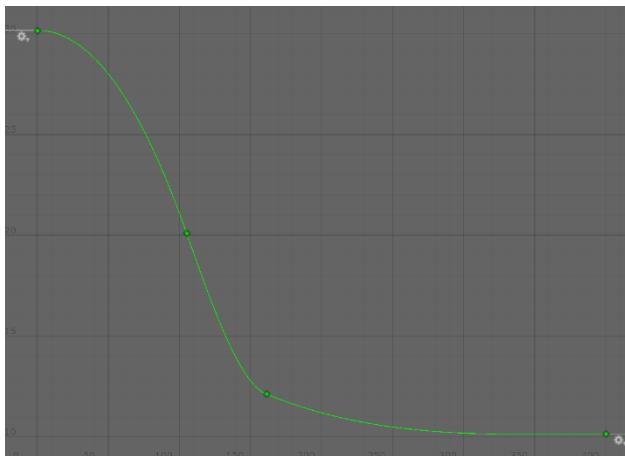
El boosting se activa si el valor de boost es mayor de 0.5.

La dirección es algo más compleja. En este videojuego se buscaba que, como en los coches reales, el ángulo máximo de giro vaya disminuyendo conforme aumenta la velocidad. Esto se usa para evitar “volantazos” a altas velocidades.

Por tanto, primero se transforma la señal de giro de [0,1] a [-1,1], y al resultado se le aplica esta curva. La fórmula sería la siguiente:

$$\text{dirección} = (\text{giro} - 0.5) * 2 * \text{ángulo (velocidad)}$$

Siendo *ángulo* la función siguiente:



Si nos fijamos en los números de la imagen, vemos que el dominio de la función se encuentra en [0,400] y que el valor máximo es 30, y el mínimo 10. En esta función, donde el eje x representa la velocidad, y el eje y representa el ángulo máximo de giro, vemos que a velocidad 0 tenemos un giro de 30 grados, y que, al ir avanzando la velocidad, este ángulo va disminuyendo, hasta llegar al mínimo de 10 grados a altas velocidades.

## *Inteligencia Artificial*

La transformación de señales en el caso de la IA es muy parecida:

El acelerador se obtiene de la misma forma que en el caso del jugador, restando la aceleración y el frenado.

El boosting se activa si boost es mayor que 0.

La dirección se obtiene con la siguiente función:

$$\text{dirección} = \text{giro} * \text{ángulo (velocidad)}$$

Vemos que es la misma que en el caso del jugador, con la diferencia de que el giro ya se nos da entre [-1,1].

### Aplicación de señales

Ya tenemos los valores obtenidos a partir de las señales enviadas por el jugador o la IA, ahora se aplican a las ruedas, concretamente a las WheelColliders de éstas.

#### Dirección

En nuestro caso sólo giran las ruedas delanteras, por tanto, a los WheelCollider de las ruedas delanteras se le asigna el valor de dirección. Una vez asignado este valor, las ruedas del coche pasarán a tener la posición girada los ángulos que el valor le haya indicado. Por ejemplo, si el valor es -25, las ruedas pasarán a estar giradas 25 grados hacia la izquierda.

#### Acelerador

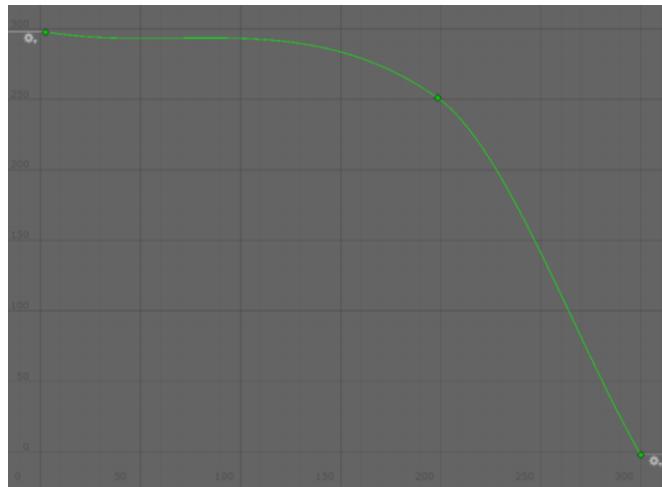
En nuestro caso las ruedas motor son también las delanteras, por lo que la aceleración y el freno se le aplicarán a éstas:

Si el acelerador es mayor que 0, significa que queremos acelerar. Por tanto, al par motor de cada rueda se le asigna el valor dado por la siguiente función:

$$\text{par motor} = \text{acelerador} * \text{potencia} * \text{motor(velocidad)}$$

La potencia es un valor distinto para cada modelo de vehículo. El modelo menos potente tiene una potencia de 6, y el más potente tiene una potencia de 14.

La función *motor* la usamos para que la aceleración no sea uniforme: Los coches reales aceleran mucho más rápido de 0 km/h a 100 km/h que de 100 km/h a 200 km/h, por ejemplo. Por tanto, esta función es la siguiente:



El eje x representa la velocidad del vehículo, y el eje y representa el valor que devuelve la función. Por tanto, vemos que a velocidades menores de 150 el valor devuelto se acerca a 300, mientras que a altas velocidades este valor cae en picado. Gracias a esto, por ejemplo, imposibilitamos que el vehículo acelere en una recta hasta velocidades inverosímiles.

Si el acelerador es menor que 0, significa que queremos frenar. Para ello, al freno de las ruedas se le asigna el valor dado por la siguiente función:

$$\text{freno} = |\text{acelerador}| * \text{fuerza de frenado}$$

La fuerza de frenado es un valor igual en todos los modelos, y es de 100.000.

### *Boosting*

El boosting se aplica de la siguiente manera.

El boost funciona aplicándole una fuerza externa hacia delante al Rigidbody adjunto al vehículo.

Si en el frame anterior se estaba usando boost, y el depósito no está vacío, se aplica en este frame también.

Si en el frame anterior no se estaba usando boost, pero el depósito está a más de la mitad, el boost se puede aplicar en este frame.

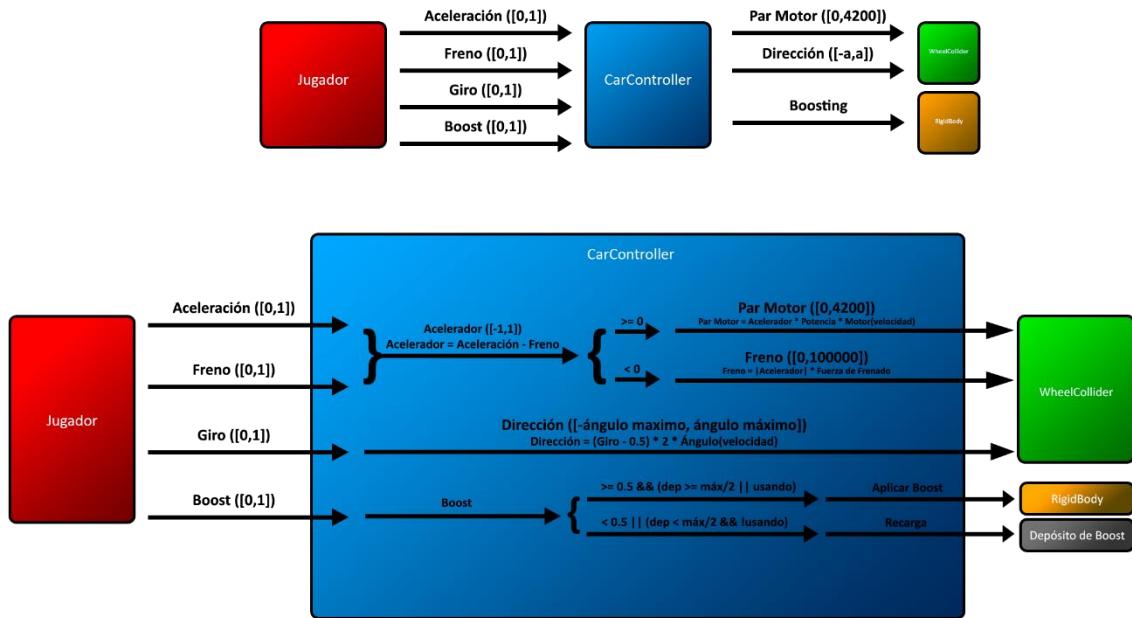
Si en el frame anterior no se estaba usando boost, y el depósito está a menos de la mitad, el boost no se puede aplicar y habrá que esperar a que el depósito se recarge a más de la mitad.

En los frames que no se usa el boost, el depósito se va recargando poco a poco.

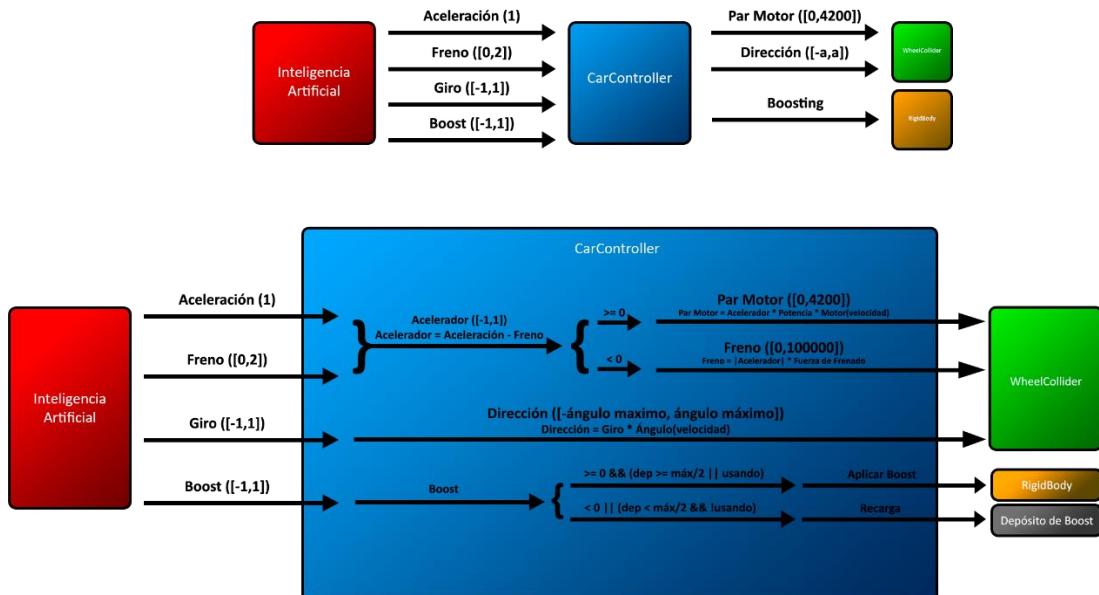
En las siguientes imágenes podemos ver un resumen del proceso del controlador del coche.

Recordamos que este proceso se repite iterativamente en cada frame del videojuego, de forma que el efecto a la larga es el de un vehículo que completa el circuito en el menor tiempo posible.

## Resumen de controlador de vehículo (Jugador)



## Resumen de controlador de vehículo (Inteligencia Artificial)



## 5.3 Modelado

En este apartado se van a detallar todos los elementos 3D que se han usado en el videojuego, incluyendo coches y circuitos. Aunque todos los coches y circuitos se han modelado desde cero,

hay algún que otro objeto (árboles y rocas) que se han obtenido de paquetes gratuitos. Estos objetos también se especificarán aquí.

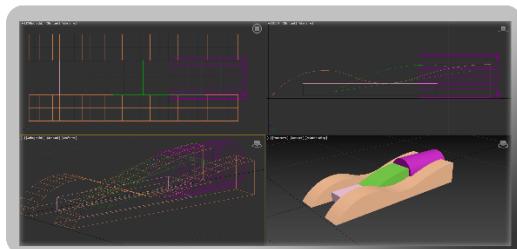
Puesto que los modelos necesitan al menos un material adjunto para que sean visibles, la gestión de materiales y texturas también las explicaré en este apartado.

Para modelar, se ha usado el software 3DS Max, el cual no es gratuito, pero he podido usar con una licencia de estudiante.

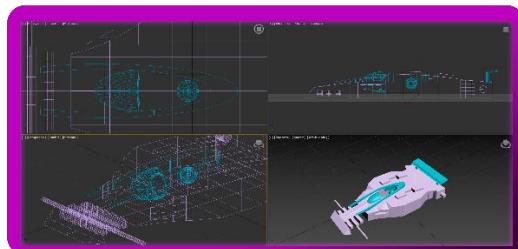
### 5.3.1 Coches

Todos los coches han sido modelados desde cero. Se han modelado un total de nueve coches distintos, uno para cada marca. Al ser novato en el campo del modelado, los coches no tienen una calidad extraordinaria, pero creo que el resultado final ha sido bueno. Casi todos los modelos de coches están compuestos por varias piezas. Esto es útil para aplicar distintos materiales a cada una de ellas, y que visualmente los coches sean más atractivos.

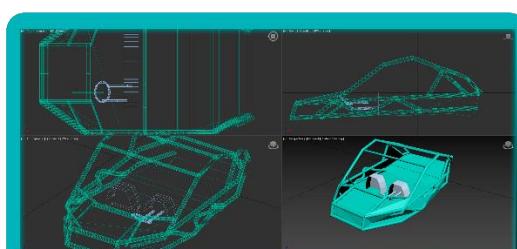
**AUDIDAS**



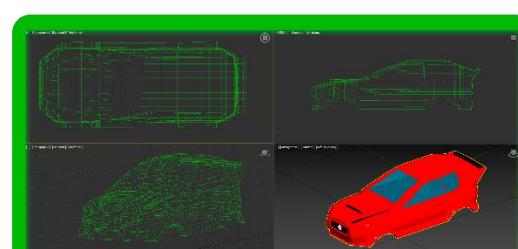
**DJED**



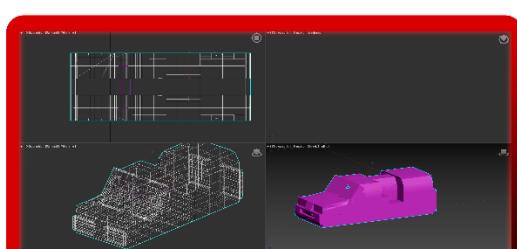
**DUCK**



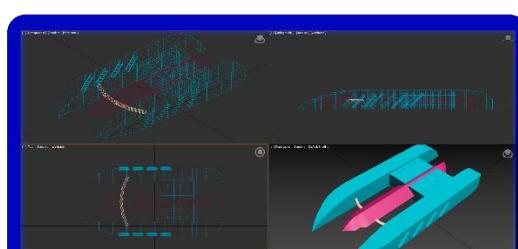
**HOA**

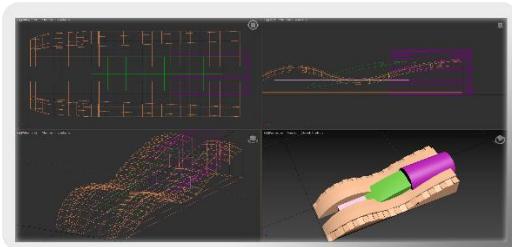
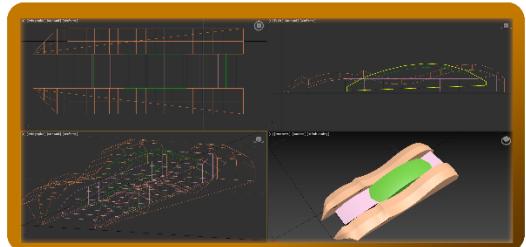
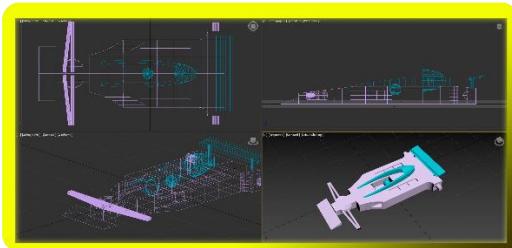


**LEVIATHAN**



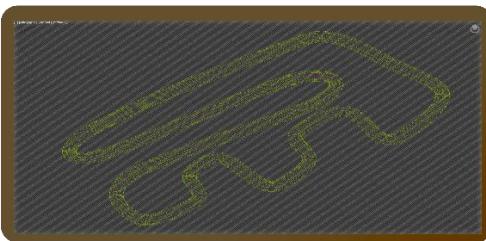
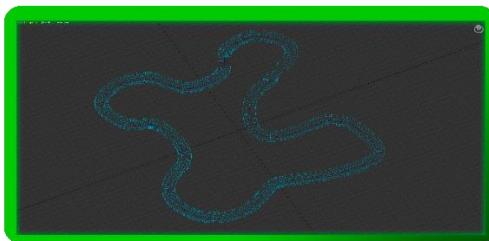
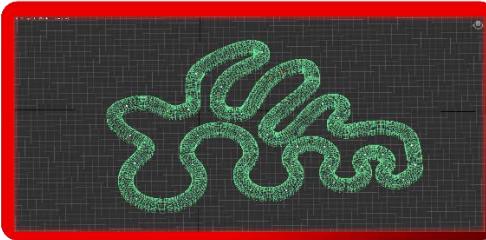
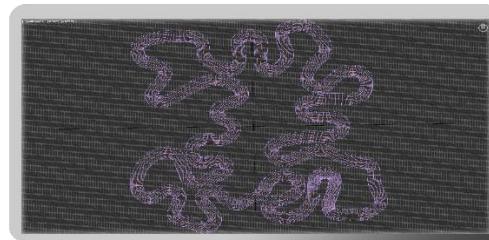
**POSEIDON**



**RAIJIN****XLYNX****VALKYRIA**

### 5.3.2 Circuitos o mapas

Se han modelado un total de cuatro circuitos distintos, cada uno con una ambientación diferente. Para ello, se ha usado el mismo software, 3DS Max.

