Universidad Zaragoza

Trabajo Fin de Grado

Grado en Ingeniería Informática

Multi Race Driving
Una plataforma general y personalizada para videojuegos de carreras en 2.5D

Autor

Rubén Rodríguez Esteban

Director

Eduardo Mena Nieto

ESCUELA DE INGENIERÍA Y ARQUITECTURA

Octubre 2020

AGRADECIMIENTOS

A mi director, Eduardo Mena, por su actitud a la hora de aportar ideas e innovaciones al trabajo, contribuyendo en todo momento a progresar y avanzar, y muy especialmente por su cooperación a la hora de probar como *beta tester* la plataforma y los distintos modos de juego que la componen.

A mi compañero de grado Daniel Naval Alcalá, por su colaboración como *beta tester*, por sus numerosas aportaciones para mejorar la calidad del *software* presentado, por su cooperación y gratificante disposición a la hora de llevar a cabo el despliegue y mantenimiento del servidor empleado en el modo multijugador, y por último, por su contribución en la programación y *testing* del *framework Linda*.

A mis compañeros de grado Víctor Miguel Peñasco Estívalez y Diego Royo Meneses, por estar siempre dispuestos a analizar y valorar aspectos de la plataforma y de los modos de juego en sus distintas versiones, aportando críticas objetivas, y en última instancia, constructivas, que han permitido que *Multi Race Driving* posea una gran calidad.

A mis amigos más cercanos, por dedicar su tiempo a responder cuestiones y dudas de estética y diseño, por poner a prueba la plataforma todas las veces que se les ha solicitado, y sobre todo, por la colaboración prestada a la hora de testear el modo multijugador.

A todos mis familiares, que son quiénes han permanecido en todo momento a mi lado animándome y dándome su apoyo para conseguir los objetivos.

Resumen

Actualmente, una de las áreas en las que más está prosperando la tecnología es el mundo de los videojuegos. Tal ha sido su creciente aumento que, hoy por hoy, el sector de los videojuegos factura más ingresos que la industria cinematográfica y musical juntas. La evolución que han experimentado desde sus orígenes, así como el incipiente aumento de la accesibilidad a los mismos por parte de una comunidad más amplía de usuarios por medio de plataformas digitales de distribución de videojuegos como *Steam*[2], *Origin*[3], o incluso repositorios *software* destinados a dispositivos móviles, como *Google Play Store*[4] o *App Store*[5], son las principales causas que han magnificado su papel en la sociedad contemporánea.

Este proyecto ha consistido en el diseño e implementación, sin ayuda de ningún motor gráfico preexistente, de una arquitectura general y personalizada para videojuegos de carreras en gráficos 2.5D[6], también denominados pseudo-3D. Estos términos describen gráficos en 2D que por ciertas técnicas (perspectiva, sombras, escalado) aparentan ser en 3D. En dicha plataforma se definen versiones totalmente parametrizables de algunos de los videojuegos más exitosos y populares de la década de los 80, como son *Pole Position* (1982), *Out Run* (1986), *Chase HQ* (1988) y *Super Hang On* (1989). Además, se ha definido el modo *Demarrage*, una submodalidad del juego *Need For Speed Undercover* (2008). Paralelamente, se ha diseñado un modo multijugador en red para que los jugadores puedan disfrutar de intensas experiencias de conducción jugando desde sus ordenadores de manera *online* contra otras personas.

[2] `https://store.steampowered.com/?l=spanish`
[3] `https://www.origin.com/esp/es-es/store`
[4] `https://play.google.com/store?hl=es-419`
[5] `https://www.apple.com/es/ios/app-store/`
[6] `https://programacionyjuegos.wordpress.com/2011/10/02/2d-3d-o-2-5d/`

Índice

Lista de Figuras

Capítulo 1

Introducción

Un videojuego es una aplicación *software* orientada al entretenimiento en el que una o más personas pueden interactuar, por medio de ciertos mandos o controladores, con un dispositivo que muestra imágenes de vídeo. Dicho dispositivo electrónico, más conocido comúnmente como plataforma puede ser una máquina recreativa, una videoconsola, un ordenador, o un dispositivo portátil como una *tablet* o un teléfono móvil.

En un principio, la idea del proyecto era el desarrollo de un videojuego, para posteriormente hacer una versión de él que fuese más configurable. Sin embargo, fue cambiada por desarrollar una arquitectura que permitiese ejecutar distintos modos de juego. Este enfoque fue el que finalmente se presentó como propuesta de este proyecto.

Para la realización de esta arquitectura se ha recurrido al uso de tecnologías y *software* de terceros. En el anexo A pueden consultarse con más detalle las tecnologías usadas en cuestión.

1.1. Motivación

Tal y como se mencionó anteriormente, *Multi Race Driving*[1] es una arquitectura para la definición de juegos de carreras en gráficos 2.5D. Posee varios modos de juego en los que los jugadores pueden poner a prueba su habilidad al volante. Además, cuenta con un modo multijugador *online* en el que los jugadores pueden medirse cara a cara con otros rivales desde sus propios ordenadores a través de la red.

[1] `https://github.com/ZgzInfinity/Multi-Race-Driving`

Inicialmente, solamente se había optado por implementar un único modo de juego que intentase recrear de la manera más realista posible el modo original (*Original mode*) del juego *Super Hang On* para la *Sega Megadrive* de 1989[2]. Durante la década de los 80, se desarrollaron una amplía variedad de videojuegos de conducción muy exitosos como *Pole Position*, *Out Run* y *Chase HQ*. Todos estos juegos, aunque son sumamente diferentes en cuánto a su temática, a nivel gráfico son prácticamente idénticos. La razón estriba en que estos videojuegos poseen gráficos en 2.5D. A continuación, se muestra un paisaje (véase figura 1.1) en el que puede observarse una carretera en perspectiva porque los objetos se ven más pequeños conforme están más lejos, pero en realidad son gráficos 2D (planos), en distintos tamaños para dar sensación de profundidad y perspectiva.

Figura 1.1: *Super Hang On* de *Sega Megadrive* (1989) con gráficos 2.5D.

Dada la similitud entre estos videojuegos, se decidió crear una arquitectura que permitiese definir versiones de todos los juegos anteriormente citados. El planteamiento que se decidió seguir fue implementar a priori una versión que recrease el modo original (*Original Mode*) de *Super Hang On*, y posteriormente, reutilizar la máxima cantidad de código posible para poder programar los modos de juego restantes.

[2] `https://www.youtube.com/watch?v=z5Jp9V5EXTg`

Aunque casi todos los juegos seleccionados son de los 80, en la actualidad, los videojuegos en 2D y 2.5D siguen siendo muy atractivos para el público, de ahí el interés por ellos. *Flappy Bird*[3] o *Rail Rush*[4] son ejemplos de algunos de los videojuegos en 2D y 2.5D, respectivamente, más populares en la actualidad.

Resultó interesante poder ofrecer para todos estos modos de juego, o al menos para alguno de ellos, algún aspecto novedoso que impactase al público, y muy especialmente, a los aficionados de los videojuegos originales. Por ello, se optó por la idea de desarrollar un modo multijugador en red para que los jugadores tuviesen tanto la opción de jugar en solitario como la de jugar en comunidad con otras personas. Puesto que todos los videojuegos tomados como referencia son solo para un único jugador, esta idea fue considerada oro para poder proporcionar al público esa originalidad tan deseada.

La programación de los modos de juego sin recurrir a ningún motor gráfico preexistente es una decisión que se tomó de forma paralela a la anterior. La razón estriba en que estos modos de juego están basados en videojuegos que fueron lanzados hace casi cuatro décadas. En aquellos años la Informática no estaba tan avanzada como en el presente. Por ello, herramientas como los motores gráficos estaban muy lejos todavía de existir. Esta decisión, aunque ha incrementado mucho la dificultad del trabajo, ha permitido viajar al pasado y hacer frente al mismo reto al que los desarrolladores originales de estos videojuegos se enfrentaron cuando los implementaron ya hace casi 40 años sin tener ninguna de las tecnologías actuales.

1.2. Objetivos

El objetivo del proyecto es el diseño completo de una plataforma totalmente personalizable de videojuegos de carreras en gráficos 2.5D. Se ha seguido un plan de fases que ha permitido pasar de unos primeros prototipos en los que todos las características (menús, vehículos, paisajes) estaban programadas y para efectuar los cambios era necesario recompilar, a una arquitectura donde todos estos aspectos están definidos en ficheros *XML* externos de configuración que pueden modificarse sin tener que tocar la implementación. La planificación de fases ha sido la siguiente:

[3] `https://www.youtube.com/watch?v=B7K4bCYV_tY`
[4] `https://www.youtube.com/watch?v=KcvpPoDQAqA`

- Diseño de un primer mapa programado en código con todas sus características: anchura de la carretera, número de carriles, fondo del mapa, colores, trazado del circuito (rectas, curvas y cambios de rasante) y los elementos decorativos del mapa.
- Diseño e implementación de la lógica de control de un primer vehículo.
- Desarrollo de un primer modo de juego con todos sus elementos: escenarios, sistemas de puntuación y récords, interfaz del juego, modos de dificultad, etcétera.
- Diseño de varios vehículos jugables de diversos colores con características diferenciadas y configurables en ficheros en formato *XML*.
- Programación de un sistema de procesado y generación de escenarios personalizables que permitiese poder representarlos en ficheros con formato *XML*.
- Diseño de una *GUI* (*Guide User Interface*)[5] simple, minimalista y preatentiva por medio de un sistema de lectura y generación de menús a partir de ficheros totalmente configurables en formato *XML*.
- Permitir visualizar el *software* en distintas resoluciones de pantalla y con distinta calidad de gráficos (efecto *pixel art*[6]).
- Implementación del resto de modos de juego tomando como base el primer modo, incorporando en cada uno de ellos sus rasgos más singulares.
- Gestión e incorporación de las bandas musicales y el *SFX* (efectos de sonido).
- Programación de varios tipos de *IA* (inteligencia artificial).
- Diseño del modo multijugador *online*.
- Realización de *tests* y pruebas para comprobar el comportamiento correcto del *software* desarrollado.

Para tener más información sobre la planificación y control de esfuerzos del proyecto se puede consultar la sección 6.1 del informe.

[5] `https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario`
[6] `https://es.wikipedia.org/wiki/Pixel_art`

Capítulo 2

Descripción de *Multi Race Driving*

Este capítulo explica las propiedades más importantes de la arquitectura diseñada, con la finalidad, de poder contextualizar de un modo más genérico cómo se ha enfocado y abordado su elaboración.

2.1. Características principales

La programación de *Multi Race Driving* ha estado marcada por los criterios que seguidamente se detallan:

- Realismo y velocidad: punto muy crítico durante todo el desarrollo de la arquitectura debido a que no se ha usado ningún tipo de motor gráfico preexistente. Por ello, todo el tema referente a las leyes físicas, más concretamente, a la dinámica de las fuerzas y la cinemática para poder modelar la inercia de los bólidos en las curvas, las colisiones tanto con objetos de los mapas como con otros vehículos o para modelar las sensaciones de velocidad, aceleración y frenado de los vehículos, respectivamente, han tenido que ser programadas desde cero.