**DIZZY****EIGHT****SUBWAY****WHIRL**

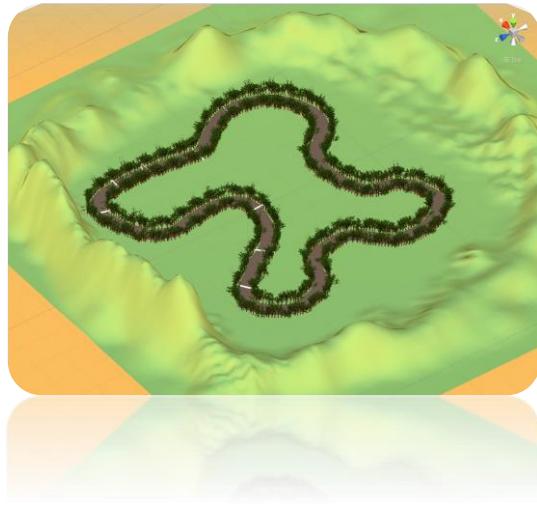
Cada circuito se compone de dos piezas principales: La carretera y el borde. La carretera es la parte donde los coches se apoyan y se mueven, y el borde delimitará la carretera. Este borde no se puede traspasar, por lo que ningún coche puede salir del circuito. Además, la IA necesitan al borde para saber su posición respecto al circuito. Esto se explicará en el capítulo de Inteligencia Artificial.

### 5.3.3 Escenas

Ya se han enseñado los modelos más importantes, que son los coches y circuitos. Pero el modelado va mucho mas allá. En este apartado se explica cómo hemos usado los modelos en cada escena.

#### EightCircuit

Vamos a ver todo lo relacionado con los modelos en el circuito “Eight”.



Vemos que además del circuito, también hay más cosas. Tenemos una serie de montañas rodeando el circuito, pero esto no se ha hecho con 3DS Max, sino con el propio Unity. Unity tiene una opción para crear terrenos donde se pueden hacer montañas relativamente fácil.

También vemos bastantes árboles rodeando el circuito. Estos árboles los he obtenido de la “Asset Store”. La “Asset Store” es una especie de tienda que Unity tiene integrada, donde hay objetos, materiales, etc. algunos de ellos gratuitos. En este caso, los árboles obtenidos pertenecen al paquete “NatureStarterKit2” y son de libre uso.



#### DizzyCircuit

Vamos a ver todo lo relacionado con los modelos del circuito “Dizzy”.

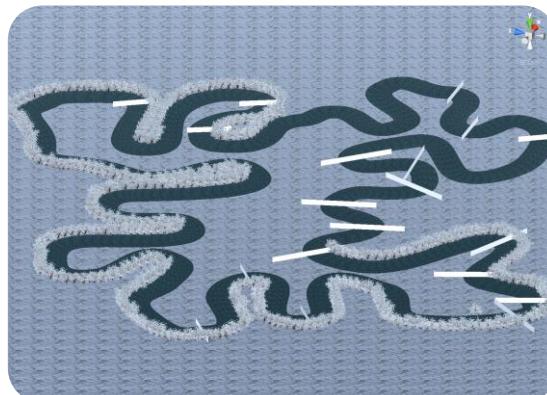


Como vemos, en esta escena no hay montañas ni árboles, pero sí que hay bastantes rocas. Las rocas las he obtenido, al igual que los árboles, de la "Asset Store". El paquete en concreto se llama "FreeRocks", y su contenido es de libre uso.



### WhirlCircuit

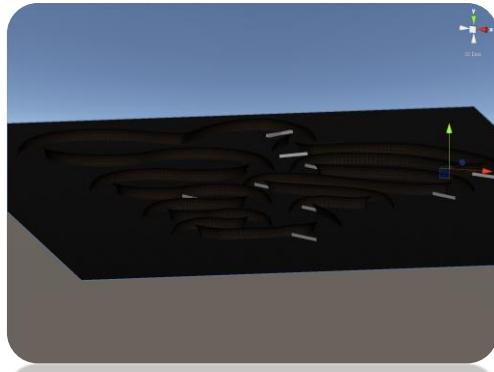
Vamos a ver todo lo relacionado con los modelos del circuito "Whirl".



En este caso hemos usado los mismos árboles que en el circuito "Eight", pero en este caso se les ha cambiado el color a blanco.

## SubwayCircuit

Vamos a ver todo lo relacionado con los modelos del circuito “Subway”.



En este circuito hay un modelo adicional en el circuito, un túnel. Todo el circuito está recubierto con un túnel de su misma forma. Este túnel está modelado con 3DS Max. Como fuera del túnel el jugador no puede ver nada, no es necesario modelar nada fuera.

### 5.3.3.1 Menú Principal

Los circuitos no son las únicas escenas que tienen modelos. Y el menú principal tiene varios.



Vemos que hay varios modelos usados en el menú principal. El más visible es el “HD” gigante (Iniciales de Hell’s Driver), pero también hay tres coches sobre un minicircuito circular.

### 5.3.4 Otros objetos

Hay algunos objetos más que se han modelado. Son los siguientes.



*Plataforma de la escena “Seleccionar Coche”*

La plataforma giratoria donde se apoyan cada uno de los coches, también se ha modelado con 3DS Max



## *Medallas*

Al finalizar la carrera, si hemos quedado terceros o mejor, nos aparece una animación de una medalla, de oro, plata o bronce, según nuestra posición. Estas medallas también se han modelado con 3DS Max.

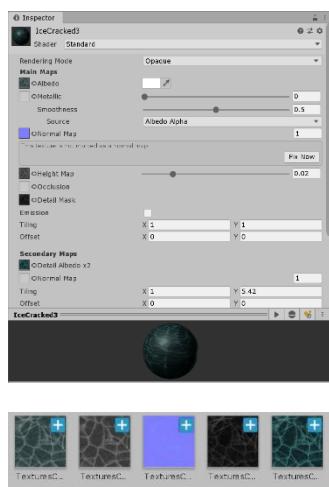
### 5.3.5 Texturas y materiales

Para cada uno de los modelos empleados, entre otros objetos, se han requerido de distintos materiales. La cantidad de materiales empleada es tan grande que no es conveniente plasmarlos en la memoria uno a uno.



Los materiales son objetos que se crean en Unity. Un material se puede crear de muchas formas, pero la más común es a partir de una textura o imagen. La mayoría de texturas las he obtenido de la página web [www.textures.com](http://www.textures.com). Estas texturas son de libre uso y se pueden incluir en el videojuego sin problemas de copyright.

Cada textura está compuesta por entre cuatro y seis imágenes distintas, cada una con su función: Albedo, metálica, mapa de normales, mapa de altura, oclusión ambiental, y máscara de detalles



La imagen albedo controla el color base de la textura. Es la imagen en la cual el resto de imágenes se aplican.

La imagen metálica se usa en texturas donde se busca un brillo o reflejo metálico. Esta imagen controla la reflectividad y la respuesta de la luz en la superficie de la textura.

El mapa de normales es un tipo de mapa de relieve. Es un tipo especial de textura que permite añadir detalles de superficie como grietas, agujeros, etc.

El mapa de altura es usado normalmente junto al mapa de normales, y permiten dar mayor definición y calidad a la superficie.

El mapa de oclusión es usado para proveer información sobre qué áreas del modelo deben recibir alta o baja luz indirecta. La luz indirecta proviene de la luz Ambiental y de reflejos.

El mapa de detalles permite añadir detalles adicionales a la textura

## 5.4 Interfaces de Usuario

En este apartado se explicará el desarrollo de las distintas interfaces de usuario del videojuego (menú, selección de coche, marcador de velocidad y posición en carrera, etc.). La tarea de diseñar interfaces de usuario no es exclusivamente visual, ya que hay que integrarla con el videojuego, de forma que las interfaces sean interactivas (por ejemplo, que los botones funcionen). En este módulo por lo tanto no se incluye solo el diseño visual de las IU, sino que también su funcionalidad.

En la creación de interfaces de usuario se han usado bastantes imágenes de [www.pngtree.com](http://www.pngtree.com). Esta página web da acceso a imágenes sin copyright, con la única condición de citar al autor en los proyectos donde usemos la imagen. En este caso, todos los autores están citados en la bibliografía.

### 5.4.1 Botones

El recurso más usado en este proyecto. Hemos visto que en este videojuego hay varios menús y pantallas con cantidad de botones, y todos se rigen por las mismas reglas.

Un botón tiene tres estados: Sin seleccionar, destacado y seleccionado. Un botón está sin seleccionar cuando no interactuamos con él y no está seleccionado. La mayoría de botones están sin seleccionar. Un botón está destacado cuando pasamos el ratón por encima. Que un botón esté destacado, nos dice que, si pinchamos, el botón desencadenará una acción. Un botón está seleccionado cuando hemos pinchado el botón. Aquí debemos dividir los botones en dos tipos: Los que si los seleccionamos se mantienen seleccionados, como los botones de selección de dificultad, y los botones que al seleccionarlos se deseleccionan al instante, como el botón de "Carrera Rápida", o el botón de ir hacia atrás. Estos botones suelen deseleccionarse porque desencadenan alguna acción visual. En la imagen de la derecha podemos ver un botón desseleccionado, destacado y seleccionado respectivamente.



Botón desseleccionado, destacado y seleccionado

Para implementar esta funcionalidad de estados, Unity tiene métodos específicos que se pueden sobrescribir con la funcionalidad que deseemos. Para realizar el efecto visual de cambiar el color del botón al destacarlo o seleccionarlo, basta con cambiar la imagen del botón en cada uno de estos métodos.

El reto viene cuando tenemos varios botones, en los cuales si seleccionamos uno los demás se deben desseleccionar. Esto ocurre varias veces, en la selección de dificultad, de mapa, de modelo, etc.

El ejemplo de la selección de dificultad es claro: Tenemos seleccionado el botón 2 (lo sabemos porque se mantiene el borde amarillo) y tenemos el ratón sobre el botón 3. Si hacemos clic, el botón 3 pasaría a resaltarse en amarillo y el botón 2 volvería a su estado desseleccionado (azul).



Botones para elegir dificultad. El 2 está seleccionado, y el 3 está destacado

Esto se resuelve de una manera sencilla con un Script que gestione los botones.

Para la funcionalidad de los botones, por tanto, se han programado dos Scripts distintos: Un Script de botón, que se adjunta una copia a cada botón, y un Script de administración de botones, que gestiona estos scripts de botones.

Por tanto, todos y cada uno de los botones del videojuego tiene adjunto un Script que hace lo siguiente:

- Cuando se pasa el ratón por encima, si no está seleccionado, se cambia su imagen por la imagen correspondiente al botón destacado.
- Cuando se quita el ratón de encima del botón, si no está seleccionado, se vuelve a cambiar su imagen por la imagen correspondiente al botón deseleccionado.
- Cuando pinchamos en el botón, si no está seleccionado, se cambia su imagen por la imagen correspondiente al botón seleccionado, y se marca como seleccionado.

Cuando además queremos gestionar varios botones donde si seleccionamos uno se deselecciona el que estaba seleccionado, usamos el Script de administración de botones, el cual selecciona el botón clicado y deselecciona el botón anteriormente seleccionado.

Evidentemente los botones, cuando se seleccionan, realizan más acciones además de cambiar su imagen: Los botones del menú abren una escena, los de mapa o dificultad guardan información que será rescatada luego para cargar la dificultad y el mapa correspondiente, etc.

#### 5.4.2 Selección de marca y modelo

La selección de marca es bastante especial, porque es la única selección del videojuego que no funciona con botones. Aunque la selección del modelo de cada marca sí que funciona con botones, la marca se elige pulsando las teclas derecha e izquierda.



Selección de marca de coche

El funcionamiento de esta interfaz no es tan simple como parece. Como se busca crear el efecto óptico de que las marcas están en fila horizontal, una detrás de otra, debemos gestionar las imágenes de una forma algo peculiar. El sistema funciona de la siguiente manera: Tenemos cinco imágenes, una para cada marca. Las tres del centro, que son las que vemos, y una oculta a cada lado. Cuando pulsamos derecha (para seleccionar la marca de la derecha), todas las imágenes se van moviendo hacia la izquierda a una velocidad constante, hasta que la marca de la derecha queda en el centro (seleccionada). Por tanto, a la quinta imagen

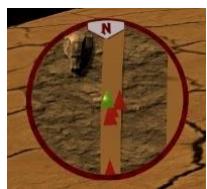
(la de la derecha) debemos asignarle el archivo de la marca que va detrás de la cuarta imagen. La primera imagen, la que está detrás del bloque negro, la movemos hacia la derecha del todo, y la segunda imagen, la que se ocultará cuando acabe la animación detrás del bloque negro, pasaría a ser la primera imagen. Para crear el efecto óptico de ver la imagen central más grande, la imagen que pasa del centro a la izquierda se va reescalando para hacerse cada vez más pequeña, y la imagen de la derecha se va reescalando haciéndose más grande conforme se va posicionando en el centro.

El proceso para cambiar hacia la izquierda es el mismo, solo que de forma invertida.

De esta forma obtenemos un efecto óptico de ir deslizando entre marcas, desde la primera hasta la novena.

#### 5.4.3 Minimap

El minimapa que vemos en la carrera abajo a la izquierda, nos ayuda a orientarnos y a saber el trazado del circuito próximo.



El funcionamiento del minimapa no es más que una cámara adicional sobre nuestro coche. Esta cámara está apuntando hacia abajo, y su posición es actualizada cada frame para que esté siempre vertical al coche que pilotamos. Para que sea más realista, la rotación de la cámara también es simétrica a la del vehículo. De esta forma, cuando giremos, en el minimapa tendremos la sensación de que es el mapa el que está rotando. Para que la brújula siempre señale al norte, simplemente rotamos la “N” los mismos grados pero en sentido contrario a la cámara.

#### 5.4.4 Velocímetro



El velocímetro nos indica dos cosas: La velocidad a la que vamos, y el boost que tenemos. La velocidad del vehículo se muestra tanto en el marcador digital como por la aguja, mientras que el boost se muestra en el arco verde alrededor del velocímetro. Conforme vamos usando boost, este arco se va haciendo cada vez más corto, y vuelve a alargarse si dejamos que el boost se recarge.

#### 5.4.5 Posición de carrera, y número de vuelta



Se actualizan conforme adelantamos o hacemos una vuelta.

### 5.5 Sonido

El módulo de sonido en este videojuego es el módulo menos relevante de todos, pues sólo se han usado tres tipos de sonido: Las canciones que suenan de fondo, los pitidos del inicio de la carrera y el sonido de los coches. Los pitidos de inicio y el sonido de los coches se han obtenido de [www.zapsplat.com](http://www.zapsplat.com) y [www.freesound.org](http://www.freesound.org). Los sonidos de los coches simulan a un coche arrancado en punto muerto en bucle, por lo que para simular el sonido de un coche acelerando y decelerando, se le ajusta el pitch o tono para que se haga más agudo conforme más rápido vaya el coche. Usando este método tan simple, obtenemos resultados bastante buenos.

Las canciones que suenan en los menús de fondo, y en las carreras con menos volumen, pertenecen al grupo BettoGh. Este grupo de música synthwave (género de electrónica) ofrece algunas de sus canciones sin copyright, para que las podamos usar en proyectos como videojuegos.

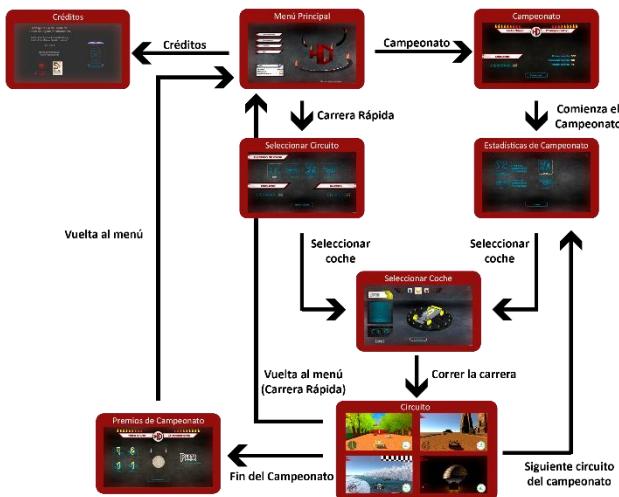
### 5.6 Programación del videojuego

Todos y cada uno de los módulos explicados anteriormente no son de utilidad por sí solos, ni se puede crear un videojuego sin una programación detrás que dé sentido a cada uno de los componentes.

Este proyecto cuenta con unos 60 scripts distintos y aproximadamente 3800 líneas de código (casi 8000 si contamos las líneas en blanco), por lo que es inviable plasmar al detalle la programación de este videojuego. Sin embargo, y sin entrar a nivel de clases, se puede explicar de forma genérica qué es lo que hacemos detrás de todos los coches, circuitos, botones y otros elementos que conforman este videojuego.

Por tanto, en este apartado se procederá a explicar la programación detrás de cada escena, dejando la Inteligencia Artificial para el siguiente capítulo.

Refresquemos las distintas escenas que tenemos en el videojuego:



### 5.6.1 Menú principal (Gestión de perfiles)

Aunque parezca que el menú principal es una de las escenas con menos programación detrás, el detalle de que la gestión de perfiles ocurra en esta escena engorda el número de líneas de código necesarias para un correcto funcionamiento.

En este videojuego podemos tener un máximo de 6 perfiles, y esto hay que gestionarlo.

Cada perfil debe tener un nickname, un número de monedas, puntos totales, y porcentaje de modelos de coches desbloqueados, así como una lista de los modelos de coches que están desbloqueados. Estos datos deben ser permanentes, es decir, si cerramos el juego y lo volvemos a abrir, no pueden desaparecer. Para ello hay que guardar localmente esta información. Unity cuenta con herramientas para ello, que, en caso de Windows, guarda esta información en el registro.

Para cada perfil podemos añadir monedas y puntos (se usa cuando ganamos carreras o campeonatos), podemos gastar monedas, por ejemplo, comprando coches, se puede desbloquear un modelo de coche permanentemente y se puede calcular el porcentaje de coches desbloqueados.

Podemos crear un nuevo perfil, el cual no tendrá monedas ni puntos, pero tiene algunos coches desbloqueados por defecto. También podemos eliminar un perfil, eliminando sus datos permanentemente.

### 5.6.2 Seleccionar Circuito y Campeonato

Estas son las escenas con menos programación detrás. Simplemente se guarda localmente la información de qué nivel de dificultad, circuito, número de vueltas, etc. para obtener esta información en las siguientes escenas (En Unity, al cambiar de escena, toda la información en tiempo de ejecución se pierde, por lo que hay que guardar lo necesario localmente).

### 5.6.3 Seleccionar Coche

Si estamos en el modo Carrera Rápida, las únicas funciones destacables son las de cambiar los seis modelos para elegir cada vez que cambiemos de marca. En el modo Campeonato, se añade la funcionalidad de comprar coches. Si estamos en este modo y decidimos comprar un coche, primero se comprueba si el perfil tiene suficientes monedas. Si las tiene, se procede a restarlas del perfil, y se le envía al perfil la información de que se desea desbloquear ese coche permanentemente. Una vez desbloqueado el coche, siempre que abramos el juego con ese perfil, tendremos ese coche desbloqueado.

### 5.6.4 Programación en las carreras

Saber qué coche va en cabeza, cuál va a la cola, saber cuándo hacer respawn de un coche, y otras muchas cosas son retos que necesitamos resolver. Como en este apartado hay que explicar muchas cosas, vamos a separarlo en subcategorías.

#### 5.6.4.1 *Inicio de la carrera y elección de los corredores*

Lo primero que se hace al comenzar la carrera es cargar los perfiles de usuario. Recordemos que, en los cambios de escena, todos los datos cargados en memoria desaparecen, por lo que hay que volver a cargar los perfiles almacenados localmente.

Una vez hecho esto, se cargan cada uno de los modelos con su respectiva IA. No voy a entrar mucho en detalle todavía, pero cada modelo ha sido entrenado para cada uno de los cuatro circuitos, por lo que tiene una IA específica. Por tanto, cargamos 54 modelos y 54 IAs distintas, una para cada modelo.

Esta lista de modelos está ordenada según el tiempo que tarda en completar el circuito.

Ya tenemos los perfiles cargados, y también los modelos. Ahora se inicializan los seis corredores.

Aquí se debe cargar el modelo de coche que hayamos elegido, y otros cinco rivales. Pero, ¿Qué cinco? Esto dependerá del nivel de dificultad que hayamos escogido.

Recordemos que hay cinco niveles de dificultad (más adelante se explicará la dificultad dinámica) y 54 modelos ordenados de mejor a peor tiempo. La solución es intuitiva. Si nuestro nivel de dificultad es 5, cogemos cinco modelos aleatorios entre los diez mejores. Si nuestro nivel de dificultad es 4, cogemos cinco modelos aleatorios entre las posiciones 20 y 11. Si nuestro nivel de dificultad es 3 o 2, los cogemos entre las posiciones 30 y 21 o 40 y 31, y si nuestro nivel de dificultad es 1, cogemos cinco aleatorios entre las últimas 14 posiciones.

De esta forma, nos aseguramos que el nivel de nuestros rivales concuerda con el nivel de dificultad elegido.

Una vez tenemos nuestros seis modelos elegidos, los inicializamos y los creamos en la línea de salida del circuito. Empieza la cuenta atrás. Nos aparece la animación: 3, 2, 1... Go! La carrera empieza.

#### 5.6.4.2 Control de posiciones

Tenemos una lista con los seis corredores, los cuales están actualmente corriendo y adelantándose continuamente. Nuestro objetivo es, mantener la lista ordenada de menor a mayor posición.

Parece una tarea simple, pero saber en qué posición está cada coche, y actualizarla prácticamente al instante contando los adelantamientos por un centímetro, requiere de algún sistema rápido y fiable.



CheckPoints en naranja

Para resolver esta tarea, necesitamos de CheckPoints. Un CheckPoint es una zona del circuito por donde el coche debe pasar. Normalmente los CheckPoints en muchos videojuegos se usan para que al jugador no se le ocurra tomar "atajos" y salirse del circuito para ir por el césped hasta la meta en línea recta, por ejemplo. Pero en nuestro caso, lo usaremos (además) para otras cosas.

Esta es una imagen del circuito Eight, la cual tiene resaltados en naranja todos los CheckPoints.

Con ayuda de los CheckPoints, podemos comparar dos corredores y saber cuál está más adelantado. El algoritmo es el siguiente.

Dados dos corredores, A y B:

Si A y B llevan distintas vueltas, el que lleve más vueltas está más adelantado

Si A y B llevan las mismas vueltas, el que lleve más CheckPoints está más adelantado

Si A y B llevan el mismo número de vueltas y CheckPoints, el que más cerca esté del siguiente CheckPoint está más adelantado.

Si ordenamos la lista de corredores con este criterio en cada frame, obtenemos las posiciones de forma instantánea.

Este sistema no es perfecto, por ejemplo si hay pocos CheckPoints y estamos en una curva, podríamos momentáneamente perder posición sin ser adelantados, sin embargo nos aseguramos de lo importante: El primero que llega a la meta, gana.

#### 5.6.4.3 Control de colisiones y respawns

La IA no es perfecta, y a veces los corredores colisionan entre ellos, de forma que de vez en cuando alguno se queda parado sin saber cómo continuar. Para solventar estas situaciones, se ha hecho un colchón de seguridad: estos coches respawnan en el último CheckPoint atravesado.

El respawn se produce por dos razones: Si un coche se ha quedado parado (por colisión o por alguna otra razón), y si un coche pasa dos veces por un mismo CheckPoint (lo que quiere decir que va en dirección contraria).

Para saber si un coche se ha quedado parado, comprobamos si el coche lleva teniendo menos de x velocidad durante más de y tiempo. Si se cumplen estas condiciones, se hace respawn. En el videojuego estos valores son menos de 5km/h y más de 5 segundos.

El proceso de respawneo consiste en mover el coche a la posición del último CheckPoint cruzado, pero no solo debemos cambiar la posición, también la rotación. Además, debemos reestablecer su velocidad y velocidad angular a cero. También se desactiva el Collider unos cinco segundos. Esto es para evitar colisiones espontáneas nada más respawnear, y por si varios coches respawnean en el mismo punto a la vez.

#### 5.6.4.4 Dificultad dinámica

La dificultad dinámica es un tipo de dificultad diferente a la escala 1-5 que antes hemos presentado. Este tipo de dificultad se caracteriza por que el nivel del rival se ajusta al nuestro. Si nuestro nivel desborda a los rivales y en la carrera vamos en cabeza, el nivel de los rivales aumentará. En cambio, si ocurre, al contrario, y vamos últimos, el nivel de los rivales disminuirá.

De esta forma, se consigue que la carrera suponga un reto durante toda su duración, habiendo dificultades y posibilidades para ganarla hasta el final.

Para conseguir este efecto, usamos la lista de modelos de coches antes mencionada. Recordamos que esta lista contiene todos los modelos de coches con sus respectivas IAs, ordenados de mejor a peor tiempo. Esta lista la dividimos en cinco secciones, una para cada nivel de dificultad (1,2,3,4,5).

Con la dificultad dinámica, los rivales empiezan con modelos pertenecientes a la dificultad 2. En la carrera, entonces, pueden ocurrir dos cosas. Que vayamos con mucha ventaja, lo que aumentará el nivel de dificultad, o que vayamos últimos por mucho, lo que disminuirá el nivel de dificultad.

El videojuego considera que vamos con mucha ventaja cuando vamos dos CheckPoints por delante del segundo corredor, y considera que vamos con desventaja cuando vamos últimos a dos CheckPoints por detrás del penúltimo corredor.

Para subir o bajar el nivel, cambiar visualmente el modelo de los rivales quedaría extraño, por tanto, el modelo visual se mantiene, pero cambian sus características. Por ejemplo, si subimos de dificultad 2 a 3, todos los modelos de los rivales se mantienen, pero adquieren potencia, peso y agarre de modelos de dificultad 3, así como su IA correspondiente. De esta forma se puede subir y bajar de dificultad sin que cambie la forma de los vehículos de los rivales espontáneamente.

# 6 Inteligencia Artificial

Llegamos al capítulo de Inteligencia Artificial. Este capítulo posiblemente sea el capítulo más importante de esta memoria, ya que la Inteligencia Artificial ha sido la piedra angular de este Trabajo de Fin de Grado.

De nada sirve, en un videojuego, un modelado fantástico con unas escenas increíbles y una jugabilidad perfecta, si el videojuego no supone un reto. Que un videojuego suponga un reto es lo que hace que nos enganchemos a él, pues aviva nuestro interés por superarlo. Sin un reto, el videojuego se hace aburrido y monótono.

En un videojuego de carreras de coches el principal reto que tiene el jugador es evidente: Ganar la carrera. Para que el videojuego que estamos desarrollando sea desafiante, tenemos que desarrollar una Inteligencia Artificial para los corredores rivales que controlen el vehículo de forma que nos hagan difícil la victoria.

El funcionamiento de la Inteligencia Artificial que controla cada vehículo se puede resumir en la siguiente frase: Controlar el giro de forma reactiva, y controla la aceleración, freno y boost con una red neuronal entrenada con algoritmos genéticos, usando el método NEAT.

Dos líneas con un trasfondo tan profundo y extenso, que darán para páginas y páginas de explicaciones y esquemas.

## 6.1 Conceptos básicos

Para explicar a fondo qué hemos hecho, primero debemos explicar algunos conceptos que se necesitan saber para entender todo correctamente.

### 6.1.1 ¿Qué es la Inteligencia Artificial?

[4] La Inteligencia Artificial es una rama de las ciencias de la computación cuyo objetivo es crear máquinas inteligentes. En los últimos años esta rama ha cogido tal importancia que se ha convertido en una parte esencial de la industria de la tecnología. Los principales problemas que se buscan resolver con Inteligencia Artificial suelen cumplir ciertos rasgos, como pueden ser:

- Conocimiento
- Razonamiento
- Resolución de problemas
- Percepción del entorno
- “Machine Learning” o aprendizaje automático
- Planeamiento
- Habilidad para manipular y mover objetos

Nuestra Inteligencia Artificial deberá tocar varios de estos campos para cumplir su objetivo.

La ingeniería del conocimiento es una parte básica de la Inteligencia Artificial. Las máquinas a veces pueden actuar y reaccionar como humanos sólo si tienen abundante información de su entorno. La Inteligencia Artificial debe tener acceso a estos objetos, categorías, propiedades y relaciones entre todos ellos para implementar ingeniería del conocimiento. El sentido común, razonamiento y poder solucionar problemas son tareas difíciles y tediosas para una máquina.

El aprendizaje automático es otro de los principales campos de la ciencia de la Inteligencia Artificial. Aprender sin ningún tipo de supervisión requiere la habilidad de identificar patrones en una gran cantidad de inputs, mientras que aprender con una adecuada supervisión es adecuado para resolver problemas de clasificación o regresión numérica. Los problemas de clasificación consisten en determinar la categoría a la que pertenece un objeto, mientras que la regresión numérica trata con, a partir de un set de ejemplos de inputs y/o outputs, descubrir funciones que predigan con precisión outputs para inputs nunca antes analizados. En el caso de aprendizaje por refuerzo, la máquina aprende de los refuerzos de la práctica basándose en los éxitos o fracasos que tenga.

El campo de percepción del entorno trata de la capacidad de usar sensores para deducir distintos aspectos del mundo, mientras que la visión por computador trata de analizar inputs visuales (imágenes) y resolver ciertos problemas relacionados con estos, como reconocimiento facial, de gestos, etc.

La robótica es también uno de los grandes campos relacionados con la Inteligencia Artificial. Los robots requieren inteligencia para manejar tareas como manipulación de objetos y navegación, y resolver problemas de localización, entre otras cosas.

El campo de la Inteligencia Artificial se podría desarrollar muchísimo más, pero este es un buen resumen.