- Estética y apariencia: se ha tenido que tratar con mucho cuidado la apariencia de los menús que componen la *GUI* (colores, texto, audio, fuentes, imágenes de fondo). Igualmente, los gráficos de los escenarios han sido creados con el fin de lograr mundos realistas, visuales y con una personalidad diferente con el fin tanto de impresionar al jugador como de dotar a cada modo de juego de una singularidad única, algo que no se consiguió a la primera.

- Parametrización: la arquitectura se ha programado de tal manera que pueda ser completamente configurable. Usando ficheros en formato *XML*, aspectos como los menús que componen la interfaz, los escenarios de los distintos modos de juego,

o incluso los vehículos, pueden ser modificados fácilmente por los usuarios. De este modo, cada jugador puede personalizar *Multi Race Driving* a su gusto.

– Jugabilidad: para ser el mejor jugador no basta con quedar el primero o lograr el objetivo de la misión, además hay que hacerlo consumiendo el menor tiempo posible. Este factor, y en algún que otro modo de juego también los puntos obtenidos, son los que determinan aquellos jugadores que se colocan en la cúspide de los récords a batir. *Multi Race Driving* es un *software* altamente desafiante ya que cuenta con varios niveles de dificultad con el objetivo de ser un reto para cualquier tipo de jugador, desde simples *amateurs* hasta jugadores expertos. Los trazados de los escenarios, el tiempo de juego disponible, los vehículos rivales y el tráfico, el objetivo de cada modo de juego, y en última instancia, el modo multijugador *online*, garantizan partidas emocionantes, diferentes e intrigantes, independientemente del tiempo que se lleve ya jugado.

– Legalidad: todos los *sprites* empleados para la elaboración de los gráficos, así como las bandas musicales y el *SFX* constituyen un material que no ha sido creado por el desarrollador del *software* sino que ha sido recopilado de otros videojuegos muy populares. Por ello, se ha asegurado el reconocimiento de los derechos de autor a los artistas y corporativas originales referenciándolas con el fin de asegurar la legalidad y evitar cometer delitos contra la propiedad intelectual.

2.2. Público objetivo

La plataforma propuesta ha sido pensada para ser dirigida a todos los públicos, y muy especialmente, a los jugadores amantes de los juegos de conducción *arcade* (juego fácil de manejar sin llegar al realismo propio de las simulaciones de conducción). *Multi Race Driving* tiene definidos varios modos de juego que mezclan habilidad, rivalidad, coordinación, velocidad y diversión. Tales elementos hacen que cualquier persona, independientemente de la edad y del sexo, pueda verse con el deseo de querer jugar y pasarlo bien. Sin embargo, se recomienda que no jueguen personas menores de siete años, tal y como indica el *PEGI*[1] (*Pan European Game Information*). La razón estriba en que las modalidades de juego pueden resultar complejas de entender y controlar para usuarios tan pequeños.

[1] `https://pegi.info/es`

Capítulo 3

Diseño de *Multi Race Driving*

El diseño de *Multi Race Driving* ha estado enfocado a que la interacción con los jugadores sea lo más amena posible. A continuación, se matizan los aspectos más importantes.

3.1. Arquitectura

Multi Race Driving es una plataforma integrada por varios módulos que se comunican y sincronizan entre sí (véase figura 3.1).

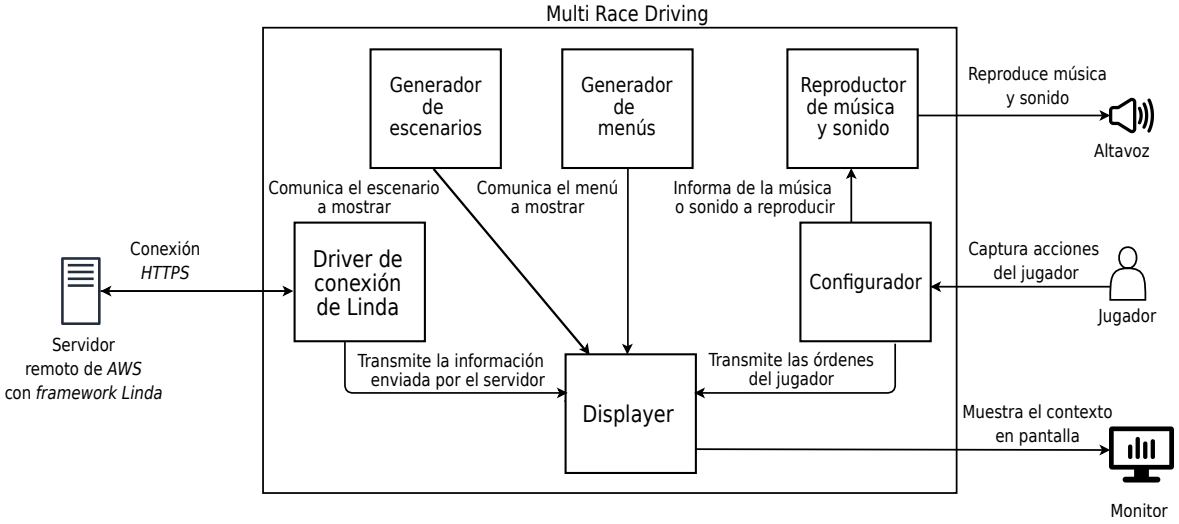


Figura 3.1: Arquitectura de *Multi Race Driving*

A continuación, se describen más en profundidad cuáles son las tareas que realizan cada uno de los módulos presentes en el diagrama anterior.

– Configurador: es el módulo más sustancial de *Multi Race Driving* puesto que es el responsable de cargar, guardar y modificar los parámetros de

configuración (dificultad, controladores, volumen, resolución, calidad de gráficos) de la plataforma. En el anexo B se explica cómo configurarlo.

- Generador de escenarios: procesa la lectura de los mapas de los distintos modos de juego, cuyos contenidos están almacenados en ficheros *XML*.
- Generador de menús: genera los diversos menús, configurados en ficheros *XML*, que componen la *GUI*.
- Reproductor de música y sonido: carga y reproduce todas las bandas sonoras y el *SFX* empleados por la plataforma.
- Driver de conexión: permite la comunicación de la arquitectura (cliente), con el servidor multijugador, alojado en *AWS*.[1]
- Displayer: muestra al jugador en pantalla toda la acción que tiene lugar mientras el *software* se está ejecutando.

3.2. Modos de juego de un solo jugador

Mientras que algunos modos de juego definidos en *Multi Race Driving* pueden jugarse tanto en solitario como en modo multijugador con ciertas variaciones, otros solamente están disponibles para un solo jugador. La razón estriba en que la perspectiva del modo multijugador no concuerda con el objetivo de dichos modos de juego. A continuación, se explican los modos disponibles para un solo jugador.

3.2.1. *World Tour*

Es un modo de juego (véase figura 3.2a) en el que el jugador se enfrenta a otros siete rivales en carreras por todo el mundo esquivando al tráfico circundante. Para poder promocionar a la siguiente carrera se debe llegar a la meta antes de que se acabe el tiempo de juego y en primera posición. A lo largo del trayecto hay distintos puntos de control que el jugador debe ir cruzando para incrementar el tiempo de juego. Si se agota el tiempo de juego el jugador es descalificado. Esta modalidad se ha inspirado en el videojuego *Super Hang On* para la *Sega Megadrive* (*Sega* 1989)[2].

[1] `https://aws.amazon.com/es/what-is-aws/`
[2] `https://www.arcade-museum.com/game_detail.php?game_id=9901`

3.2.2. *Out Run*

Es una modalidad de juego (véase figura 3.2b) en la que el jugador debe recorrer cinco escenarios y llegar a la meta antes de que se acabe el tiempo de juego y con la mayor puntuación posible. Al final de los cuatro primeros escenarios hay una bifurcación por la que el jugador puede optar ir a izquierda o a derecha. La dirección que tome determina el siguiente escenario por el que correr. Cada vez que se pasa por una bifurcación se incrementa el tiempo de juego. Si se agota el tiempo de juego, el jugador es eliminado y se pierde la partida. Cuanto más rápido se circula, más puntos se consiguen. Este modo se ha basado en el videojuego *Out Run* para la recreativa (*Sega* 1986)[3].

(a) Modo *World Tour*

(b) Modo *Out Run*

3.2.3. *Pole Position*

En este modo de juego (véase figura 3.3a) el jugador debe competir en solitario o contra diversos rivales dando vueltas a un circuito e intentar llegar a meta en primera posición antes de agotarse el tiempo de juego. En esta modalidad de juego, tanto el circuito donde competir, como el número de vueltas que dar y los rivales contra los que correr son seleccionados previamente por el jugador. En el circuito hay distintos puntos de control que el jugador debe ir cruzando para incrementar el tiempo de juego. Si llega a la meta pero no en primera posición, se le solicita si desea repetir la carrera. Si se agota el tiempo de juego el jugador es descalificado. Esta modalidad se ha inspirado en el modo de juego *Pole Position 2* para *Atari* (*Namco* 1982)[4].

[3] `https://www.arcade-museum.com/game_detail.php?game_id=8938`
[4] `https://www.arcade-museum.com/game_detail.php?game_id=9063`

3.2.4. *Driving Fury*

En esta modalidad de juego (véase figura 3.3b) el jugador dispone de un tiempo de juego para perseguir a un vehículo criminal y colisionar contra él, hasta que el nivel de salud baje a cero para arrestarlo, antes de completar cinco escenarios. Al final de los cuatro primeros escenarios del recorrido hay una bifurcación por la que el jugador puede optar ir libremente a izquierda o a derecha. La dirección que tome el jugador en la bifurcación determina el siguiente escenario por el que correr. Al pasar por cada bifurcación se incrementa el tiempo de juego. En caso de arrestar al criminal (bajarle el nivel de salud a cero), el jugador promociona al siguiente nivel y debe perseguir a un criminal distinto. Si se agota el tiempo de juego el jugador pierde la partida. Como referencia para este modo de juego se ha tomado el videojuego *Chase HQ* para la máquina recreativa (*Taito* 1988)[5].

(a) Modo *Pole Position*

(b) Modo *Driving Fury*

3.2.5. *Demarrage*

En esta modalidad el jugador dispone de un tiempo de juego para perseguir a un vehículo rival, adelantarlo, y ponerse por delante de él una distancia de 400 metros, antes de completar cinco escenarios. Al final de los cuatro primeros escenarios del recorrido hay una bifurcación por la que el jugador puede optar ir libremente a izquierda o a derecha. La dirección que tome el jugador en la bifurcación determina el siguiente escenario por el que correr. Al pasar por cada bifurcación se incrementa el tiempo de juego. Si se logra el objetivo, el jugador promociona al siguiente nivel y se enfrenta a un rival distinto. Si se agota el tiempo de juego el jugador pierde la partida.

[5] `https://www.arcade-museum.com/game_detail.php?game_id=7318`

Esta modalidad de juego se ha inspirado en el submodo *Demarrage* del videojuego *Need For Speed Undercover* para *PS2* (*EA Black Box* 2008)[6].

Aspectos técnicos de estos modos de juego como los modos de dificultad, los distintos niveles de juego, las adversidades a las que se debe hacer frente o los sistemas de clasificación y puntuación pueden encontrarse en el anexo C.