### 6.1.2 ¿Qué es una red neuronal?

Antes hemos dejado caer que las redes neuronales son parte de la Inteligencia Artificial que cada corredor tiene integrada. “Red neuronal” son dos palabras que todo el mundo ha oído de pasada, pero no es algo trivial. El campo de las redes neuronales es un campo cuya extensión y profundidad es equivalente a su potencia y posibilidades en el mundo de la Inteligencia Artificial.

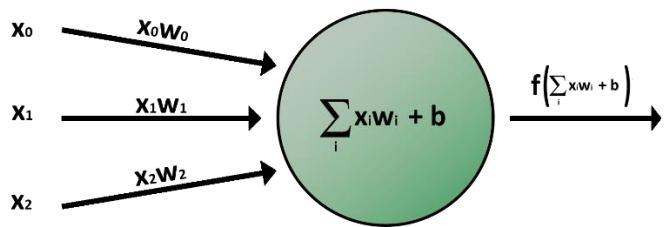
Las redes neuronales son técnicas de Machine Learning muy potentes y ciertamente complicadas, que simulan en ciertos sentidos el cerebro humano y cómo éste funciona.

Igual que nuestro cerebro tiene millones de neuronas y una red que las interconecta entre ellas gracias a los axones, las cuales se comunican por señales eléctricas de una capa a otra. A esto lo llamamos sinapsis. Gracias a esto, el ser humano es capaz de aprender. Todo lo que vemos, oímos, sentimos y pensamos es de cierta manera un conjunto de impulsos eléctricos que es disparado de una neurona a otra en una jerarquía que nos permite aprender, recordar y memorizar cosas en nuestra vida diaria desde el día en que nacemos.

Dejando de lado el romanticismo, una red neuronal es una caja negra formada por un conjunto de nodos o neuronas de los cuales, dados unos inputs, obtenemos unos outputs. Para explicar el funcionamiento de una red neuronal, vamos a explicar primero el concepto de nodo o neurona:

Una neurona es una función. Esta función recibe uno o varios parámetros de entrada en forma de valores numéricos. Cada uno de estos valores son multiplicados por una constante distinta para cada uno. Estas constantes son llamadas pesos. Una vez multiplicados, se hace una sumatoria de todos estos valores, y se le suma una constante llamada bias.

A la cifra resultante, se le aplica una función concreta llamada función de activación, y el resultado final es lo que conocemos como output.



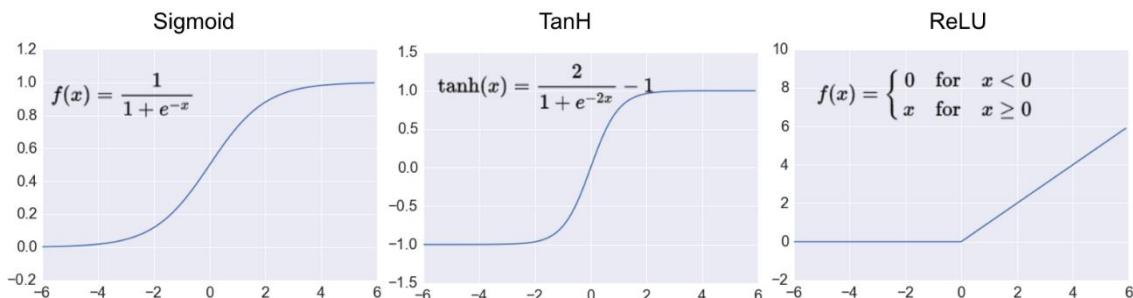
*Neurona perteneciente a una red neuronal*

Como vemos en la imagen, a los inputs ( $x_0$ ,  $x_1$  y  $x_2$ ) se les multiplica un peso, se hace la sumatoria, a ésta se le añade un bias, y se le aplica la función de activación.

Las funciones de activación son realmente importantes en una red neuronal para que ésta pueda aprender y darle sentido a cosas realmente complicadas como complejas funciones no lineales entre inputs y outputs. Su tarea es transformar las señales de input de una neurona en un output que sea de utilidad para las siguientes neuronas, y pueda usarse de input en éstas.

Pero, ¿por qué no obtener el output sin usar la función de activación sobre los inputs? Porque entonces obtendríamos simplemente una función lineal. Las funciones lineales están bastante limitadas si buscamos aprender complejas funciones a partir de datos.

Hay infinidad de funciones lineales distintas. Las más comunes son la Sigmoid, la Tanh (Tangente hiperbólica) y la ReLU.

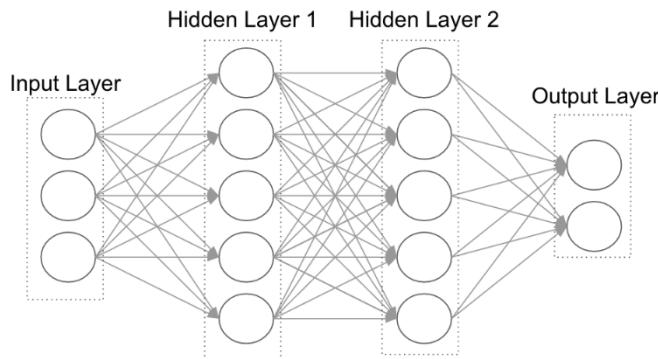


[5] Distintas funciones de activación

Ya hemos explicado qué es una neurona y cómo funciona, ahora explicaremos qué es una red neuronal.

Las redes neuronales son un conjunto de neuronas que se pueden clasificar en tres tipos de capas: Capa de Inputs, capa de Hidden (Ocultas) y capa de Outputs. En las redes neuronales, los inputs que obtienen del exterior entran por las neuronas de la capa de Inputs, los outputs de esta capa van a la capa de Hidden, y por último a la capa de Outputs.

En la imagen de abajo tenemos un ejemplo de red neuronal de tipo “Feedforward” (Sólo hacia delante).



[5] Red Neuronal

En el ejemplo de la red neuronal de la imagen, recibiría tres inputs distintos, y devolvería dos outputs. La capa central (Hidden u oculta) se llama así porque a la hora de usar una red neuronal, lo único con lo que interaccionamos es con la capa de Inputs y de Outputs, la capa de neuronas ocultas para nosotros es una caja negra.

Ya sabemos cómo funciona una red neuronal, y sabemos que para resolver ciertos problemas puede ser muy potente. Pero no sabemos algo fundamental: ¿Qué pesos usamos? ¿Cuántas capas, y de qué tamaño? No basta con cifras aleatorias, para que los outputs tengan sentido, los pesos y el resto de hiperparámetros deben ser elegidos cuidadosamente.

El proceso de encontrar estos pesos e hiperparámetros es lo que se conoce como proceso de aprendizaje de la red neuronal.

Hay infinidad de técnicas para entrenar una red neuronal. El más común es el algoritmo de back-propagation o propagación hacia atrás, el cual es un tipo de aprendizaje supervisado.

En el caso de este proyecto no se ha usado esta técnica, las redes neuronales han sido entrenadas usando algoritmos genéticos, concretamente usando el algoritmo NEAT.

Antes de entrar en detalle con este algoritmo, conviene explicar qué es un algoritmo genético.

#### 6.1.3 ¿Qué es un algoritmo genético?

Un algoritmo genético es una búsqueda heurística inspirada por la teoría de evolución natural de Charles Darwin. Este algoritmo refleja el proceso de selección natural donde los individuos que mejor se adaptan al medio son seleccionados para su reproducción en orden de producir mejores genes para la siguiente generación.

El proceso de selección natural empieza con la selección de los individuos mejor adaptados al medio (con mejor fitness). Estos individuos se reproducirán entre ellos traspasando sus genes a la siguiente generación. Manteniendo este proceso de forma iterativa hasta el final, cada generación será mejor que la anterior, acercándonos cada vez más a individuos cuya adaptación al medio sea perfecta.

Estos conceptos pueden ser usados en problemas de búsqueda.

En los algoritmos genéticos, hay cinco fases que se repiten de forma iterativa:

1. Obtención de la población inicial
2. Aplicación de la función de fitness
3. Selección
4. Crossover (Reproducción)
5. Mutación
6. Reemplazo

#### Población Inicial

El proceso empieza con un conjunto de individuos el cual es llamado Población. Cada individuo es una solución para el problema que buscamos resolver.

Un individuo está caracterizado por una serie de parámetros (variables) a los que llamamos genes. El conjunto de genes forma lo que llamamos cromosoma, el cual es una solución.

#### Aplicación de la función de fitness

La función de fitness determina cómo de bien un individuo se adapta al medio. Esta función nos da una puntuación de fitness para cada individuo. La probabilidad de que un individuo sea seleccionado para su reproducción viene dada por el fitness.

#### Selección

La idea de la fase de selección es seleccionar los individuos con mejor fitness para que éstos dejen sus genes para la siguiente generación. Los individuos con mayor fitness tienen más probabilidad de ser escogidos para la reproducción

#### Crossover (Reproducción)

La fase de reproducción es la fase más significativa en un algoritmo genético. Para cada par de padres seleccionados para reproducirse, se escogen ciertos genes de un parente y del otro y se mezclan creando hijos. La selección y mezcla de genes se puede hacer de diversas formas.

#### Mutación

Una vez nuevos hijos son creados, a algunos de ellos se les aplica una mutación. Una mutación es un cambio aleatorio en alguno de sus genes. El objetivo de las mutaciones es mantener la diversidad entre la población y prevenir de convergencia prematura.

#### Reemplazo

El reemplazo consiste en obtener una nueva generación inicial a partir de los hijos ya mutados. De esta forma repetimos todas las fases en bucle hasta encontrar una solución cuyo fitness sea el adecuado para nuestro problema.

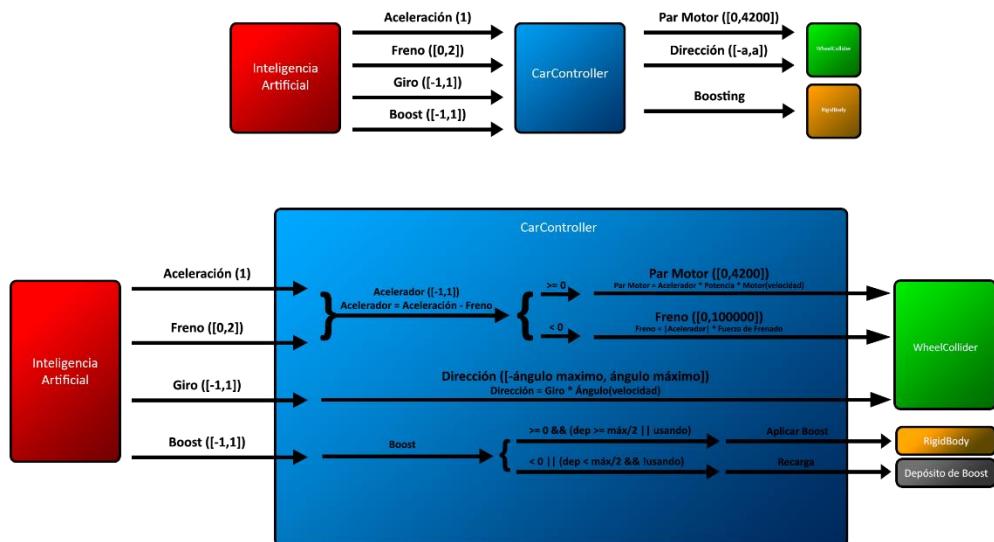
El algoritmo termina si la población ha convergido (no se producen hijos significativamente diferentes de los padres), o si se ha encontrado una solución óptima.

Ya se han explicado unos cuantos conceptos básicos que viene bien tener claro. Ahora se procederá a explicar en qué consiste la Inteligencia Artificial de este Trabajo de Fin de Grado.

## 6.2 Inteligencia Artificial en el vehículo

Recordemos el objetivo de la Inteligencia Artificial en cada vehículo: Completar el circuito en el menor tiempo posible. Para ello, la IA de un modelo debe proporcionarle al mecanismo de control de ese coche los valores necesarios para que este mecanismo de control lleve al coche a la meta.

Vamos a recordar el diagrama visto en el capítulo de diseño.



La parte del control del vehículo ya la tenemos resuelta. En este capítulo, lo que nos interesa es lo siguiente:



Por cuestiones de comodidad se ha elegido que la aceleración sea siempre 1, por lo que la inteligencia artificial se tiene que ocupar de ofrecerle al controlador del coche tres valores: Freno, giro y boost.

Cabe recordar que estas variables se le ofrecen al controlador del coche en cada frame, es decir, en cada frame del videojuego el controlador está obteniendo unas variables distintas, y las aplica al vehículo de forma que, a la larga, el vehículo llegue a la meta en el menor tiempo posible.

Encontrar sólo tres valores a priori parece una tarea sencilla, pero si nos paramos a pensar, empezamos a ver dificultades: ¿Cuánto frenamos en una curva donde el coche puede derrapar si la pasa demasiado rápido? Si tenemos otro coche delante, ¿Cuánto frenamos y cuánto giramos para adelantarlo? ¿Cuánto aceleramos si nos acercamos a una curva muy cerrada?

Vemos que, si nos detenemos en estas cuestiones, las tres variables que parecían pocas, empiezan a dar mucho juego.

En este proyecto, la solución empleada para encontrar una Inteligencia Artificial efectiva se divide en dos ramas: Para obtener el giro hemos usado una técnica puramente reactiva, mientras que para el freno y boost hemos usado una red neuronal entrenada con un algoritmo genético, NEAT.

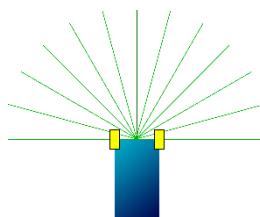


### 6.2.1 Inteligencia Artificial para el giro del vehículo

El problema que debemos solucionar aquí es cuántos grados girar las ruedas de un vehículo para recorrer la trazada de una forma eficiente y sin colisionar con los bordes del circuito o (en la medida de lo posible) con otros vehículos.

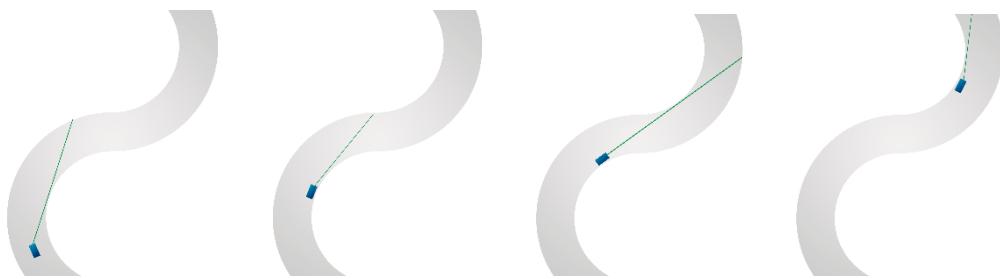
Dentro del campo de los agentes reactivos, la solución más intuitiva que me vino a la cabeza fue una solución tan simple como evidente: Apuntar las ruedas hacia el punto del circuito más lejano. A priori, este razonamiento tiene sentido: Dado que la línea recta es la distancia más corta entre dos puntos, cuanto más recta/con curvas menos pronunciadas intentemos realizar la trazada, menos tiempo tardará el vehículo en llegar a su destino.

Para llevar a cabo esta idea podemos poner en el vehículo unos sensores que detecten la distancia del circuito cada x ángulo, de la siguiente forma:

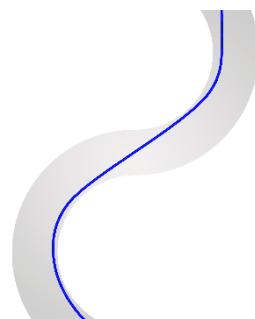


Unity tiene una herramienta que hace exactamente esto, llamada RayCast. Esta herramienta, como su nombre indica, realiza raycasting. Esto consiste, en resumen, en un rayo que sale desde un origen, en una dirección, y con una longitud, y nos da información del objeto con el que ha colisionado. Entre esta información está la distancia. De esta forma podemos usar una serie de RayCasts en el vehículo para que nos diga la distancia del borde en cada uno de los puntos.

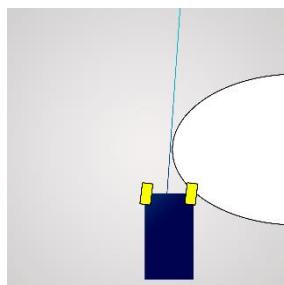
Vamos a ver unos ejemplos de cómo giraría el vehículo usando esta técnica con imágenes de unos bocetos:



En estas cuatro imágenes podemos ver a un vehículo (rectángulo verde) en dos curvas. La línea verde señala al punto más lejano, por tanto, es el punto donde el vehículo se dirige. Este punto no es el mismo siempre, se va actualizando en cada frame de forma que el resultado final sería una trazada parecida a la siguiente:



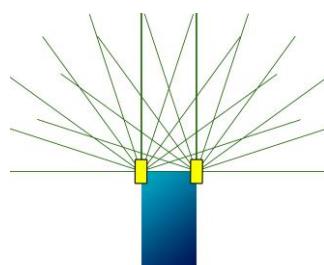
Parece que hemos solucionado el problema del giro... ¿No? Vamos a ver el siguiente caso:



En este caso, vemos como el coche encuentra el punto más lejano. Sin embargo, ir hacia ese punto va a acabar inevitablemente en un choque contra el muro del circuito.

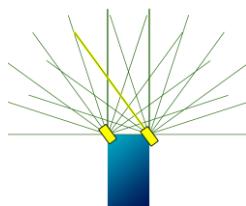
El problema que tenemos es que los sensores no tienen en cuenta el tamaño del vehículo.

La solución final que se ha dado para el problema es un poco más compleja. En vez de hacer raycasting desde el centro, se va a hacer desde las dos esquinas del Collider del vehículo. Concretamente 11 sensores van a salir desde cada esquina:



En total tenemos un número de 22 sensores, separados en dos grupos: 11 en la esquina izquierda, y 11 en la esquina derecha. De esta forma, cubrimos las distancias cada 18 grados para cada grupo. El algoritmo cambia significativamente: Ya no cogemos el punto más lejano. Primero cogemos el punto más lejano de cada grupo, comparamos, y nos quedamos con el menos lejano de los dos elegidos.

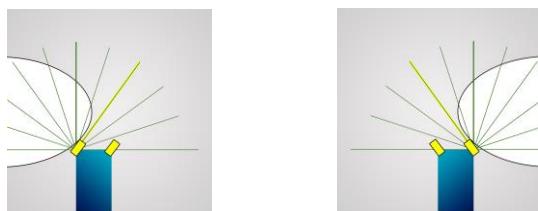
En este caso las ruedas no apuntarían exactamente al punto, si no que tomarían el mismo ángulo que el raycast elegido, como en la siguiente imagen.



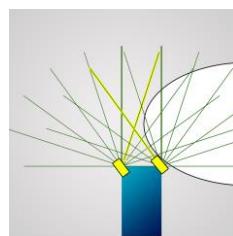
¿Por qué soluciona esto nuestro problema?

Si seguimos al punto más lejano del grupo izquierdo, nos aseguramos de que no nos vamos a chocar por la izquierda.

Si seguimos al punto más lejano del grupo derecho, nos aseguramos de que no nos vamos a chocar por la derecha.



Si de los dos grupos escogemos hacer caso al grupo cuya máxima distancia sea menor, evitaremos siempre el choque. En la imagen, por ejemplo, vemos que escogeríamos el grupo de la derecha, pues la distancia más lejana detectada del grupo de la derecha (sensor amarillo) es menor que la distancia más lejana del grupo de la izquierda. Por tanto, las ruedas tomarían una posición de 36 grados hacia la izquierda.



Pero, ¿qué pasa si la distancia más lejana del grupo de la izquierda es menor? Eso nunca va a pasar.

Para que este sistema funcione, el circuito debe cumplir una condición: El ancho del circuito debe ser igual en todo su recorrido.

Esto tiene fácil explicación: Si la curva es hacia la derecha, la parte de la izquierda del vehículo va a llegar a “ver” más lejos que la parte de la derecha. Por tanto, el grupo de sensores al que

haremos caso será al derecho. Ídem para las curvas hacia la izquierda. Esto cambiaría drásticamente si la anchura del circuito cambiara en según que tramos, pero no es el caso.

¿Por qué solo 11 sensores en cada grupo?

Evidentemente a más sensores más precisión en la trazada. La elección de solo 11 sensores ha sido por problemas de potencia de mi PC. El raycasting es un sistema que consume bastantes recursos, y más aún si se usa en cada frame. A esto hay que añadirle que, en el entrenamiento de la red neuronal, tenemos 100 coches entrenando, con 22 sensores cada uno. 2200 raycasts en cada frame es algo que mi ordenador no puede asumir. Sin embargo, con 11 sensores se obtienen muy buenos resultados, y si bien la trazada no es perfecta, se le acerca bastante.

### Adelantamientos

Los adelantamientos son un problema adicional en el videojuego, pero fundamental de resolver para que los vehículos sean suficientemente inteligentes.

La solución que he encontrado a este problema es más fácil de lo que parece: Tratar al resto de vehículos como borde. De esta forma, cuando un vehículo vaya a chocar a otro, al detectarlo como borde lo esquivará y adelantará con éxito.

Al principio pensé que sólo se trataran como borde si el coche de delante iba más despacio que el de atrás, pero tratándolos todos como borde indiferentemente de la velocidad mejora la fluidez en los adelantamientos y descongestiona un poco el circuito.

Esta solución es una solución que para mi sorpresa funciona bastante bien, con adelantamientos bastante fluidos y realistas. No quiere decir que no haya colisiones, pero hay pocas más de las inevitables.



Frames de un adelantamiento. El vehículo azul detecta al blanco como borde, y procede a adelantarlo

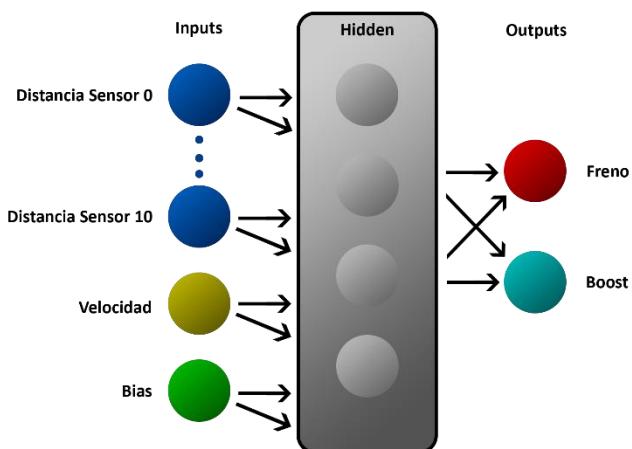
### 6.2.2 Inteligencia Artificial para el freno y boost del vehículo

Ya tenemos una variable resuelta, el giro, y puesto que la aceleración siempre es 1, nos quedan dos variables: Freno y boost. Quizás el boost no es demasiado importante, pero el freno lo es, y mucho. Del freno dependerá si el coche va tan lento que no supone un reto, o tan rápido que se choca en cada curva. Saber cuándo frenar y con qué fuerza, dependiendo de lo cerrada que sea la curva, hará que el corredor nos ponga las cosas difíciles o no.

Como ya se ha avanzado anteriormente, el freno y el boost serán controlados por una red neuronal. Cómo se ha entrenado la red se comentará más adelante.

Esta red neuronal tendrá como Outputs el freno, dado entre 0 y 2, y el boost, dado entre -1 y 1. Vamos a definir los inputs necesarios. Vamos a necesitar la velocidad del vehículo, ya que el freno actuará distinto dependiendo de la velocidad. Necesitaremos también la distancia de cada uno de los puntos detectados por los censores. No hacen falta los 22 censores y dos grupos, con un grupo basta. Por tanto, como inputs, se usarán los 11 sensores del grupo cuyo sensor se esté usando para girar. Se necesitará un input de bias. No se necesitará un input de la cantidad de boost restante, el propio entrenamiento de la red hará que el boost se use cuando sea conveniente, quede combustible o no.

Por tanto, la red neuronal sería la siguiente.



El funcionamiento es, iterativamente, obtener los outputs de freno y boost, aplicarlos junto con la aceleración y giro, y obtener las nuevas distancias y velocidad.

El proceso de entrenamiento se explicará en los siguientes apartados.

### 6.3 NEAT

[6] Evolving Neural Networks through Augmenting Topologies. Esto es NEAT. Que en español se traduce en evolucionar redes neuronales a través del aumento de la topología.

Una cuestión importante en neuro evolución es cómo ganar ventaja evolucionando la topología de las redes neuronales además de sus pesos. NEAT es un método, que (según el autor) tiene gran eficiencia gracias a usar un método de crossover de diferentes topologías, proteger innovación estructural usando especiación, e ir incrementando gradualmente la topología. NEAT ofrece la posibilidad para los algoritmos genéticos de optimizar y complejizar soluciones simultáneamente, ofreciendo la posibilidad de evolucionar incrementando la complejidad de las soluciones generación tras generación.

#### 6.3.1 Introducción

Neuro evolución (NE), la evolución artificial de redes neuronales usando algoritmos genéticos, han demostrado ser una gran promesa en complejas tareas de aprendizaje por refuerzo. Como

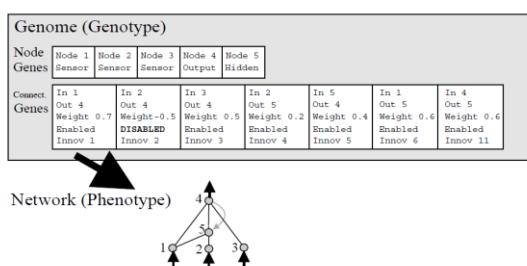
los NE se rigen por el comportamiento en vez de por una función que devuelve un valor, son muy efectivos en problemas continuos como el caso de recorrer un circuito.

En las NE tradicionales, una topología es predefinida para evolucionar la red neuronal antes de que el entrenamiento comience. Cabe recordar que la topología es la estructura que tiene la red: Número de capas, número de neuronas por capa, y conexiones entre éstas.

NEAT en cambio propone una alternativa: Comenzar por una red con el mínimo número de neuronas posible, e ir complejizándola generación tras generación.

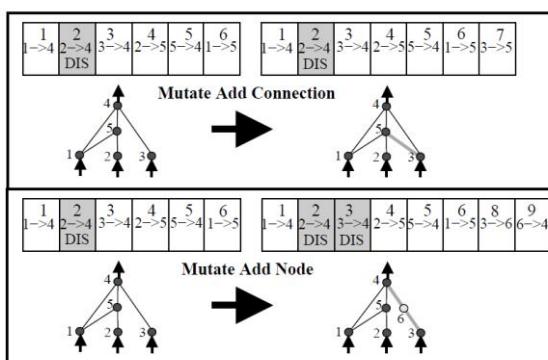
### 6.3.2 Codificación genética

El esquema de codificación genética está diseñado para permitir que la correspondencia entre genes sea fácil de encontrar cuando dos genomas se reproducen. Los genomas son la representación lineal de la conectividad de la red.



Cada genoma incluye una lista de conexiones de genes, la cual llamaremos genes de conexión, donde cada elemento representa a dos genes siendo conectados. Por genes entendemos neuronas o nodos. También tenemos una lista de genes, la cual llamaremos genes de nodo, y la cual está compuesta por todos los inputs, neuronas ocultas y outputs que se pueden conectar. Cada gen de conexión especifica el nodo de entrada, el nodo de salida, el peso de la conexión, si la conexión está o no activa, y un número de innovación, el cual nos permite encontrar genes correspondientes a éste, como se explicará más adelante.

La mutación en NEAT puede cambiar tanto los pesos de las conexiones como la topología de la red. El peso de las conexiones muta como en cualquier otro sistema NE, donde cada conexión puede ser alterada o no en cada generación. Las mutaciones estructurales ocurren de dos formas: Añadiendo un gen, o añadiendo una conexión.



En la mutación que consiste en añadir conexión, un nuevo gen de conexión con un peso aleatorio se añade conectando dos nodos que previamente no estaban conectados. En la mutación que consiste en añadir un nodo, una conexión ya existente es dividida en dos y un nuevo nodo ocupa lugar donde la vieja conexión solía estar. Esta vieja conexión es desactivada (no eliminada) y dos

nuevas conexiones son añadidas en el genoma. La nueva conexión que deriva en el nuevo nodo recibe un peso de 1, y la nueva conexión cuyo nodo de entrada es el nuevo nodo, recibe el mismo peso que la vieja conexión. Este método de añadir nodos ha sido elegido para minimizar el efecto inicial de la mutación. La nueva no-linealidad en la conexión cambia la función tangencialmente, pero los nuevos nodos pueden ser integrados inmediatamente en la red.

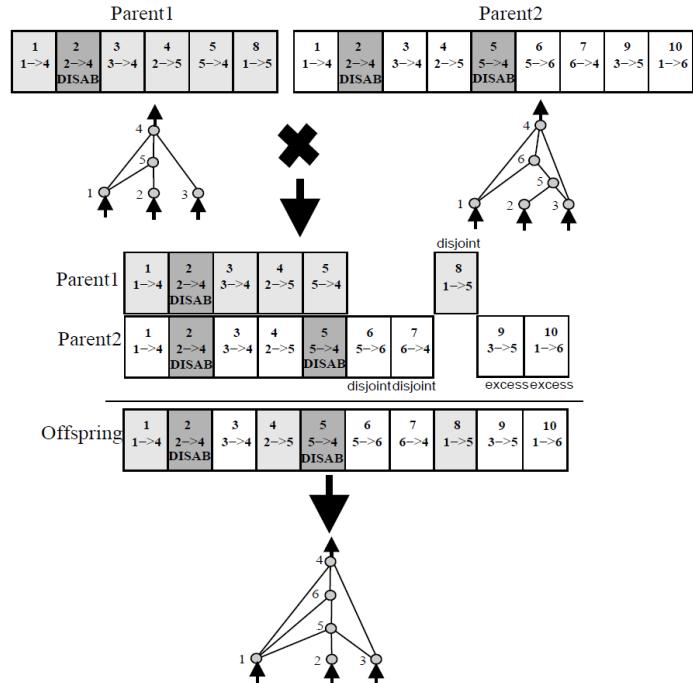
Con las mutaciones, los genomas serán cada vez más largos. Obtendremos genomas de diversos tamaños, algunas veces con diferentes conexiones en la misma posición. ¿Cómo podemos hacer un crossover con genomas de diferente tamaño de forma efectiva? En la siguiente sección se explica cómo resolvemos este problema.

### 6.3.3 Rastreando genes usando marcas históricas (Número de innovación)

Existe información en la evolución que nos dice exactamente qué genes hacen match con otros entre cualquier individuo de la población, por muy diferente que sea la topología. Esta información es el origen histórico de cada gen. Dos genes con el mismo origen histórico deben representar la misma estructura (no tienen por qué tener el mismo peso), ya que ambos derivan del mismo gene ancestral en algún punto del pasado. Por lo tanto, todo lo que un sistema debe hacer para saber qué genes se alinean con lo que es hacer un seguimiento del origen histórico de cada gen en el sistema. Rastrear el origen histórico requiere de muy poca computación. Cuando un gen nuevo aparece (a través de una mutación estructural), un número de innovación global es incrementado y asignado a ese gen. El número de innovación, pues, representa la cronología de la aparición de cada gen en el sistema. Como ejemplo, digamos que las dos mutaciones en la imagen anterior ocurrieron una detrás de otra en el sistema. El nuevo gen de conexión añadido durante la nueva mutación de nodo tiene asignado el número 7, y los dos nuevos genes de conexión añadidos durante la nueva mutación de nodo tienen asignados los números 8 y 9. En el futuro, cuando esos genomas se reproduzcan, el “offspring” tendrá de forma inherente el mismo número de innovación en cada gene; número de innovación nunca son cambiados.