3.3. Gráficos

El tema de los gráficos ha sido un factor muy peliagudo desde el comienzo del proyecto ya que parte de ellos están basados en una combinación de estilos, que se deriva de los distintos videojuegos en los que se ha inspirado *Multi Race Driving*. En definitiva, los gráficos se han diseñado mezclando ciertos aspectos de los videojuegos originales con la imaginación del desarrollador, logrando así, una apariencia y expresividad considerablemente diferente. A continuación, se explican los diversos tipos de gráficos diseñados (interfaz, escenarios, etcétera), sus características, y la metodología empleada para conseguirlos.

3.3.1. Interfaz

La interfaz de *Multi Race Driving* cuenta con una gran diversidad de menús (véase figura 3.4) por los que el jugador puede navegar y realizar diversas acciones. La estética y apariencia de los menús que componen la *GUI* del *software* presenta un estilo muy similar al empleado en un *remake* implementado en *C++* del videojuego *Out Run*[7], un proyecto *open source* con licencia *GPL-3.0*. Cada menú tiene un fichero *XML* donde aspectos como el contenido de los textos, la fuente usada, las imágenes de fondo o incluso los colores pueden ser modificados libremente por los jugadores.

La interfaz de *Multi Race Driving* está compuesta por menús, animaciones y créditos. Cada menú de la arquitectura se caracteriza por tener un fichero de configuración en formato *XML* que el jugador puede leer y modificar si lo desea con el fin de poder realizar los cambios que él quiera. De esta manera se permite que toda la apariencia

[6] `https://www.ea.com/es-es/games/need-for-speed/need-for-speed-undercover`
[7] `https://github.com/ZgzInfinity/OutRun`

(colores, tipos de fuentes, texto, imágenes de fondo, etcétera) puedan ser cambiadas y ajustadas a gusto de los usuarios. Las animaciones, en cambio, no son parametrizables como ocurre con los menús debido a que muchas de ellas precisan de movimiento de elementos en pantalla, es decir, se necesita que todos los elementos que intervienen en ellas estén perfectamente coordinados para conseguir que las animaciones estén cuadradas y sean estéticas. Si fuesen configurables este aspecto sería muy difícil de conseguir. La interfaz de *Multi Race Driving* cuenta con un total de 18 menús, ocho animaciones y 10 créditos. Con el objetivo de poder facilitar el uso de *Multi Race Driving* a los jugadores, se ha elaborada un manual de usuario[8], presente en el anexo D.

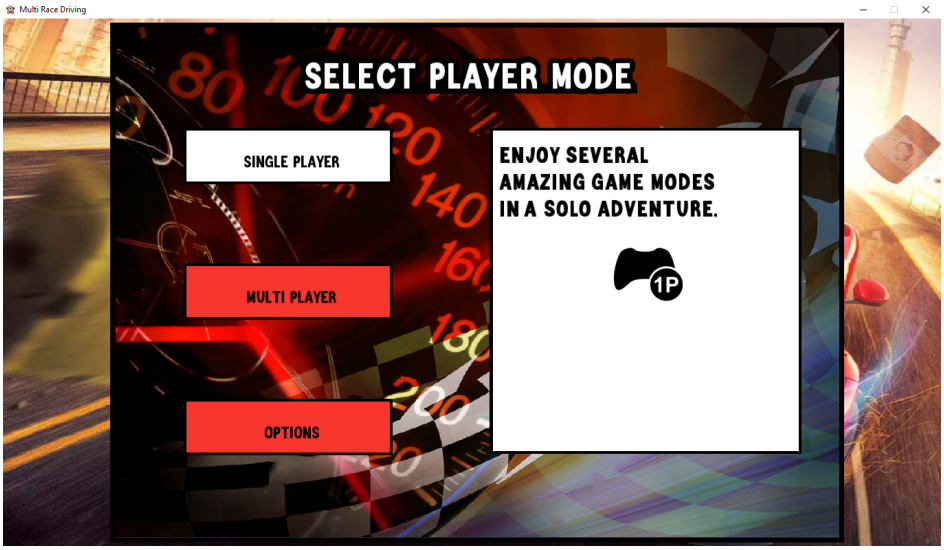


Figura 3.4: Ejemplo de menú de la *GUI*

Con el objetivo de garantizar una generación de menús rápida y eficiente, la lectura de los ficheros *XML* de los menús solo es efectuada una vez, es decir, el generador procesa los ficheros y los genera conforme el jugador va navegando por la *GUI*, guardando la configuración en memoria. De este modo, no es necesario leer el fichero de un menú cada vez que se accede a él porque sería mucho más ineficiente. Así las cosas, si el jugador hace cambios en un fichero *XML*, podrán visualizarse si y solo si todavía no se ha navegado por ese menú. En caso contrario, *Multi Race Driving* habrá leído y guardado en memoria la configuración anterior y deberá ser reiniciado para hacer efectivos los cambios. En el anexo E se explican con más detalle las propiedades de estos ficheros y cómo editarlos.

Otro aspecto importante en el diseño de la interfaz ha sido la creación de las *HUD's* de los distintos modos de juego. Estas *HUD's* dependen del modo de juego. Son

[8] `https://es.wikipedia.org/wiki/Gu%C3%ADa_del_usuario`

minimalistas, concisas para maximizar la visibilidad de juego, y significativas, ya que aportan la información necesaria para que el jugador juegue cómodamente.

3.3.2. *Sprites*

En consonancia con el estilo propio de los videojuegos de las décadas de los 80 y 90, la arquitectura se ha diseñado usando gráficos 2D, planos, a través de *sprites*[9] muy variados (véase figura 3.5). Los *sprites* son un conjunto de imágenes que representa un personaje u objeto (o una parte de ellos) de manera gráfica y que se utiliza para poder crear cualquier efecto de movimiento o para cambiar su estado o posición en la escena.

Para el diseño de *Multi Race Driving* se han usado, entre vehículos y escenarios, un total de 4527 *sprites*. Para poder representar los distintos tipos de vehículos disponibles en *Multi Race Driving*, se han usado un total de 2744 *sprites* (aproximadamente 58 *sprites* por vehículo). Para los escenarios, entre fondos y elementos decorativos (algunos de estos últimos repetidos en varios escenarios), se han usado un total de 1783 *sprites* (alrededor de 33 *sprites* por escenario).

Figura 3.5: Ejemplos de *sprites* de *Multi Race Driving*

En la sección F.2 del informe se pueden consultar más detalles acerca de las características y procedencia de los *sprites*.

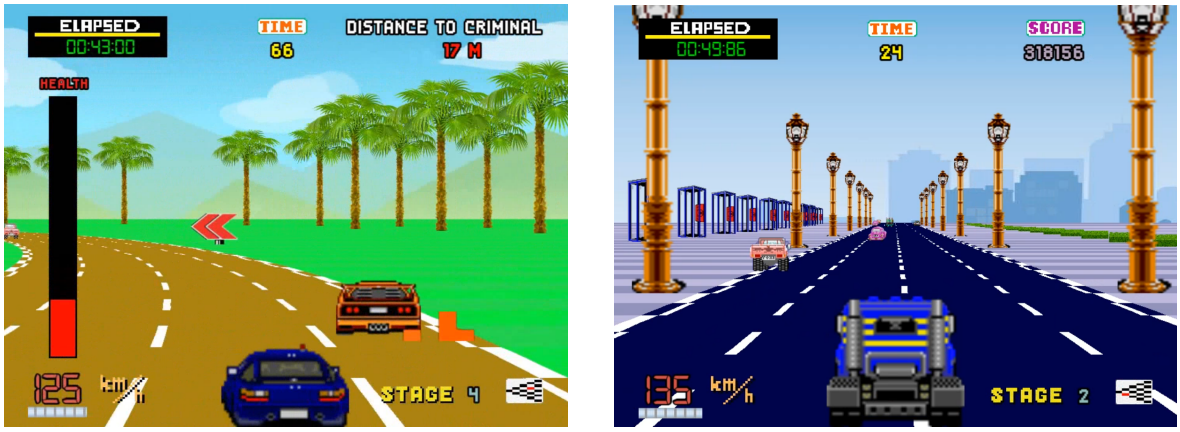
[9] `https://www.tokioschool.com/noticias/sprite-videojuegos/`

3.3.3. Escenarios

Los escenarios son una parte muy importante de los gráficos de *Multi Race Driving* ya que se encargan de transmitir realismo a los jugadores, y además, de dar personalidad a los distintos modos de juego. A continuación, se describen los distintos tipos de escenarios diseñados y cuál ha sido la metodología aplicada para conseguirlos.

3.3.3.1. Escenarios de nivel

Los escenarios de nivel son los mapas que representan las fases en las que se dividen los modos de juego de la arquitectura. Dichos escenarios han sido muy trabajados con el fin de transmitir expresividad y variabilidad visual a los jugadores. Se ha procurado que todos los escenarios tengan una estética y personalidad únicas que los diferencien del resto, y además, que su apariencia sea acorde al espíritu de la modalidad de juego a la que pertenecen (véase figura 3.6). Para los modos de juego *World Tour* y *Pole Position* se han diseñado cuatro escenarios. Para los modos *Out Run*, *Driving Fury* y *Demarrage* se han diseñado, para cada uno, 15 escenarios. Como el modo multijugador extiende los modos de juego *World Tour* y *Pole Position*, se han reutilizado sus propios escenarios.



(a) Escenario del modo *Driving Fury* (b) Escenario del modo *Out Run*

Figura 3.6: Ejemplos de escenarios de nivel

3.3.3.2. Generador de escenarios de nivel

La construcción de los escenarios de nivel ha sido laboriosa. Para crearlos se ha diseñado una técnica de generación automática a través de ficheros de configuración en formato *XML*. Estos ficheros especifican los colores del mapa, el trazado de la

carretera, las curvas, los cambios de rasante (subidas y bajadas de montañas) y todos los obstáculos y elementos decorativos del paisaje. El generador se encarga de procesar los ficheros y guardar el contenido en memoria. La técnica de generación de escenarios se caracteriza por ser sencilla y flexible. De este modo, se posibilita que cualquier usuario pueda modificar los mapas de los distintos modos de juego manipulando exclusivamente los ficheros *XML*. Para procesar los ficheros *XML* se ha diseñado una versión del patrón *Master-Worker*[10] (véase figura 3.7). En base al modo de juego, el proceso *master* (hilo principal) crea un proceso *worker* por cada escenario a procesar, asegurando así, un equitativo reparto de la carga. El orden de guardado de los mapas leídos es el único aspecto de sincronización a tener en cuenta. Cuando un *worker* termina de procesar su correspondiente fichero, permanece a la espera de que el *master* guarde su mapa leído cuando sea su turno. En la sección F.4.1 se explica con precisión como configurar este tipo de escenarios.

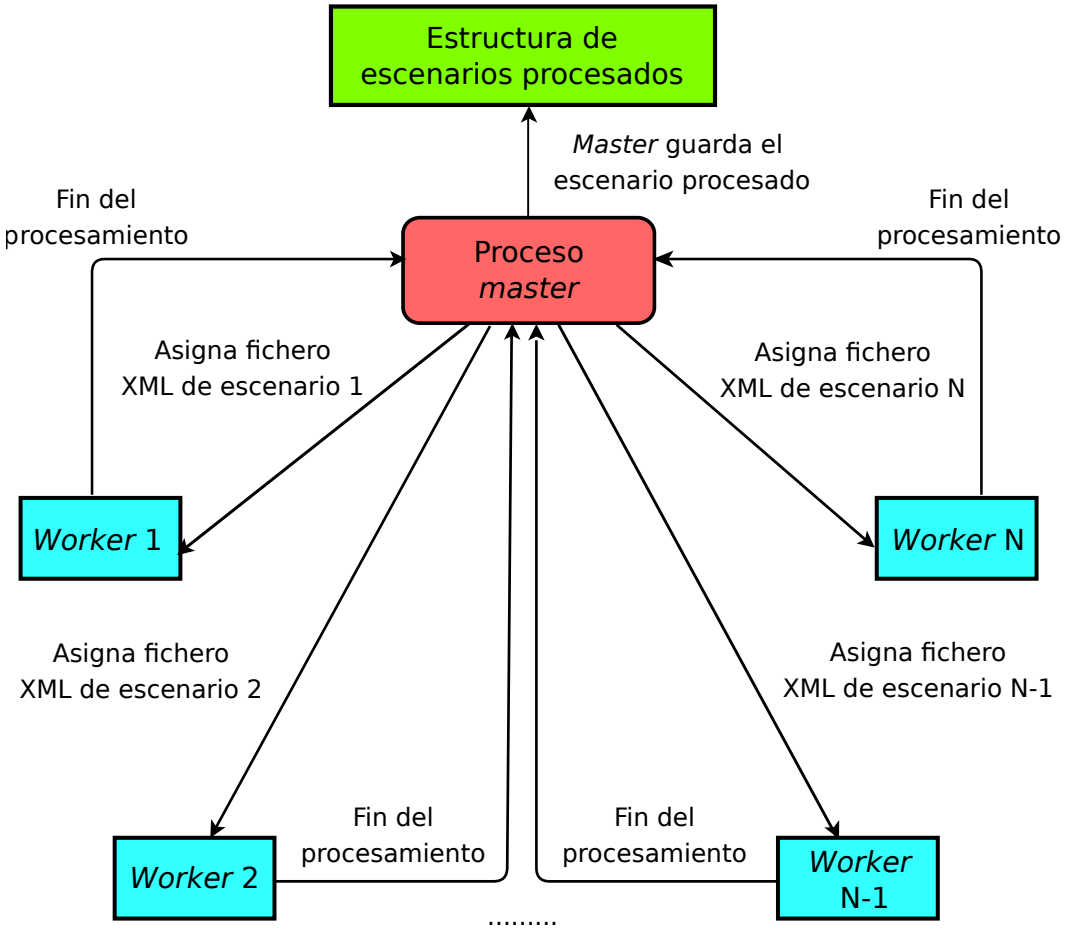
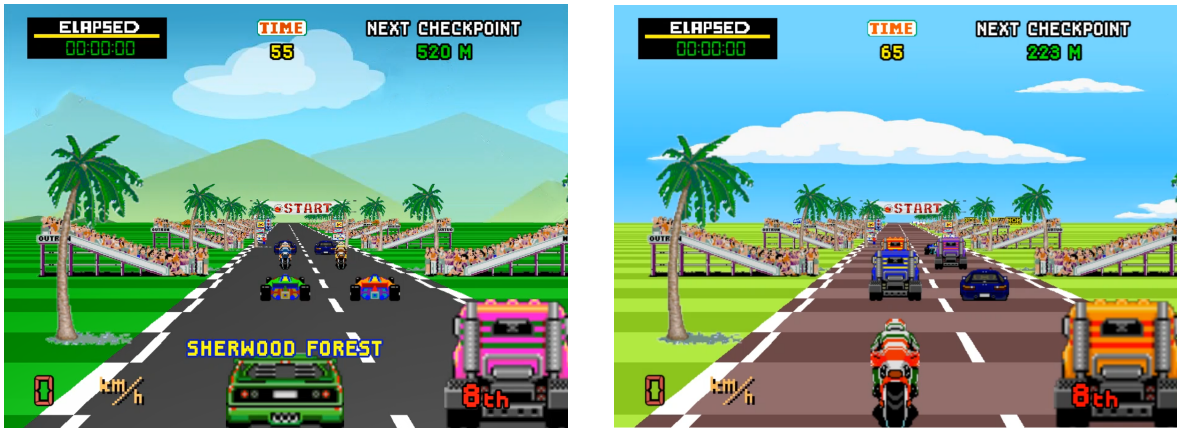


Figura 3.7: Patrón *Master-Worker* para el procesamiento de escenarios de nivel

[10] `http://web.cs.iastate.edu/~panini/man/indexch9.html`

3.3.3.3. Escenarios de salida, vuelta intermedia y llegada

Además de los escenarios de nivel, también se han tenido que crear escenarios específicos para representar las salidas en las carreras, las llegadas a meta y las vueltas intermedias en caso de jugar al modo *Pole Position* (véase figura 3.8). Estos escenarios no se han podido generar con ficheros *XML*, sino que han tenido que ser programados en el código debido a que durante las etapas de salida, llegada y vuelta intermedia tienen lugar diversas animaciones (abanderado, semáforos, indicador de vuelta, etcétera) que resultaron imposibles de poder representar en ficheros. Para más detalles sobre este tipo de mapas se puede consultar la sección F.4.2 del informe.



(a) Primer nivel del modo *World Tour* (b) Primer circuito del modo *Pole Position*

Figura 3.8: Ejemplos de escenarios de salida

3.3.3.4. Bifurcaciones

Este tipo de escenario también se ha programado en código. De todos los escenarios diseñados, las bifurcaciones son en las que más tiempo se ha invertido debido a la complejidad de las matemáticas usadas. Las bifurcaciones se han diseñado tomando dos circunferencias con centros y radios precalculados con el fin de que la bifurcación resultante (véase 3.9) no estuviese en ningún punto deformada, fallo muy recurrente durante la implementación.

El trazado de la bifurcación se construye sobre dos circunferencias de centros y radios precalculados (véase figura 3.10). Sobre las circunferencias verde y azul se dibuja la bifurcación. Existen dos puntos de corte, el primero conecta la primera circunferencia con la segunda, y el segundo conecta la segunda circunferencia con la carretera recta hasta comenzar el siguiente escenario. El uso de centros y radios precalculados permite

que el primer punto de corte entre las dos circunferencias coincida con los primeros 45° recorridos de la primera circunferencia desde el origen, y el segundo punto de corte coincida con los primeros 45° recorridos de la segunda circunferencia desde el primer punto de corte. Así, se logra que no haya una curva imposible y se restablece la trayectoria de la carretera hacia la misma dirección desde la que se partía. En la sección F.4.3 se detalla en profundidad la construcción de este tipo de escenarios.

Figura 3.9: Aspecto de una bifurcación

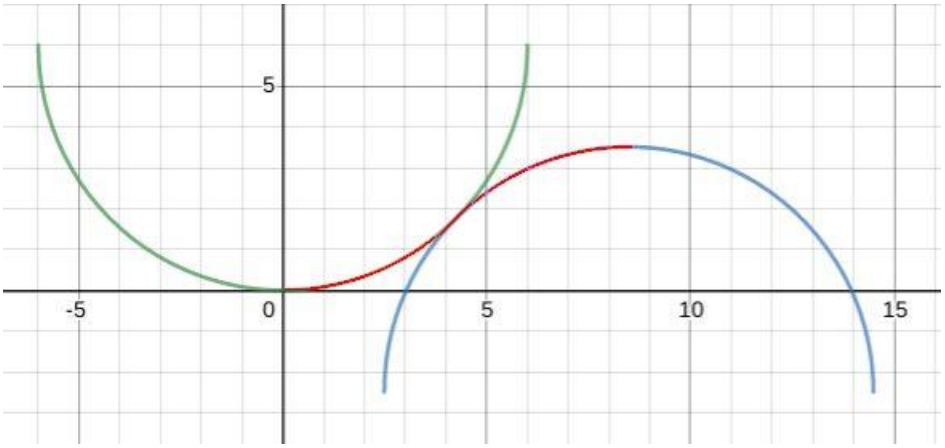


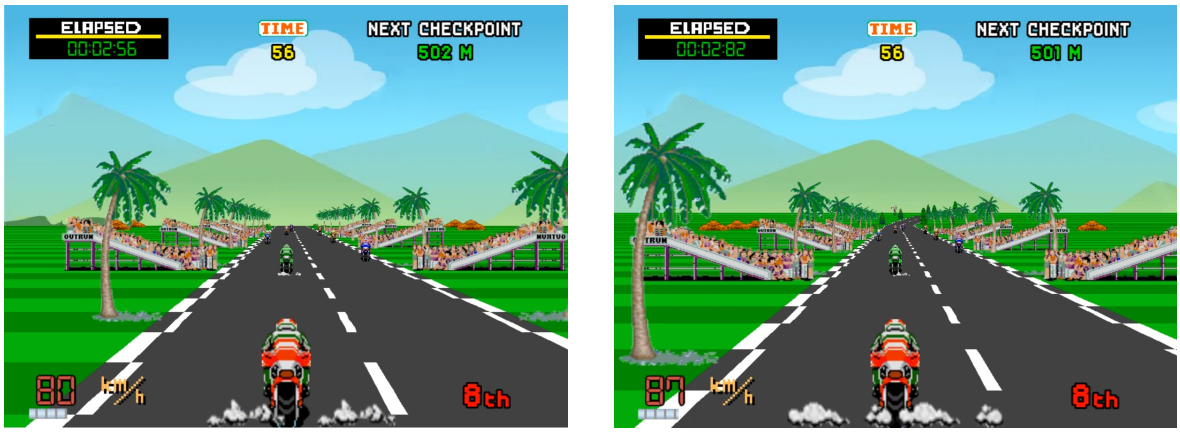
Figura 3.10: Gráfica que muestra la apariencia de las bifurcaciones

Todos los escenarios diseñados están formados por rectángulos horizontales. Cada rectángulo contiene el trazado de la carretera, curvado o recto, llano o con elevación, y los posibles objetos de cada lado de la carretera. Los rectángulos están divididos a su vez en líneas para que los elementos del escenario se muestren separados y que los vehículos se muevan poco a poco por dentro de los rectángulos en vez de a saltos.

3.3.3.5. Punto de fuga

Un problema notable en este proyecto ha sido lograr sensación de profundidad en los escenarios. Al ser gráficos 2.5D se han tenido que hacer muchos cálculos para poder determinar el punto de fuga[11] de la escena, es decir, el lugar en el que convergen dos o más líneas paralelas (reales o imaginarias) hacia el infinito en una imagen. En este contexto, las líneas paralelas son los bordes y carriles de la carretera.

La sensación de perspectiva es crucial ya que permite crear una sensación de horizonte muy realista. El punto de fuga es quién da la sensación a los jugadores de que los gráficos son en 3D. Se han hecho muchas pruebas para colocarlo a una distancia adecuada, ni muy cercana ni muy lejana (véase figura 3.11). En ambos casos, la escena resultante no sería realista.