Un posible problema es que la misma innovación estructural recibirá diferentes números de innovación en la misma generación si ocurre por casualidad más de una vez. Sin embargo, al mantener una lista de las innovaciones que se produjeron en la generación actual, es posible garantizar que cuando surja la misma estructura más de una vez a través de mutaciones independientes en la misma generación, a cada mutación idéntica se le asigne el mismo número de innovación. Por lo tanto, no hay una explosión resultante de números de innovación.

Las marcas históricas dan a NEAT una nueva y poderosa capacidad. El sistema ahora sabe exactamente qué genes coinciden con cuál (Imagen de abajo). Al cruzar, los genes en ambos genomas con los mismos números de innovación están alineados. Estos genes se llaman genes coincidentes. Los genes que no coinciden son disjuntos o excesivos, dependiendo de si ocurren dentro o fuera del rango de los números de innovación del otro parente. Estos, representan una estructura que no está presente en el otro genoma. Al componer la descendencia, los genes se eligen aleatoriamente de cualquiera de los progenitores para que coincidan con los genes, mientras que todos los genes excesivos o disjuntos siempre se incluyen del progenitor con mejor fitness.



De esta forma, las marcas históricas permiten a NEAT realizar crossover usando genomas lineares sin necesidad de costosos análisis topológicos.

Añadiendo nuevos genes a la población y reproduciendo genomas con diferentes estructuras, el sistema puede formar una población de diversidad topológica. Sin embargo, esto conlleva a que una población por sí sola no puede mantener innovaciones topológicas. Como pequeñas estructuras se optimizan más rápido que estructuras grandes, y añadiendo nodos y conexiones normalmente inicialmente decrementa el fitness de la red, las estructuras aumentadas recientemente tienen no demasiada esperanza de sobrevivir más de una generación incluso que la innovación que lleva pueda ser crucial a la larga. La solución es proteger la innovación especiando la población, como se explicará en la siguiente sección.

#### 6.3.4 Protegiendo las innovaciones a través de la especiación

Añadir el concepto de especie a la población permite a los organismos a competir primariamente con los de su misma especie en vez de con el resto de la población. De esta manera, las innovaciones topológicas están protegidas en un nuevo nicho donde tienen tiempo para optimizar su estructura a través de la competencia en ese nicho. La idea es dividir la población en especies de forma que similares topologías estén en la misma especie. Esta tarea parece ser un problema de comparar topologías. Sin embargo, gracias a las marcas históricas (número de innovación), tenemos una eficiente solución.

El número de genes excesivos y disjuntos entre un par de genomas es una forma natural de medir su distancia de compatibilidad. Cuanto más disjuntos están dos genomas, menos historia evolutiva comparten, por lo que menos compatibles son. Por tanto, podemos medir la distancia de compatibilidad de diferentes estructuras en NEAT como una simple combinación lineal del número de genes excesivos o disjuntos, así como la media de la diferencia de pesos de los genes compartidos, incluyendo los desactivados:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W}$$

Los coeficientes  $c_1$ ,  $c_2$  y  $c_3$  nos permiten ajustar la importancia de estos tres factores, y el factor  $N$ , el número de genes en el genoma más largo, normaliza el tamaño del genoma ( $N$  puede ser 1 si ambos genomas son pequeños, por ejemplo, menos de 20 genes).

La medida de distancia  $\delta$  nos permite especiar usando un límite de compatibilidad  $\delta_t$ .

Mantenemos una lista ordenada de especies. En cada generación, los genomas son secuencialmente colocados en especies. Cada especie existente es representada por un genoma aleatorio de la generación anterior de esa especie. Dado un genoma  $g$  en la generación actual, es colocado en la primera especie donde  $g$  sea compatible con el genoma representativo de esa especie. De esta forma, las especies no se sobreponen. Si  $g$  no es compatible con ninguna especie existente, una nueva especie es creada con  $g$  como representante.

Como mecanismo de reproducción de NEAT, usamos el “explicit fitness sharing”, o intercambio explícito de fitness, donde los organismos de la misma especie deben compartir el fitness de su nicho. De esta forma, una especie no se puede permitir convertirse demasiado grande incluso si muchos de sus organismos se desarrollan bien. De esta forma, ninguna especie tiene demasiadas posibilidades de quedarse con toda la población, lo cual es crucial para que la evolución especiada funcione. El fitness ajustado para cada organismo es calculado de acuerdo a la distancia  $\delta$  de cada otro organismo en la población:

$$f'_i = \frac{f_i}{\sum_{j=1}^n sh(\delta(i,j))}$$

La “sharing function”  $sh$  es 0 cuando la distancia está encima del límite  $\delta_t$ ; de otra forma,  $sh(\delta(i,j))$  es igual a 1. Si nos fijamos bien, el denominador de la fracción se reduce al número de organismos en la misma especie que el organismo  $i$ . Esta reducción es natural ya que las especies ya son agrupadas por compatibilidad usando el límite  $\delta_t$ . Cada especie tiene asignada un número de offsprings potencialmente diferente en proporción a la suma del fitness ajustado  $f'$ .

Las especies entonces se reproducen primero eliminando los miembros de la población con peor  $f'$ . La población entera es entonces reemplazada por los offsprings de los resultantes organismos de cada especie.

El efecto deseado de especiar la población es proteger las innovaciones topológicas. El objetivo final del sistema, entonces, es obtener una solución lo más eficientemente posible. Este objetivo es cumplido minimizando la dimensionalidad del espacio de búsqueda.

Merece la pena leer este apartado alguna vez más, ya que el sistema elegido tiene cierta complejidad. Para aclarar dudas, recomiendo leer el documento original donde se explica este algoritmo. Si bien este apartado ha sido una traducción casi literal del documento, es posible que mi traducción no sea demasiado buena, y leer el documento original pueda aclarar algunos conceptos.

## 6.4 Entrenamiento de la Red Neuronal

Ya se ha explicado en qué consiste NEAT. Como vemos es algo complejo, pero ofrece muchas posibilidades. En este apartado se va a explicar el proceso de entrenamiento de la red neuronal que utiliza cada vehículo.

### 6.4.1 Implementación de NEAT

NEAT se ha implementado en C# usando el sistema de Scripts de Unity. La implementación la he hecho desde cero, tomando como referencia el *paper* del autor. Esencialmente la implementación es exactamente igual que la descrita por el autor en su *paper*, pero con un sistema de crossover adicional. Además del sistema de crossover propuesto por el autor, se ha implementado (y usado) un crossover que consiste simplemente en, por cada individuo, obtener dos individuos: Una copia del mismo, y una copia mutada. Esto lo veremos más adelante.

### 6.4.2 Uso del algoritmo en nuestro problema

Aunque la implementación es esencialmente igual a la teoría expuesta anteriormente, vamos a intentar explicarla paso por paso para ver cómo hemos adaptado NEAT al entrenamiento de la IA de los vehículos.

#### Inicialización

Cada individuo de la población es una red neuronal que intentará conducir a su vehículo a la línea de meta sin colisiones de por medio. El primer paso del algoritmo, es crear esta población inicial de redes neuronales. Estas redes se crean de la forma más básica posible, sin capa de nodos ocultos, simplemente los inputs y outputs vistos anteriormente. La función de activación de las neuronas input es una función lineal:  $f(x) = x$  para todo  $x$ . La función de activación de la neurona de output de freno es la sigmoid, cuyo resultado se multiplica por dos. De esta forma obtenemos un valor entre 0 y 2. La función de activación de la neurona de output de boost es la tangente hiperbólica, pues buscamos un valor entre -1 y 1. Todas las input y output están interconectadas entre sí.

#### Obtención de fitness

Una vez tenemos nuestra población lista, vamos a obtener el fitness de cada individuo. El fitness se calcula de la siguiente manera: A toda la población de redes neuronales se le asigna un vehículo, siendo todos los vehículos del mismo modelo, y comienzan a correr el circuito a la vez. Los vehículos aparecen en el punto de salida, movidos hacia adelante o atrás, o izquierda o derecha una distancia aleatoria. El fitness es correlativo con el número de CheckPoints cruzados: A mayor número, mayor fitness. Si el número es igual, tendrá mayor fitness el que más distancia haya recorrido desde el último CheckPoint.

Esto genera un problema: Los vehículos que vayan tan lento que nunca se choquen, tendrán el mayor fitness, aunque tarden un tiempo excesivo. Esto lo corregimos multiplicando el fitness por un peso. Este peso es el porcentaje de tiempo que el vehículo ha mantenido el acelerador a fondo. Si el vehículo no ha dejado de acelerar en toda la carrera, este número será 1. Normalmente en los entrenamientos no se ven valores de 0.95, lo cual nos dice que el 95% del tiempo se acelera a fondo, lo cual es buena señal. Pero que no baje de este número no es casualidad. Para asegurarnos de que el peso no tiene un valor demasiado bajo, tenemos un límite: Si el peso es menor a ese límite, el fitness será cero.

Cuando un vehículo colisiona con el borde, su entrenamiento acaba y se le asigna el fitness. También hay un límite de tiempo de entrenamiento, por si hay coches que no se chocan (lo cual es lo que necesitamos). Además, también hay un límite de tiempo entre CheckPoints, de forma que un vehículo que se quede atascado finalice su entrenamiento.

Una vez todos los vehículos han concluido su entrenamiento, procedemos a evolucionar la generación.

#### Dividir en especies

El proceso de especiar es el mismo que el explicado en el apartado de NEAT, tenemos una distancia de compatibilidad, y unos representantes de las especies de la generación anterior. Para cada uno de los individuos, se comprueba su distancia con cada representante. En cuanto haya algún representante cuya distancia de compatibilidad sea menor al límite, se añade a esa especie. Si no hay ninguno, se crea una nueva especie.

#### Obtener a los campeones

Como una medida de elitismo, y para no perder a los mejores individuos de cada especie, guardamos en una lista al mejor individuo de cada especie. Esta lista se usará más tarde.

#### Crossover o reproducción

El primer paso es generar el “shared fitness” o fitness compartido, para evitar que haya especies que acaparen toda la población en número. Este fitness se calcula usando la fórmula vista anteriormente.

Una vez hecho esto, ordenamos a toda la población de mejor a peor individuo, y eliminamos a la mitad peor de toda la población (sin mirar especies, puede ser que haya especies que desaparezcan por completo). Ya podemos empezar el crossover.

Como se ha dicho antes, se han implementado dos formas de realizar el crossover.

Dada la forma original del autor, se escogen iterativamente parejas de la misma especie y se reproducen entre éstas, usando el método de crossover explicado anteriormente. Estos nuevos individuos se añaden a una nueva lista de hijos.

Dada la forma personalizada, no se usa el método de crossover del autor, si no que, para cada individuo, a la lista de hijos, se añade una copia idéntica, y otra mutada.

En ambos casos se añaden a la lista de hijos los campeones obtenidos en la etapa anterior.

#### Mutación

La mutación puede ser de tres tipos: Añadir una conexión a la red, añadir una neurona, o modificar un peso.

El método de añadir una conexión consiste simplemente en conectar dos neuronas aleatoriamente, teniendo cuidado de que no se formen ciclos infinitos.

El método de añadir una neurona consiste en dividir una conexión que exista actualmente, e insertar una neurona en medio. La anterior conexión queda deshabilitada, y las nuevas conexiones tienen un peso de 1 la anterior y el mismo peso que la vieja, la posterior.

El método de modificar un peso consiste bien en cambiar el peso por otro totalmente aleatorio, o bien en modificarlo en un ligero porcentaje.

Las probabilidades de cada tipo de mutación se especifican en los hiperparámetros.

### Reemplazo

Antes de reemplazar la población inicial por los hijos, se guardan representantes aleatorios de cada especie para especiar la generación siguiente. Una vez hecho esto, se reemplaza la población inicial, y volvemos al punto de partida con una generación un poco más evolucionada.

### Finalización del entrenamiento

El entrenamiento termina cuando se alcanza un límite de generaciones, u opcionalmente, cuando consigamos individuos que completen el circuito, o cuando llevemos x generaciones con individuos que hayan completado el circuito.

Una vez acabado el entrenamiento, obtenemos el mejor individuo, y codificamos la red neuronal para que se guarde en un archivo de texto, y se pueda leer en cualquier momento. Por tanto, las IAs de los modelos se guardan en archivos de texto.

## Esquema

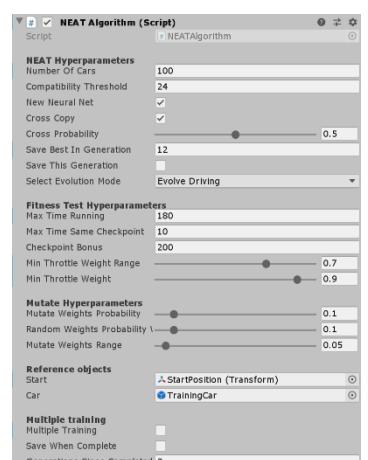
Podemos resumir el entrenamiento en las siguientes fases:



### 6.4.3 Hiperparámetros

Este algoritmo tiene gran cantidad de hiperparámetros y valores que podemos elegir libremente. Los más relevantes son los siguientes.

- Cada población tiene 100 individuos
- El límite de compatibilidad es de 24
- Se ha usado el método de Crossover personalizado
- El máximo de generaciones son 12
- El tiempo de entrenamiento máximo por generación son 180 segundos
- El tiempo máximo entre CheckPoints son 10 segundos
- El porcentaje mínimo del acelerador pulsado a fondo es del 90%
- La probabilidad de mutar un peso es del 10% para cada peso
- La probabilidad de que un peso mutado sea aleatorio es del 10%
- El rango de mutación de peso cuando no es aleatorio es del 5%



Selección de hiperparámetros en el entrenamiento en Unity

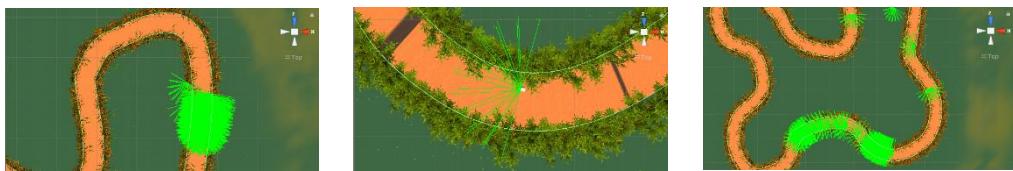
- El máximo de generaciones después de haber encontrado un individuo que complete el circuito es 3
- Las constantes  $c_1$ ,  $c_2$  y  $c_3$  de la distancia de compatibilidad son 1, 1 y 0.4 respectivamente.

Hay algunos más, pero estos son los más importantes.

#### 6.4.4 Resultados

Los resultados son bastante aceptables. Para ganarle una carrera a la IA en máxima dificultad, hay que cogerse alguno de los mejores coches del juego, en otro caso es imposible. A continuación, se mostrarán unas tablas con los tiempos que tardó cada IA en el entrenamiento en completar cada circuito.

Se ha entrenado cada uno de los 54 modelos en cada uno de los 4 circuito, por lo que se ha aplicado el algoritmo NEAT un total de 216 veces para obtener las IAs definitivas.



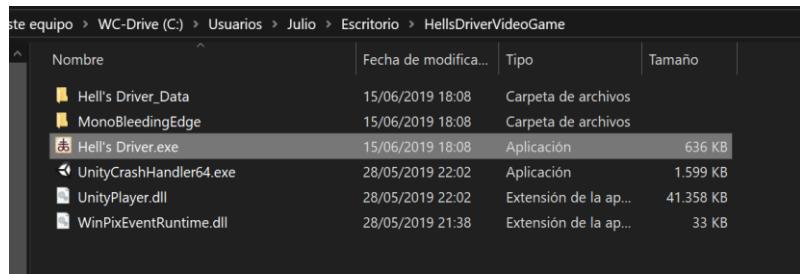
*Frames del entrenamiento usando NEAT, en el circuito Eight*

# 7 Jugando al videojuego

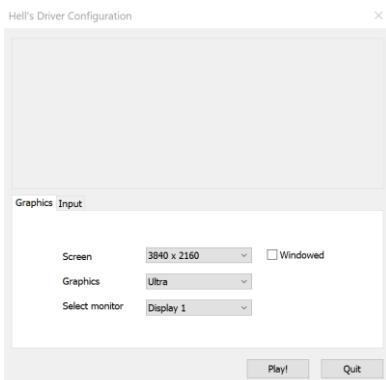
Para poder explicar correctamente cómo se ha hecho, primero debemos saber exactamente qué se ha hecho. Por tanto, antes de nada, vamos a dar un repaso al videojuego en sí, viendo cada una de las opciones, y enseñando cómo se juega. Una vez mostremos en qué consiste exactamente el videojuego, pasaremos a la parte donde se explica al detalle lo que hay detrás.