(a) Punto de fuga muy cercano (b) Punto de fuga a distancia correcta

Figura 3.11: Análisis del realismo del punto de fuga

3.3.4. Geometría de los escenarios

El modelado de la trayectoria de la calzada se ha elaborado a partir de un conjunto técnicas matemáticas de proyección gráfica en 2.5D[12] similares a las utilizadas en el proyecto *Out Run*[13], implementado para C++. Estos conocimientos no se han considerado como parte de este proyecto porque ya se conocían pero si que se ha hecho uso de ellos por la gran estética y apariencia visuales que permitieron conseguir. En la sección F.3 del informe se pueden consultar detalles sobre las técnicas empleadas.

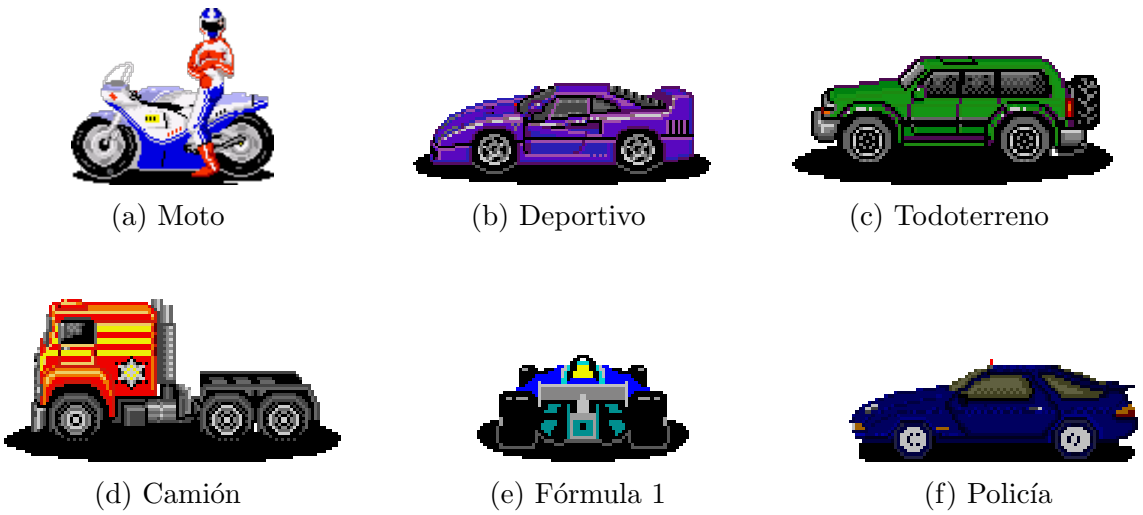
[11] https://www.dzoom.org.es/punto-de-fuga/
[12] https://codeincomplete.com/articles/javascript-racer/
[13] https://github.com/ZgzInfinity/OutRun

3.4. Gestión de los vehículos

A fin de lograr un mayor grado de realismo, dinamismo y originalidad, *Multi Race Driving* ha sido diseñado para soportar varios tipos de vehículos.

3.4.1. Vehículos controlados por el jugador

Los vehículos diseñados están disponibles para los jugadores en todos los modos de juego, incluido multijugador. No obstante, disponibilidad no es sinónimo de rentabilidad. Los vehículos son muy variados (véase figura 3.12). Las características de cada tipo de vehículo tienen valores establecidos por defecto, de manera que se pueda notar una clara diferencia al conducirlos.



(a) Moto (b) Deportivo (c) Todoterreno

(d) Camión (e) Fórmula 1 (f) Policía

Figura 3.12: Tipos de vehículos disponibles en *Multi Race Driving*

Todos los bólidos se han diseñado con unas caracterísiticas, como la marca o la escudería, la velocidad en Km/h, el ángulo de giro (expresada en unidades debido a que es en una métrica ficticia) o el modelo del motor, que los hacen ser muy diferentes los unos de los otros, haciendo que para según que modos de juego, algunos sean mucho más propicios que otros (véase tabla 3.1). Estas características son configurables. Cada bólido, con la excepción del coche policial, tiene una paleta de ocho colores para escoger libremente. Cada vehículo tiene un fichero de configuración *XML* donde se hallan recogidas todas sus especificaciones. Además, las marcas de los vehículos, modelos de motor y escuderías son ficticias y no se corresponden con la realidad. En la sección G.3 del informe se describe de modo detallado como modificar las propiedades de los vehículos jugables.

Vehículos disponibles y sus características				
Tipo	Marca	Velocidad	Giro	Motor
Moto	Suzuki GSX-RR	300 Km/h	0.049 u	Yamaha yzWR 260
Deportivo	Devastator IX	280 Km/h	0.048 u	Kia Stinguer V6
Todoterreno	Muscle PCSX2	270 Km/h	0.046 u	Buick Lacrosse xsf
Camión	Scania XLION	260 Km/h	0.046 u	DAF 95XF 430
Fórmula 1	Escudería	310 Km/h	0.049 u	Motor de la escudería
Policía	Porsche 928	320 Km/h	0.048 u	SX mezgner 911

Tabla 3.1: Características por defecto de los vehículos de *Multi Race Driving*

Las físicas son uno una pieza angular de este proyecto. Al no recurrir a motores gráficos, la cinemática[14] y la dinámica[15] de los vehículos han tenido que ser programadas a mano. Para lograr efectos de inercia, aceleración, velocidad, frenado y colisión realistas se han utilizado funciones matemáticas simples que han permitido obtener aproximaciones visuales muy cercanas a la realidad.

La cinemática se ha conseguido con una función matemática (véase figura 3.13) que representa fácilmente el comportamiento de un motor real. La función crece rápido con valores pequeños (el motor parte del reposo y acelera rápido), y con valores grandes tiende a estabilizarse a un cierto valor asintótico (el motor alcanza la velocidad máxima). La sección G.1 describe en profundidad la cinemática de los vehículos.

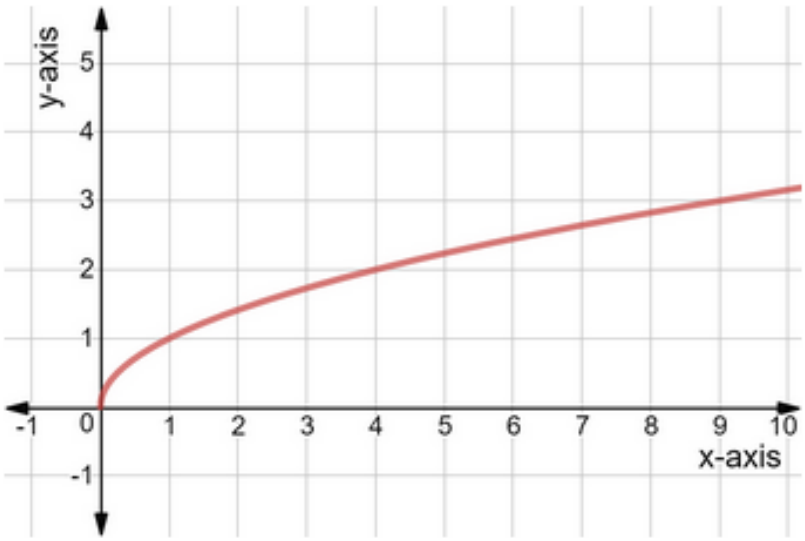
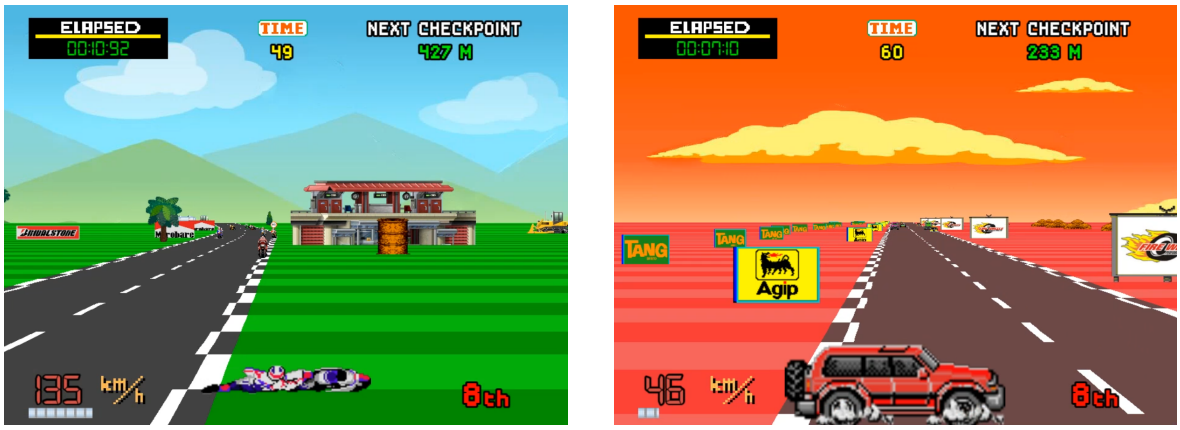


Figura 3.13: Relación entre la aceleración (eje X) y la velocidad (eje Y)

[14] https://es.wikibooks.org/wiki/F%C3%ADsica/Cinem%C3%A1tica
[15] https://es.wikipedia.org/wiki/Din%C3%A1mica

Otra cuestión física sumamente importante ha sido el efecto de la colisión. Tener varios tipos de bólidos, algunos de ellos con un número y tipo de *sprites* distintos, ha dificultado mucho esta tarea ya que no se ha podido realizar una animación común para todos ellos. Para poder crear este efecto se ha diseñado un algoritmo que de forma simple puede calcular las trayectorias que los vehículos deben seguir desde el momento del impacto hasta quedarse sin velocidad, es decir, la trayectoria del choque.

Se ha invertido mucho tiempo para poder implementar un algoritmo que tomando la velocidad y posición inicial del golpe, pueda obtener una trayectoria de colisión realista ya que aunque estos factores anteriores influyen directamente, son muchas tanto las formas como los elementos contra los que el vehículo puede chocar (rivales, tráfico, obstáculos). El algoritmo debía ser lo suficientemente versátil como para poder modelar de manera realista un abanico de posibilidades tan amplio. En este aspecto, además de a la física, también se ha dedicado bastante tiempo al tema gráfico y sonoro puesto que se han tenido que modelar diversos fenómenos como derrapes, trompos, así como buscar efectos de sonido que enriqueciesen estas animaciones (véase figura 3.14). En la sección G.2 del informe se puede encontrar más información sobre el modelado de la dinámica de los vehículos.



(a) Colisión de una moto

(b) Colisión de un todoterreno

Figura 3.14: Ejemplos de animaciones de colisión

3.4.2. Vehículos controlados por el ordenador

Una parte muy importante de este proyecto ha sido la creación de la inteligencia artificial (*IA*). Pues es ésta la encargada de coordinar el comportamiento tanto del tráfico como de los vehículos rivales.

El comportamiento de la *IA* es bastante variado y depende en todo momento tanto de la dificultad con la que se esté jugando como del modo de juego seleccionado. Se han diseñado un total de cuatro modos de *IA* distintos, los cuáles, se explican a continuación.

- *Aleatorio*: este es un modo de *IA* muy sencillo ya que aquellos vehículos que lo poseen avanzan realizando acciones de forma aleatoria, es decir, no tienen en cuenta el comportamiento del vehículo del jugador. Solamente se aplica al tráfico.
- *Obstáculo*: este modo de *IA*, a diferencia del anterior, si que depende del comportamiento del jugador. Es aplicado tanto al tráfico como a los vehículos rivales. Los vehículos que tienen este tipo de *IA* tratan de bloquear el paso al jugador o de adelantarle y ponerse por delante de él si está próximo.
- *Evasivo*: este modo de *IA* se aplica tanto al tráfico como a los vehículos rivales y, al igual que el modo anterior, si que depende del comportamiento del jugador pero hace justamente lo contrario, trata de alejarse todo lo posible para no chocarse con él.
- *Maníaco*: este modo de *IA* se aplica solo a los coches rivales, y es muy diferente a los anteriores. No tiene en cuenta el comportamiento del vehículo del jugador sino los movimientos de los coches del tráfico más cercanos a él. Los vehículos que tienen este tipo de *IA* tratan de evitar por todos los medios chocarse con el tráfico.

Los coches de tráfico en casi todos los modos de juego son del tipo *Evasivo* y *Aleatorio*, a excepción del *Out Run*, ya que al ser los únicos enemigos, están más nivelados con el modo *Obstáculo*. Los coches rivales suelen tener modo *Obstáculo* en los modos *World Tour* y *Pole Position*, con el fin evitar ser adelantados por el jugador o de adelantarle, según sea la situación. En los modos *Driving Fury* y *Demarrage*, el modo que poseen es el modo *Maníaco*, siendo así mucho más difícil adelantarlos o golpearlos dada su capacidad para esquivar el tráfico. Paralelamente, existe un factor de agresividad que condiciona como de estricto debe de ser el comportamiento de la *IA*.

La asignación de los tipo de *IA* se efectúa través de una función de probabilidad uniforme que, con un número aleatorio entre cero y uno, determina que tipo de *IA* asignar. Se han usado un conjunto de cotas límite para delimitar los intervalos de asignación. Los umbrales varían con la dificultad. De este modo, se consigue que para cada nivel de dificultad, unos tipos de *IA* sean más probables que otros, tanto para el tráfico como para los vehículos rivales (véanse figuras 3.15 y 3.16), respectivamente. El

modo *Maníaco* esta reservado única y exclusivamente para los modos de juego *Driving Fury* y *Demarrage*. En dichos modos de juego los tipos de *IA Evasivo* y *Obstáculo* no están presentes para los rivales.

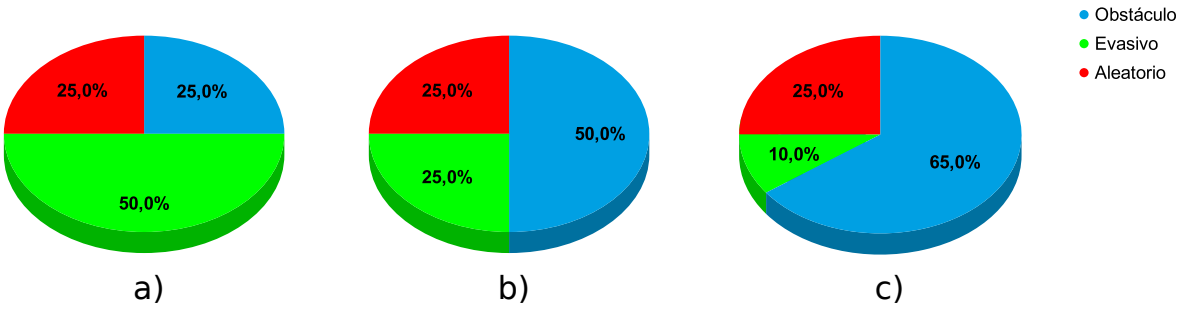


Figura 3.15: Tráfico en modo fácil (a), normal (b) y difícil (c)

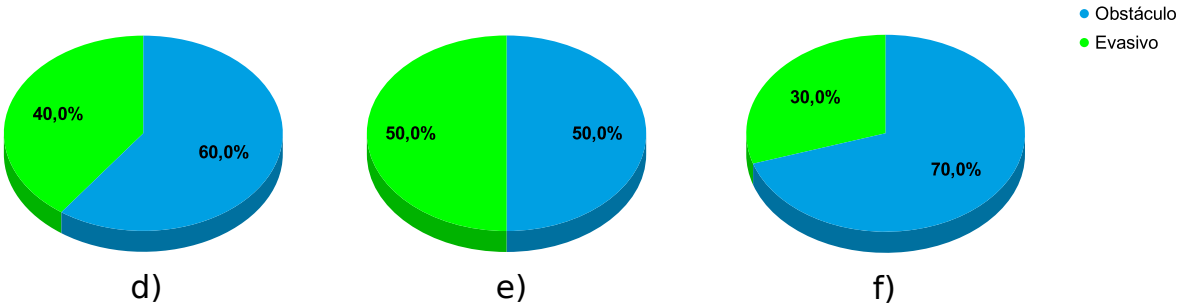


Figura 3.16: Rivales en modo fácil (d), normal (e) y difícil (f)

En el anexo H se pueden consultar los aspectos más técnicos en cuánto a las características e implementación de los vehículos de tráfico y rivales.

3.5. Música y *SFX*

Multi Race Driving ha estado siempre orientado a proporcionar un audio de alta calidad. Cuenta con 23 pistas musicales y con 127 efectos de sonido. Algunas de las bandas musicales son fijas y otras son dinámicas (se pueden cambiar) durante el transcurso de las partidas. La colección de efectos de sonido combinada con los gráficos genera un ambiente de conducción mucho más realista.

Ninguno de los efectos de sonido ni ninguna de las bandas sonoras ha sido compuesta por el desarrollador de la arquitectura, sino que han sido extraídas de otros videojuegos de conducción muy similares. En el anexo I se ofrece más información detallada tanto de la música y *SFX* empleados como de su gestión.

Capítulo 4

Modo multijugador *online*

El modo multijugador ha sido la parte más dificultosa del proyecto. Es un módulo independiente de la arquitectura. Para la implementación se ha usado *multithreading*. El estado del juego lo controla el hilo principal, mientras que el envío y recibo de mensajes por la red lo controlan hilos aparte.

Para mantener operativo el modo multijugador se ha recurrido a *AWS*, un servicio de *cloud computing*. Un problema a afrontar ha sido la latencia física de la red. La latencia promedio desde el lugar de residencia del desarrollador hasta *Virginia* (*EEUU*), lugar donde se encuentra el servidor remoto, es de 100 ms. Dicho coste ha sido medido como el tiempo transcurrido entre el envío de la petición, su procesado interno por el servidor, y su regreso al cliente. En la sección J.1 se puede observar el proceso de despliegue del servidor en *AWS*.

El diseño del modo multijugador se ha apoyado en el esquema cliente-servidor (véase figura 4.1) debido fundamentalmente a que es muy modular y puede integrarse fácilmente en los sistemas. Sobre dicho modelo, se sustenta un sistema altamente modular de código abierto, denominado *Boreas*[1]. Es una implementación, programada en *C++*, altamente rápida y eficiente del lenguaje de coordinación *Linda*[2], muy utilizado en sistemas distribuidos. *Linda* es un sistema de coordinación distribuido en el que los procesos se comunican entre sí a través de tuplas. *Linda* hace uso de un espacio lógico de memoria compartida, el espacio de tuplas. En él, se pueden dejar tuplas, se puede consultar la existencia de tuplas o retirarlas de manera selectiva.

[1] `https://github.com/aeri/Boreas`

[2] `https://www.researchgate.net/publication/237558957_On_the_Use_of_Linda_as_a_Framework_for_Distributed_Database_Systems_Position_Paper`

Las tuplas son listas planas, en cuyas componentes, se almacenan los datos a compartir. En las secciones J.2 y J.3 del informe se explica con detalle el funcionamiento de *Linda* y las características principales de *Boreas*, respectivamente.

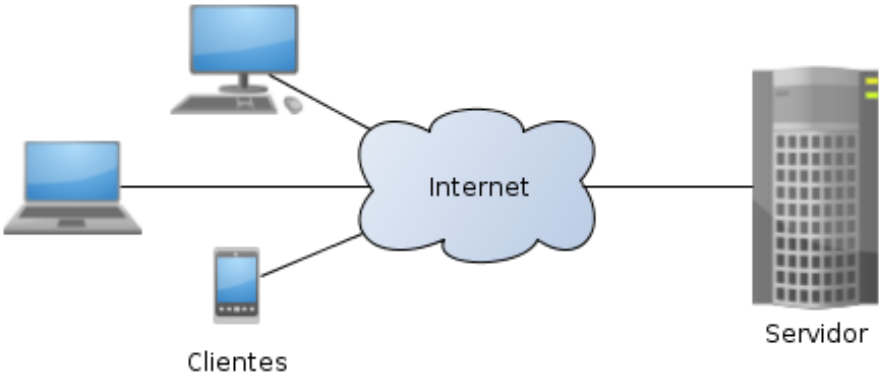


Figura 4.1: Modelo cliente-servidor

La estructuración de los jugadores se ha hecho mediante grupos. En este contexto, los jugadores pueden crear grupos a los que se pueden unir otros jugadores (propietarios), o pueden unirse a grupos (invitados). Los grupos son como máximo de cuatro jugadores. *World Tour* y *Pole Position* son los únicos modos disponibles.

Durante la formación de grupos, selección de modo de juego, de circuitos y de vehículos se notifica el mínimo número de mensajes que garantiza la correcta sincronización. Durante las carreras se notifica de manera continua la situación de cada jugador. En definitiva, la coordinación está marcada por la ausencia de esperas y de latencias internas, salvo la física de la propia red, con el fin de brindar una correcta experiencia de juego. A continuación, se explican las etapas del modo multijugador.

4.1. *Test* de conexión al servidor remoto

En esta etapa el cliente realiza un *test* para medir la calidad de conexión con el servidor remoto. Haciendo uso del comando *shell ping* se obtiene un diagnóstico del estado de la conexión. Como resultado, se obtienen los tiempos de latencia mínima, media y máxima junto con un indicador de cómo de fluida va a ser la experiencia de juego. Así, antes de comenzar, el jugador ya sabe en que condiciones va a jugar. En la sección J.4 se explica esta fase con más precisión.

4.2. Formación de grupos

En esta etapa los jugadores deben introducir el apodo con el que identificarse en el sistema y si desean crear o unirse a un grupo, ya sea un grupo concreto validando el apodo o un grupo aleatorio. Tanto los jugadores como los grupos se identifican con apodos de hasta como máximo un total de ocho caracteres, todos ellos alfanuméricos (véase figura 4.2). El jugador debe determinar si desea crear o unirse a un grupo, y después, debe validar su apodo en el sistema. Si desea unirse a un grupo aleatorio, son el cliente y el servidor quiénes se encargan de emparejar al jugador en un grupo correcto (grupo abierto con no más de cuatro jugadores).

Figura 4.2: Ejemplo de plantilla de grupo

En esta etapa se han tenido que contemplar varios aspectos de sincronización. Se ha tenido que desarrollar un sistema que permite guardar en tiempo real los apodos de todos los jugadores mientras están jugando. Del mismo modo ocurre con los apodos de los grupos. En el momento en el que un grupo es creado, se guarda su alias hasta que el propietario lo cancela. Este sistema garantiza que nunca puede haber dos jugadores o salas con el mismo apodo, lo cuál facilita la identificación de ambos. Además, permite la reutilización de apodos ya que si un jugador abandona o un grupo se cancela, los apodos pueden ser usados de nuevo ya que son borrados del sistema.