## 7.1 Iniciando el videojuego

La parte más evidente es abrir el propio videojuego. En la carpeta nos encontraremos varios archivos, abrimos el ejecutable.



Cuando lo abrimos, vemos la siguiente ventana:

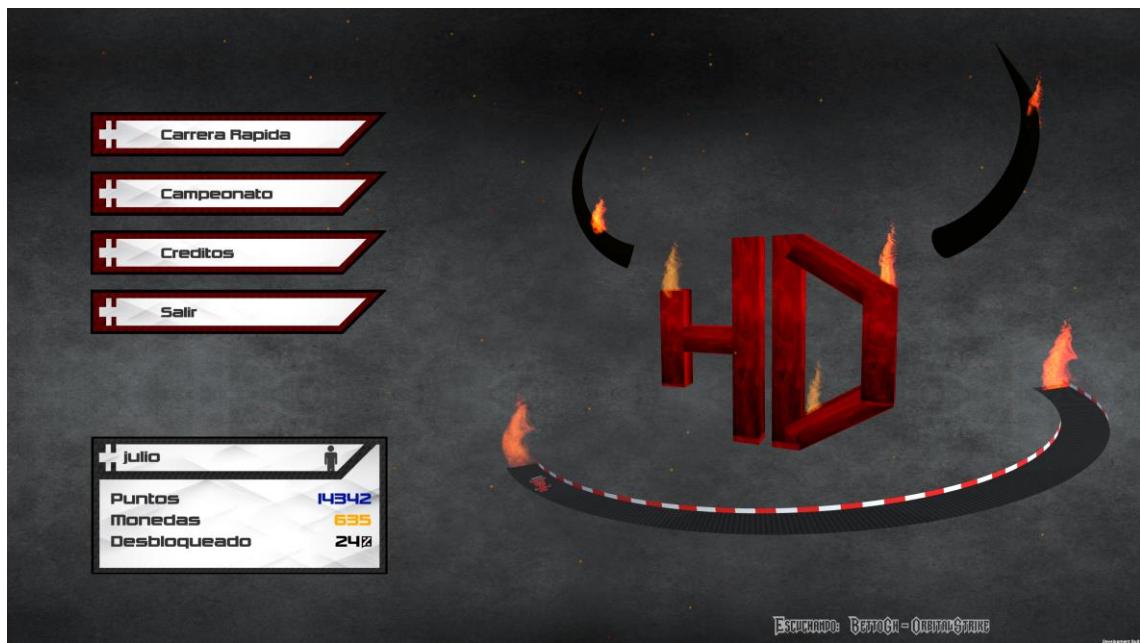


En esta ventana simplemente se elige qué resolución queremos para nuestro monitor. En el apartado “Graphics”, podemos elegir la calidad de texturas, pero en este videojuego todas tienen la misma calidad, por lo que es indiferente qué calidad elijamos en esta opción.

En la pestaña “Input” podemos modificar las teclas de entrada por defecto.

Le damos a jugar.

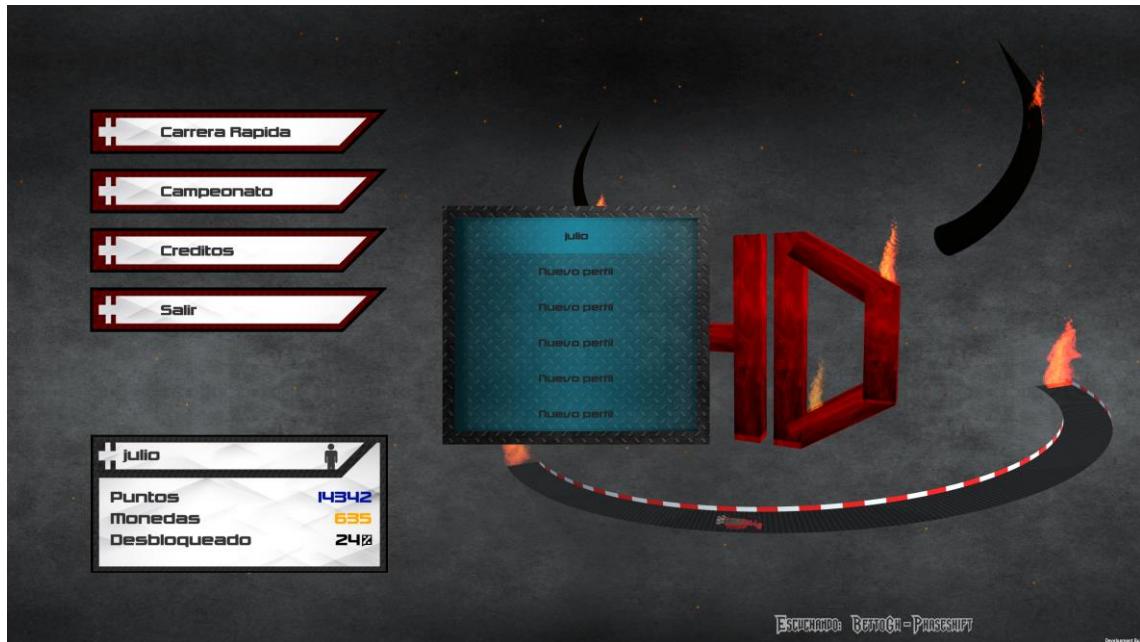
Lo primero que vemos es la escena de menú del videojuego. A la izquierda arriba tenemos cuatro botones, y a la izquierda abajo tenemos los datos de nuestro perfil. A la derecha vemos el logo del videojuego, y abajo a la derecha podemos leer el nombre de la canción que se está reproduciendo en ese momento.



## 7.2 Perfiles

Vamos a comenzar explicando la gestión de perfiles. Como ya se especificó en el análisis de requisitos, este videojuego tiene una gestión de perfiles, de forma que cada jugador puede tener su propio perfil, con sus propios puntos y monedas, y sus propios coches desbloqueados.

Para elegir un perfil, pulsamos en el nombre del perfil actual, en este caso, "Julio".



Como vemos, se nos abre una ventana con 6 slots. Podemos crear por tanto un máximo de 6 perfiles. En las siguientes imágenes, mostraré el proceso de crear otro perfil, y también de borrarlo.

Primero vamos a crear un perfil. Para ello, seleccionamos algún slot libre, y arriba veremos un cuadro donde escribir nuestro nuevo nickname. Una vez escrito, pulsamos enter. Como vemos en las dos últimas imágenes, el nuevo usuario se ha creado correctamente, con ningún punto ni monedas, y solo algunos coches desbloqueados. En la última imagen vemos que nos sale una

pequeña cruz al lado del perfil “Julio”. Esto sirve para eliminar el perfil. Dado que no podemos borrar nuestro propio perfil, al nuevo perfil no le sale la cruz.

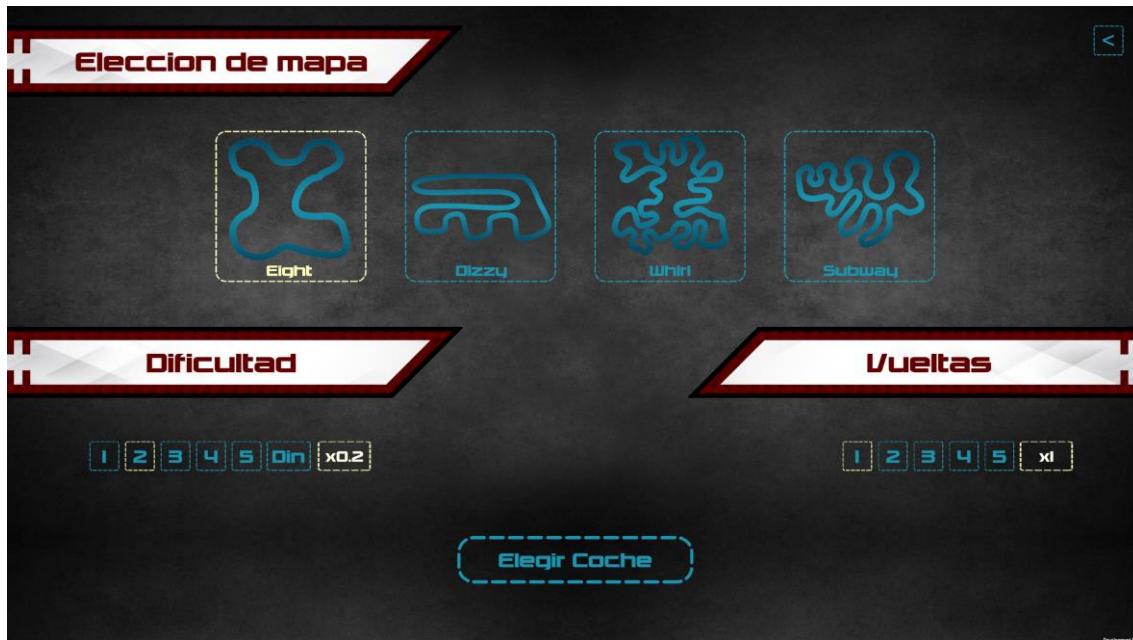
En el caso de que iniciemos el videojuego desde cero, sin ningún perfil, nos saldrá directamente el cuadro para crear uno.



### 7.3 Carrera Rápida

La carrera rápida consiste en una sola carrera, en el circuito que deseemos, con las vueltas y dificultad que deseemos, y con el modelo de coche que más nos guste (en este modo están todos desbloqueados).

Si pulsamos el botón de carrera rápida se nos abrirá la siguiente escena:

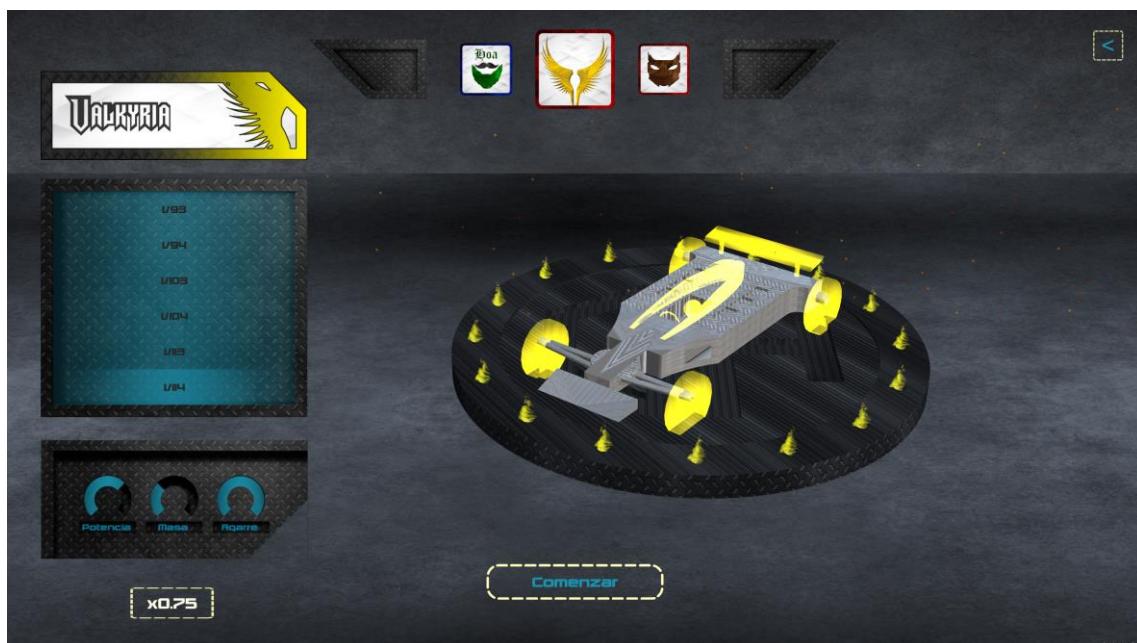


Como vemos, tenemos cuatro mapas a elegir, además del nivel de dificultad y número de vueltas. La dificultad se puede elegir entre 1 y 5, siendo 5 la más difícil. Vemos que a la derecha de la dificultad y el número de vueltas tenemos dos multiplicadores (x0.2 y x1 en este caso). Estos multiplicadores actúan sobre la puntuación final. El valor de los multiplicadores aumenta conforme se elige mayor nivel de dificultad o número de vueltas, lo que se traduce en una

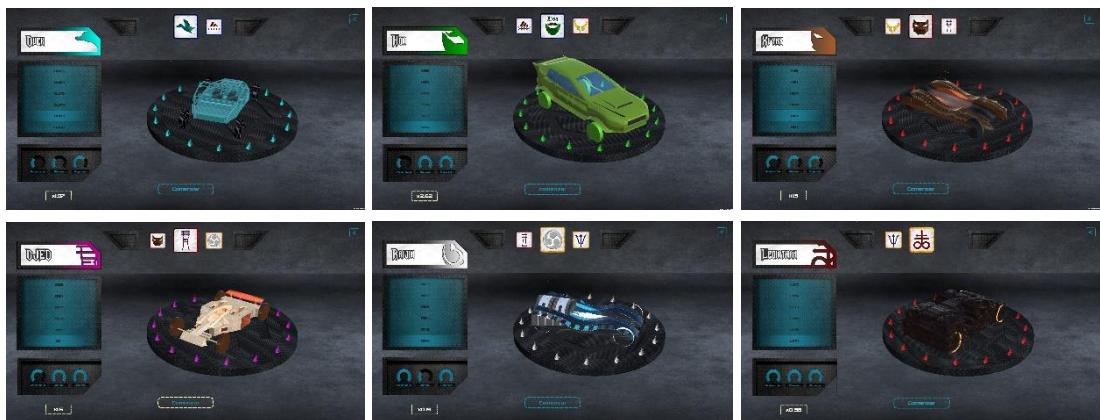
recompensa por jugar carreras difíciles o largas. Una vez elegida la dificultad y el número de vueltas, vamos a elegir coche.

Cuando pulsamos el botón de “Elegir Coche”, se nos abre una nueva escena. En esta nueva escena (imagen de abajo) vemos varias cosas. En la parte de arriba vemos en qué marca estamos actualmente. Podemos cambiar de marca con las flechas de izquierda y derecha, o con las letras A y D. Conforme vamos avanzando hacia la derecha, las marcas van teniendo coches más potentes. Las características de cada coche, marcas, etc. lo veremos en el capítulo de Diseño y Desarrollo.

Cada marca tiene seis modelos distintos, los cuales se pueden elegir en la izquierda. Cada uno de los modelos, tiene unas propiedades únicas: Potencia, masa y agarre. Además, tenemos otro multiplicador. Este multiplicador es mayor conforme peor características tiene el coche, de forma que el videojuego nos recompensa por jugar carreras con coches con peor características, y evita que siempre elijamos el mejor coche.