También ha sido necesario sincronizar las uniones de los jugadores a los grupos. Se ha tenido que coordinar que las peticiones de unión de los jugadores les sean notificadas a los propietarios de los grupos, y además, que dichas peticiones sean confirmadas si el grupo no está cerrado (no admite más jugadores aunque no haya superado el

límite) o si está abierto y no se supera la capacidad máxima. Si un grupo no cumple los dos requisitos anteriores, las solicitudes son denegadas. También se ha tenido que sincronizar que cuando el propietario recibe la solicitud de unión de un jugador, si es aceptado, se comunique al resto de integrantes del grupo que hay un nuevo miembro. En este punto, también ha sido necesario controlar tanto que el grupo se cierre cuando el propietario lo ordena como que se marque como completado si ya hay cuatro jugadores. De esta manera, se garantiza que futuras peticiones de unión por parte de otros jugadores sean denegadas.

Para no sobrecargar al sistema con información excesiva, se ha diseñado un algoritmo que permite que cada jugador localmente guarde los distintos miembros de su equipo. De esta manera, el servidor solamente gestiona peticiones de unión y borrado de jugadores, y son los propios integrantes quiénes localmente se encargan de hacerlas efectivas. Además, en esta fase también se ha tenido que asegurar que si un jugador se une a un grupo aleatorio, el servidor debe buscar el primer grupo que sea válido, es decir, que esté abierto y que el número de integrantes no exceda el máximo permitido. Escogiendo el primer grupo válido se garantiza el mínimo tiempo de búsqueda. Esta fase se explica en profundidad en la sección J.6.1.

4.3. Selección de modo de juego

En esta etapa el propietario escoge a que modo de juego se va a jugar, *World Tour* o *Pole Position*. Los invitados, por su parte, esperan. Si el propietario selecciona *Pole Position*, el siguiente paso es determinar en que circuito correr y cuántas vueltas dar. Si se escoge *World Tour*, se procede directamente a elegir vehículo.

En esta fase se ha tenido que sincronizar que los jugadores invitados reciban correctamente el modo de juego escogido por el propietario. Además, también se ha tenido que controlar la posibilidad de que el propietario pueda cancelar el grupo recién formado. El aspecto de coordinación más costoso ha sido poder encontrar un algoritmo que permita poder monitorizar los posibles fallos por caída de los invitados. Dicho algoritmo se basa en el envío de notificaciones continuas por parte de los invitados al propietario. Las notificaciones son pulsos que el propietario consulta para poder deducir si los invitados se han caído o no. Existe un registro de peticiones fallidas por invitado controlado por el propietario. Cada consulta fallida (el invitado no ha mandado pulso) incrementa en una unidad su contador. Si supera el límite, el

propietario elimina de su registro de jugadores local ese jugador y notifica al resto que ese jugador debe ser eliminado. Si el propietario recibe el pulso de ese jugador reinicia su registro de peticiones fallidas a cero. De este modo, todos los miembros del grupo permanecen coordinados. Para más información sobre esta fase se puede consultar la sección J.6.2 del informe

4.4. Selección de circuitos

Esta fase solo tiene lugar si el modo de juego es *Pole Position*. El propietario selecciona el circuito donde correr y el número de vueltas. Se ha diseñado una técnica que permite asignar a cada circuito un identificador. De este modo, cuando el propietario escoge el circuito, manda a cada miembro el identificador correspondiente a ese circuito y las vueltas a dar. En esta fase el sistema de control de pulsos sigue en curso para seguir asegurando el control ante fallos por caída de los jugadores. En la sección J.6.3 se recogen más detalles sobre esta fase.

4.5. Selección de vehículos

En esta etapa los jugadores escogen con que bólido van a correr. Se ha diseñado un algoritmo de sincronización mediante barrera en el que cuando un jugador escoge vehículo (tipo y color), manda al resto de integrantes esta información y espera a que el resto de jugadores también lo hagan. De esta manera, la carrera no es comenzada hasta que todos han elegido. Cada jugador carga localmente el vehículo de cada integrante. Cuando un jugador selecciona vehículo no puede ni abandonar el grupo ni deshacer su elección. Sin embargo, si el jugador no ha escogido coche si puede abandonar el grupo. Si un jugador abandona se lo notifica al propietario y éste a su vez al resto de integrantes. Tras haber seleccionado vehículo, todos los corredores cargan la partida localmente con su vehículo, los bólidos del resto de jugadores y el circuito. El orden de salida, si es la primera carrera, es el orden cronológico de incorporación al grupo. Si no, viene marcado por la clasificación de la carrera anterior. Para más información sobre esta fase se puede consultar el apartado J.6.4 del informe

4.6. Carrera

Esta es la fase donde se sincroniza toda la acción que ocurre durante las carreras (véase figura 4.3). La posición y el estado de los jugadores se notifica continuamente con el

objetivo de conocer con mayor exactitud en qué contexto está cada jugador. A nivel local también se han tenido que controlar posibles fallos de sincronización debido a la exclusión mutua ya que no puede permitirse simultáneamente cambiar y enviar tanto el estado como la posición del jugador. Se han usado semáforos como mecanismo de sincronización. El estado de un jugador ha sido representado con un identificador que indica la textura que se debe dibujar. Al tener guardado el tipo y color del vehículo de cada jugador, para poderlos visualizar se tiene que cargar en el *sprite* la textura correspondiente al identificador. Además, también se comunica la posición de cada jugador en el mapa para poderlos dibujar en sus respectivas ubicaciones.

(a) Primer jugador

(b) Segundo jugador

(c) Tercer jugador

(d) Cuarto jugador

Figura 4.3: Perspectivas de parrilla de salida

Cuando el jugador llega a la meta, notifica su posición al resto de corredores y permanece a la espera en otra barrera hasta que estos lleguen. Cuando llega el último corredor, todos promocionan a la siguiente fase (selección de circuito si se juega a *Pole Position* o siguiente carrera si se juega a *World Tour*). En el apartado J.6.5 del informe se pueden encontrar más detalles sobre esta fase.

Capítulo 5

Evaluación del prototipo

En este capítulo se describen las distintas plataformas para las que *Multi Race Driving* ha sido preparado para ser ejecutado y el proceso de *testing* al que ha sido sometido para poder verificar su calidad.

5.1. Plataforma

La arquitectura se ha diseñado para poder ser ejecutada exclusivamente en computadores de sobremesa y/o en *PC*'s. *Multi Race Driving* es compatible principalmente con *Windows*, siempre y cuando la arquitectura del sistema operativo sea de 64 *bits*. También se ha probado su funcionamiento en *Linux* y *Mac* mediante *WineHQ 5.0*[1].

Para poder compilar *Multi Race Driving* se ha utilizado el compilador *MinGW*[2], versión *3.82.90*, que permite compilar programas escritos en *C* y *C++*. El ejecutable ha sido preparado para que los usuarios no tengan que instalar ningún tipo de aplicación auxiliar en *Windows* para poderlo arrancar. Todas las dependencias, tanto las de *MinGW* como las de *SFML*, han sido *linkadas* de manera estática, es decir, todos las librerías usadas han sido integradas dentro del fichero ejecutable, a excepción de la librería dinámica de gestión y reproducción del audio, el fichero *openal32.dll*. Debido a que esta librería se halla bajo la licencia *GPL*[3], el fichero debe ser integrado junto con el ejecutable vaya o no a usarse con fines comerciales.

[1] `https://www.winehq.org/`
[2] `http://www.mingw.org/`
[3] `https://es.wikipedia.org/wiki/GNU_General_Public_License`

5.2. Evaluación

Una parte clave en el desarrollo de videojuegos es el *Game Testing*[4]. Esta etapa es un proceso de prueba de *software* para el control de calidad de los videojuegos. Las razones por las que se efectuó este proceso eran poder conocer las opiniones sobre el *software* de un conjunto de jugadores reducido, los *game testers*, con el fin de poder estimar cuál será la aceptación de *Multi Race Drivng* por parte del público, y como patrón de descubrimiento y documentación de defectos de *software*.

Los *game testers* probaron todas las funcionalidades (modos de juego para un solo jugador, configuración de controles, la navegabilidad por los menús, resoluciones de gráficos, modo multijugador, etcétera). A cada *tester* se le pidió anotar los aspectos atractivos del *software* y los matices que mejorarían o cambiarían. También se les pidió que anotasen los posibles fallos o *bugs* que hallasen con el fin de ayudar a depurar el *software* de la versión *beta*.

La siguiente tabla recoge los principales aspectos de *Multi Race Driving* evaluados por los *beta testers* (véase tabla 5.1). Cada fila se corresponde con una funcionalidad y cada columna se corresponde con un *beta tester*. Para cada característica, cada *beta tester* anotó un ✓ si le pareció correcta, una ✗ si consideró que debía ser mejorada, o un - si no aportó ninguna retroalimentación al respecto.

Evaluación de *Multi Race Driving*					
Característica	*Beta Testers*				
Gráficos	✓	✗	✓	✗	✓
Interfaz	-	✗	-	✓	✗
Físicas	✓	✓	✓	-	✓
Música y *SFX*	✓	✓	✓	✓	✓
Controles	✓	-	✓	✓	✓
Configurabilidad	✓	✓	✓	✓	✓
Multijugador	✗	✓	✗	✓	-
Jugabilidad	✓	✗	✓	✗	✓
Originalidad	-	✓	-	✓	✓

Tabla 5.1: Resultados de la evaluación de *Multi Race Driving*

En el anexo K se pueden encontrar los distintos *game testers* que participaron en la evaluación del prototipo y cuáles fueron sus valoraciones.

[4] `https://en.wikipedia.org/wiki/Game_testing`

Capítulo 6

Conclusiones

El objetivo de *Multi Race Driving* era crear una plataforma que integrase varios modos de conducción en los que los jugadores pudiesen vivir auténticas experiencias de conducción. Tomando como base algunos de los videojuegos más exitosos de los 80, se buscaba poder brindar dichas experiencias creando nuevas versiones que conservasen el espíritu de los juegos originales y que tuviesen aspectos novedosos que cautivasen al público. Para aportar estas innovaciones se ha desarrollado un modo multijugador *online*, se han diseñado varios tipos de vehículos y se ha permitido que todos los aspectos de la arquitectura sean parametrizables.

También es importante mencionar la existencia de ciertas extensiones que se había pensado introducir en *Multi Race Driving*. Sin embargo, debido a ciertos imprevistos no han podido ser integradas. Por ello, se han dejado como posibles extensiones del proyecto. Estas ampliaciones y los motivos por los que no se han incorporado pueden consultarse con más detalle en la sección L.1 del informe.

6.1. Dedicación y gestión del proyecto

La gestión del proyecto *software* ha sido un aspecto que se ha tratado de cuidar en todo momento. El proyecto se ha compuesto de diversas tareas. Para cada una se ha indicado las horas dedicadas, su fecha de inicio y su fecha de finalizado. En total se han contabilizado 960 horas de trabajo para la realización del proyecto durante el transcurso de cinco meses. Además, se ha realizado un diagrama de *Gantt* (véase figura 6.1) que expone de un modo gráfico, el tiempo de dedicación para las actividades realizadas. En el diagrama, las tareas han sido clasificadas de acuerdo a su dificultad. Así las cosas, las tareas de color verde se corresponden con una dificultad fácil, las de color amarillo con un nivel de dificultad normal, y las de color rojo con un nivel de dificultad alto.