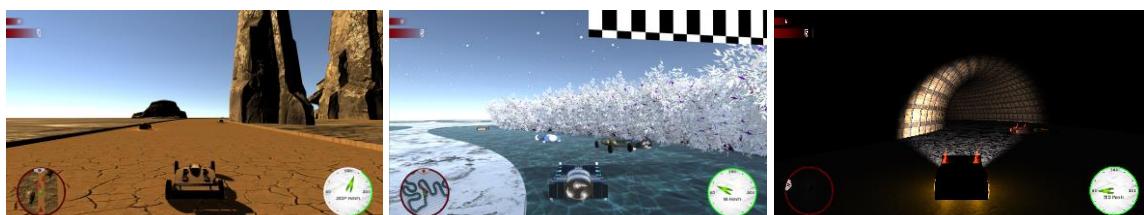
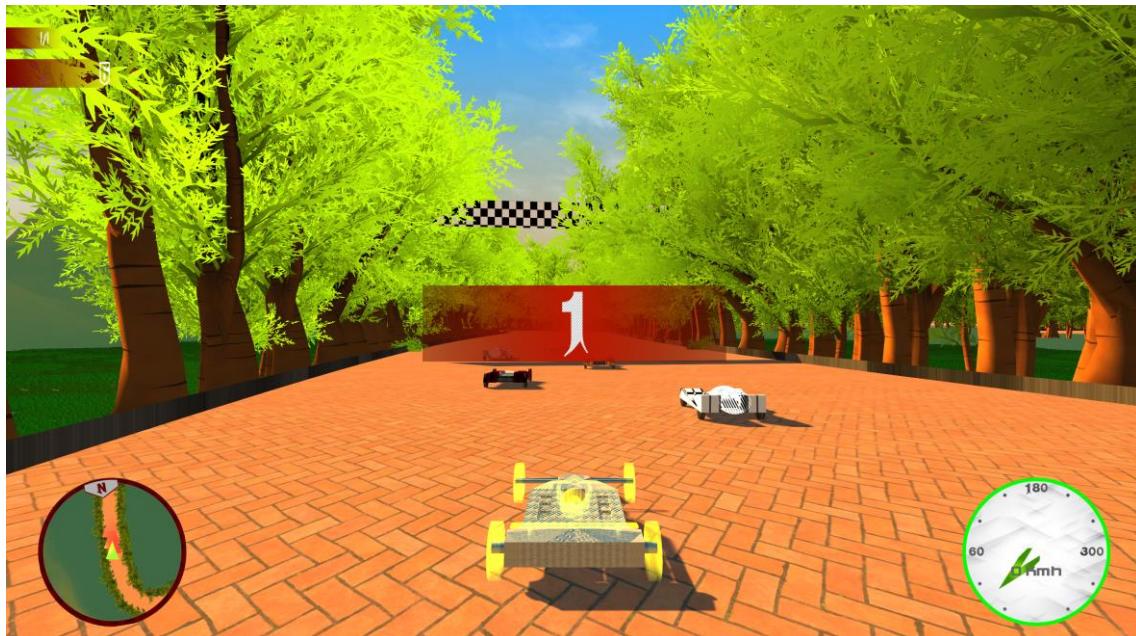
En total tenemos 9 marcas, lo que hacen un total de 54 modelos distintos.



Una vez hemos elegido marca y modelo, podemos darle a “Comenzar” y empezar a correr.

Una vez se nos abre la nueva escena, lo primero que vemos es una cuenta atrás. Cuando esta cuenta atrás llegue a cero, la carrera ha comenzado. En la imagen podemos ver varias cosas. Abajo a la derecha tenemos el velocímetro. La circunferencia verde que lo rodea es el nitro/boost, que al comenzar está completo. Si pulsamos el botón derecho lo usaremos y se irá gastando. Para evitar que se pulse de forma continua, si estamos usando el boost y soltamos, y está a menos de la mitad, no podemos volver a usarlo hasta que se vuelva a recargar a la mitad.

Abajo a la izquierda tenemos el minimapa, el cual nos muestra nuestra posición y la de nuestros rivales.



Una vez acabamos la carrera, nos aparece la tabla con los resultados:

**Carrera Finalizada**

Nombre	Modelo	Tiempo
1º Jose38	Poseidon P144	1 min 10 s
2º Justice	Poseidon P134	1 min 12 s
3º Julio	Valkyria V114	1 min 14 s
4º Kavinsky	Hlynx H114	1 min 22 s
5º Turbine64	Leviathan L144	1 min 34 s
6º SunsetNeon	Hlynx H104	1 min 35 s

**Calcular puntos**

En esta tabla podemos ver las posiciones, el modelo de coche usado, y el tiempo que cada coche ha hecho.

Vamos a calcular puntos:

**Carrera Finalizada**

<b>Puntos de carrera</b>	x1	<b>100</b>
<b>Segundos de ventaja</b>	x10	<b>596.2</b>
<b>Multiplicador de dificultad</b>	x0.5	<b>298.1</b>
<b>Multiplicador de vueltas</b>	x1	<b>298.1</b>
<b>Multiplicador de coche</b>	x0.75	<b>223.5?</b>
<b>Multiplicador de posición</b>	x1	<b>223.5?</b>
<b>Total puntos</b>		<b>224</b>
<b>Monedas</b>		<b>22</b>

**Ir al menu**

Como vemos, los multiplicadores hacen efecto en el total de puntos y de monedas.

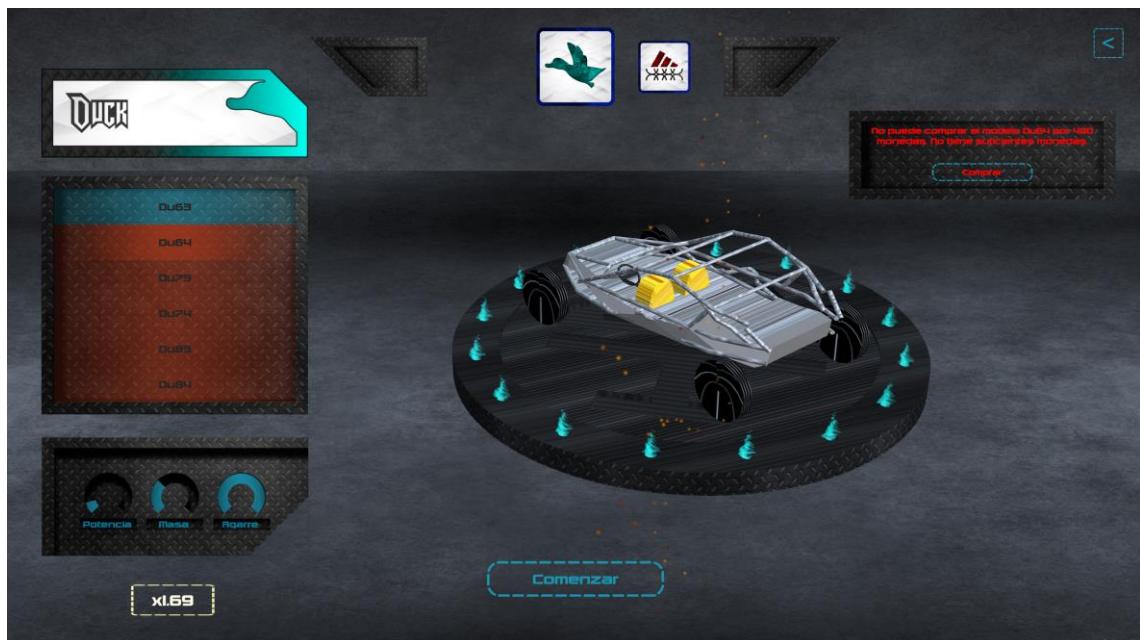
## 7.4 Campeonato

El modo campeonato consiste en cuatro carreras, una en cada mapa.

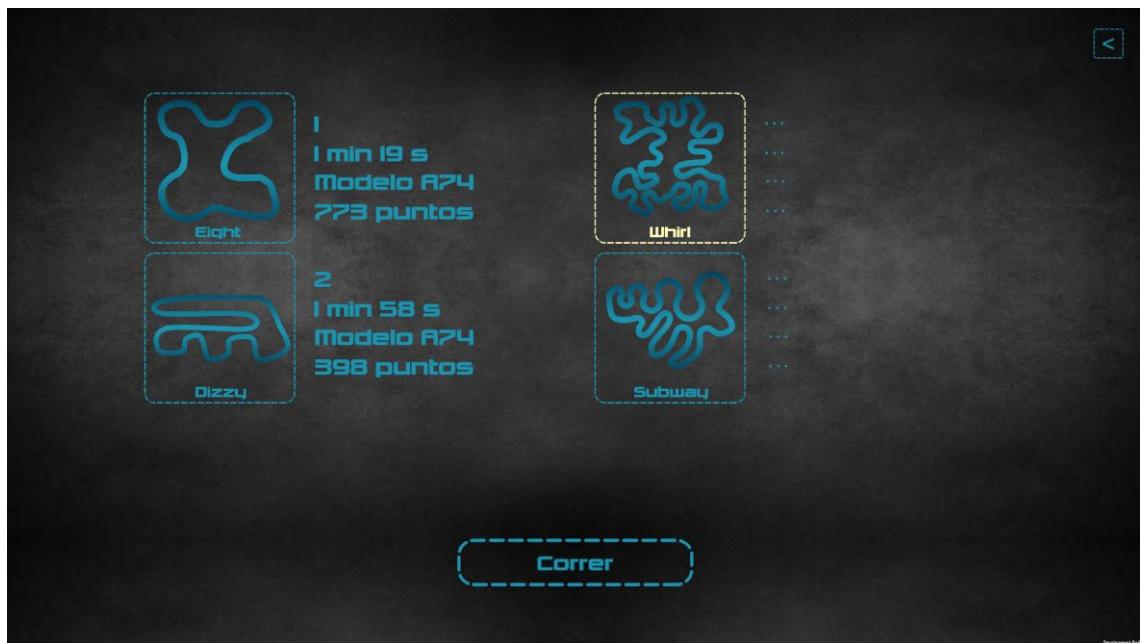
Como vemos en la imagen, podemos elegir la dificultad deseada para competir. A mayor dificultad, mejores son los premios.



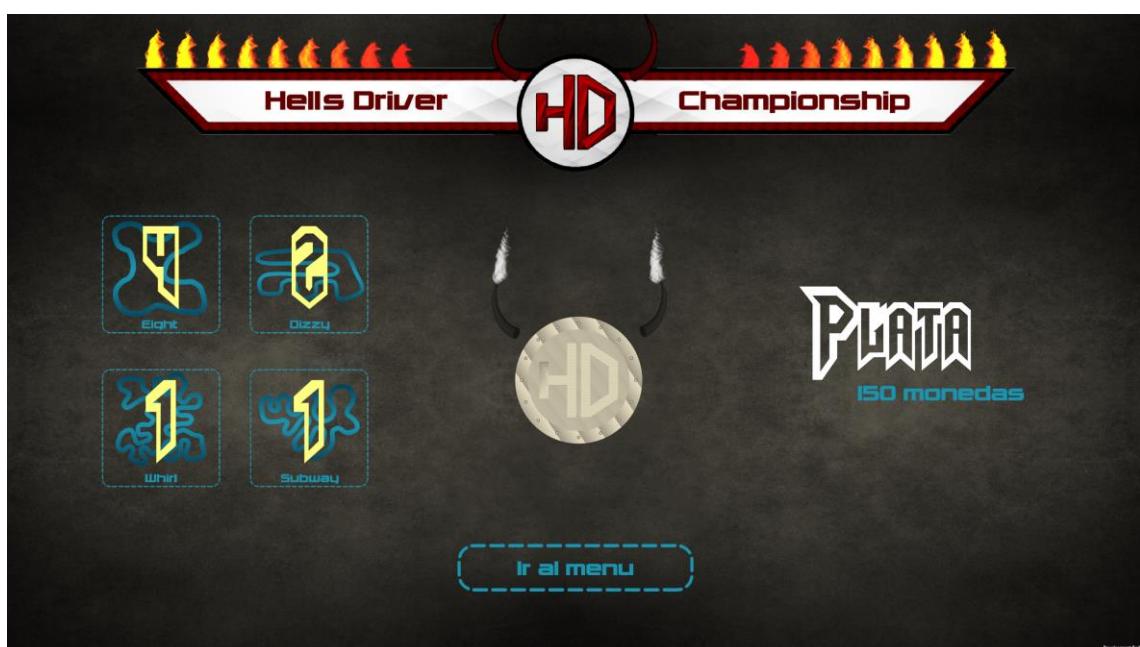
A la hora de correr en cada circuito, podemos elegir modelo de coche. Como vemos, en el modo campeonato tenemos la mayoría de coches bloqueados. Para desbloquearlos, hay que comprarlos como vemos a la derecha. En este caso no tenemos suficientes monedas, por lo que habrá que ahorrar.



Entre carrera y carrera, podemos ver los resultados en anteriores circuitos y en qué circuito tenemos que correr.



Una vez acabadas las cuatro carreras, finaliza el campeonato y se nos entregan los premios.



En cuanto a los modos de juego ya hemos visto todo. Hemos visto tanto el modo de carrera rápida como el modo campeonato. Habiendo enseñado en qué consiste realmente el videojuego, en el capítulo siguiente se procederá a explicar todo el proceso de desarrollo, desde los menús hasta la cómo se maneja el coche.

# 8 Conclusiones

Al principio del proyecto definimos un objetivo: Desarrollar un videojuego de coches cuya Inteligencia Artificial sea lo suficientemente buena para que ganar las carreras suponga un reto.

Dada mi visión subjetiva, puedo decir que ese objetivo se ha cumplido. Si queremos tener alguna posibilidad de ganar una carrera, deberemos coger alguno de los mejores coches. En cualquier otro caso, ganar se nos hará muy cuesta arriba.

A nivel visual el resultado me parece muy atractivo, tanto las interfaces como el entorno de los circuitos me parece que tienen un buen acabado. Si bien los modelados son un poco amateurs (no sabía nada de modelado hasta empezar con este proyecto), al final y gracias a la magia de las texturas, los coches tienen un diseño bastante decente.

A nivel de Inteligencia Artificial, no puedo exigirme más. Si bien siempre es mejorable, los mejores tiempos de la IA se acercan por mucho a los mejores tiempos hechos por mí, lo cual para mí personalmente es un logro.

También hay que decir que un punto a mejorar es el tema de las colisiones, pues en cada carrera siempre hay alguna colisión de vehículos, lo cual empeora un poco la experiencia. Pero resolver este punto es algo tan complejo, que escapa de mis posibilidades actualmente.

NEAT lleva un trabajo de implementación detrás enorme, pues es un método muy complejo y con muchas operaciones a tener en cuenta. Sin embargo, trabajar con algoritmos genéticos es algo que me apasiona y usarlo para entrenar redes neuronales ha sido una experiencia que he disfrutado mucho y también me ha dado muchos dolores de cabeza, pero a cambio he aprendido más que en muchas asignaturas.

En conclusión, yo personalmente diría que la valoración final es muy positiva, y el Trabajo de Fin de Grado aquí presentado es un proyecto sólido, con mucho trabajo detrás, y que cumple las expectativas que esperaba cumplir al solicitar este proyecto.



# 9 Bibliografía y referencias

- [1] [Online] <https://www.fib.upc.edu/retro-informatica/historia/videojocs.html>
- [2] [Online] [https://es.wikipedia.org/wiki/Desarrollo\\_de\\_videojuegos\\_independiente](https://es.wikipedia.org/wiki/Desarrollo_de_videojuegos_independiente)
- [3] Dalila García Notario, “Análisis de requisitos en el desarrollo de software”, Universidad Carlos III de Madrid, Madrid, España, 2015
- [4] [Online] [https://sites.google.com/site/myangelcafe/articles/history\\_ai](https://sites.google.com/site/myangelcafe/articles/history_ai)
- [5] [Online] <http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/>  
NEAT
- [6] O. K. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation, Vol. 10, No. 2, 99-127, 2002. [Online]  
<http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf>

*Referencias de material multimedia usado en el desarrollo del videojuego*

Imágenes y texturas

[Online] <http://pngtree.com>

[Online] <http://textures.com>

Fuentes

[Online] <http://en.fontreactor.com/foundries/justme54s/charlie-brown-m54>

[Online] <http://en.fontreactor.com/foundries/lokidesign/hel-grotesk-gothiq>

Sonido

[Online] <http://freesound.org>

[Online] <http://zapsplat.com>

[Online] <https://soundcloud.com/bettogh>