Nombre	Fecha de ini...	Fecha de fin	Horas
Creación de escenarios	17/2/20	3/7/20	50
Análisis y pruebas del software	17/2/20	7/10/20	130
Diseño de vehículos	19/2/20	21/8/20	74
Generador de escenarios	2/3/20	14/9/20	30
Generador de menús	9/3/20	19/10/20	47
Incorporación de música y SFX	16/3/20	4/9/20	20
Diseño de la interfaz	24/3/20	14/9/20	70
Implementación de físicas	17/4/20	22/7/20	40
Diseño de los modos individuales	8/6/20	14/8/20	142
Integración de los modos individuales	13/7/20	29/9/20	50
Implementación de la IA	23/7/20	14/8/20	90
Modo multijugador	14/8/20	2/10/20	180
Memoria	20/8/20	7/10/20	60
Documentación, maquetación y tráiler	28/9/20	7/10/20	27

Figura 6.1: Diagrama de *Gantt* del proyecto

6.2. Problemas encontrados

A lo largo de la elaboración del proyecto se han tenido que abordar diversos problemas que han dificultado el progreso a la hora de diseñar la arquitectura. Algunos de los más destacados son los que a continuación se citan:

- La generación de los escenarios de nivel de los distintos modos de juego ha sido un aspecto muy difícil de tratar por parte del desarrollador de la plataforma. Ha sido necesario invertir un tiempo bastante considerable en la búsqueda de fondos y *sprites* para los paisajes dado que desde el principio se pensó que fueran distintos visualmente y que cada uno tuviese una personalidad y apariencia únicas. Por ello, se han precisado varias horas de búsqueda para localizar todos los *sprites* y fondos en cuestión.

- Las bifurcaciones también supusieron un reto dado que costó encontrar una técnica que permitiese la generación de éstas de una forma realista y adecuada desde el punto de vista estético. Además, hubo que hacer frente a diversos cálculos matemáticos complejos y a largos periodos de pruebas hasta verificar que los resultados obtenidos eran los esperados.

- Proporcionar al jugador control del vehículo a través de un *gamepad* o mando también fue un duro reto que no se pudo superar debido a que *SFML* no ofrece compatibilidad con los *drivers* de ciertos controladores, como es el caso de los mandos de *PS3*, que no eran detectados. De igual modo, se intentó probar con mandos de *PS2* pero la librería por razones desconocidas no reconocía todos los botones del mando, solo algunos.

– El desarrollo del modo multijugador ha sido, sin duda alguna, el factor de mayor complejidad de todo el proyecto. La principal razón es que nunca se había diseñado antes un modo multijugador y al principio no se tenía muy claro como poder abordar los problemas de sincronización y comunicación de un modo práctico y eficiente. Otros factores como la disponibilidad de los *beta testers* para poder probar el multijugador o la latencia de la red presente también fueron altamente influyentes en el tiempo invertido para poder probar y *debuggear* su comportamiento.

6.3. Opinión personal

Multi Race Driving ha sido sin duda alguna el reto más duro al que me he tenido que enfrentar en todo el Grado. Varios han sido los problemas a los que he tenido que hacer frente a lo largo del proyecto (físicas, generación de escenarios y de la *GUI*, implementación de la *IA*, etcétera). Sin embargo, el reto más difícil ha sido el modo multijugador debido a que éste ha sido el primero que he hecho.

El pilar en el que se ha sustentado este trabajo radica en comprender y afrontar en primera persona las diversas fases de desarrollo por las que debe pasar un videojuego a lo largo de su proceso de producción. No obstante, también se ha procurado en todo momento que dicho proyecto sirva tanto de estudio de todo el proceso, como de guía. Así pues, servirá de ejemplo a todos aquellos desarrolladores que quieran iniciarse tanto en la creación y/o adaptación de videojuegos antiguos en diversas plataformas de distribución.

Inicialmente, *Multi Race Driving* era solamente una simple idea. El hecho de poder haberla transformado en el *software* que es actualmente es sin duda alguna lo que más me ha llenado de satisfacción. Conocimientos adquiridos en muchas asignaturas del Grado como *Programación de sistemas concurrentes y distribuidos*, *Sistemas operativos*, *Sistemas de información*, *Proyecto Software*, *Redes de computadores*, *Inteligencia artificial* o *Videojuegos* han sido empleados en este proyecto. Puedo concluir, sin lugar a dudas, que *Multi Race Driving* es el resultado de la unificación de todo lo aprendido.

Capítulo 7

Bibliografía

[1] Alejandro Crespo. 150 videojuegos a los que tienes que jugar al menos una vez en la vida. 2010.

[2] Tony Mott. 1001 videojuegos a los que hay que jugar antes de morir. *Barcelona: Grijalbo*, 2011.

[3] Margaret A Ellis and Bjarne Stroustrup. *The annotated C++ reference manual*. Addison-Wesley, 1990.

[4] Jan Haller and Henrik Vogelius Hansson. *SFML Game Development*. Packt Publishing Ltd, 2013.

[5] Albrecht Schmidt and Kjetil Nørvåg. Rapid xml database application development. 2004.

[6] Richard H Carver and Kuo-Chung Tai. *Modern multithreading: implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs*. John Wiley & Sons, 2005.

[7] Tony Manninen. Interaction forms and communicative actions in multiplayer games. *Game studies*, 2003.

[8] Ashish Deshpande and Martin Schultz. Efficient parallel programming with linda. 1992.

[9] Jinesh Varia. Migrating your existing applications to the aws cloud. *A Phase-driven Approach to Cloud Migration*, 2010.

[10] Zheng Xue, Di Wu, Jian He, Xiaojun Hei, and Yong Liu. Playing high-end video games in the cloud: A measurement study. *IEEE Transactions on Circuits and Systems for Video Technology*, 2014.

Anexos

Anexo A

Tecnologías y servicios

En esta sección del documento se procede a explicar cuáles han sido las tecnologías empleadas para el desarrollo de la arquitectura, detallando los fundamentos por los que se ha escogido cada una de ellas.

A.1. Lenguaje de implementación

Para programar *Multi Race Driving* el lenguaje que se ha utilizado ha sido *C++*[1]. Las razones por las que se ha decidido utilizar este lenguaje de programación en alto nivel y no otros como por ejemplo *Java* son las siguientes:

- Ofrece un alto rendimiento dado que la compilación genera un código que es muy cercano al lenguaje ensamblador.
- Es un lenguaje muy actualizado, permitiéndole crear, relacionar y operar con datos complejos.
- Ofrece compatibilidad con bibliotecas. De este modo, se puede hacer uso de rutinas que ayudan a escribir código fácilmente.

Como entorno de desarrollo integrado (*IDE*) donde programar se ha empleado *CodeBlocks*[2] (versión 17.12). *CodeBlocks* es un entorno de desarrollo totalmente *open source* para programar en *C++* que ha sido empleado por las siguientes razones:

- Es súper extensible y fácilmente configurable.
- Posee una interfaz sencilla y minimalista que acelera el grado de aprendizaje de los usuarios.

[1] `https://www.programiz.com/cpp-programming`
[2] `http://www.codeblocks.org/`

A.2. Interfaz de usuario

Para abordar todo el tema referente a la programación de los gráficos y la gestión de las bandas sonoras y el *SFX* se ha usado la *API* de *SFML*[3] (*Simple and Fast Multimedia Library*). *SFML* proporciona una interfaz sencilla para los diversos componentes del *PC*, con el fin de facilitar el desarrollo de juegos y aplicaciones multimedia. Seguidamente se exponen cuáles son sus rasgos más característicos:

- Es una *API* muy eficiente.
- Cuenta con una amplía comunidad de usuarios y foros de ayuda. Este factor ha sido de gran utilidad a la hora de resolver problemas y anomalías en el código.
- Es multiplataforma. Actualmente es compatible con *Windows*, *Linux*, *MAC*, y en un futuro muy cercano, ofrecerá compatibilidad con *Android* e *iOS*.
- Posee una documentación muy detallada y llena de ejemplos de código prácticos sobre como hacer un uso correcto de sus servicios.
- Es compatible con muchos lenguajes de programación, entre los que destacan *C++*, *Ruby*, *Java* y *Python*.
- Es muy fácil de usar, incluso para los programadores principiantes.

La decisión de qué *API* emplear no fue fácil, ya que *SFML* no era la única opción. *SDL*[4] y *Allegro*[5] fueron otras posibles alternativas a emplear porque eran muy alabadas por la comunidad. Ante esta incertidumble se realizó una minuciosa comparativa entre todas ellas, que finalizó convirtiendo a *SFML* en la opción más idónea.

A.3. Procesamiento de ficheros *XML*

Para poder trabajar con ficheros en formato *XML* se ha recurrido a la librería *RapidXml*[6]. La biblioteca *RapidXml* es un analizador de texto (*parser*) escrito en *C++* que permite la lectura y escritura de ficheros en *XML* de una manera rápida, eficiente

[3] `https://www.sfml-dev.org/`
[4] `https://www.libsdl.org/`
[5] `https://liballeg.org/`
[6] `http://rapidxml.sourceforge.net/`

y siguiendo los principios del *DOM*[7] (*Document Object Model*). Además, conserva la utilidad, la portabilidad y una razonable compatibilidad con el *W3C*. Las principales causas por las que se decidió emplear este analizador son las que a continuación se enumeran:

- No tiene más dependencias que un subconjunto muy pequeño de la biblioteca estándar de *C++* como *cassert*, *cstdlib*, *new* y *exception*.
- Soporta sin problemas cualquier tipo de codificación de caracteres. Algunos de los más destacados son *UTF-8*, *UTF-16* o *UTF-32*.
- Es muy robusto y confiable dado que ha sudo sometido a una intensa batería de pruebas unitarias por parte de sus desarrolladores.

La decisión de usar *XML* no fue tomada a la ligera, principalmente porque era *JSON* el formato deseado para escribir los ficheros de configuración. Sin embargo, a pesar de que *JSON* es un lenguaje de intercambio de datos que se procesa más rápido que *XML*, la sintaxis y estructura de los ficheros iba a ser muy difícil de comprender. Este aspecto se iba a convertir en un problema a la hora de poder representar los elementos configurables (escenarios, menús, vehículos). Se observó que los ficheros iban a quedar con una longitud muy extensa en cuánto al número de líneas, y además, que la depuración de fallos de sintaxis iba a ser mucho más costosa. Por ello, se escogió usar *XML*.

A.4. Modo multijugador

La tecnología a la que se ha recurrido para tener en funcionamiento ininterrumpido el servidor encargado de gestionar el modo multijugador ha sido *AWS* (*Amazon Web Services*). La principal razón por la que se ha decidido emplear este servicio de *cloud computing* y no otros como por ejemplo *Microsoft Azure* o *Google Cloud* es porque de los tres es la alternativa más económica en cuánto a la relación entre calidad y precio.

Además, *Amazon* ofrece un servicio de prueba gratuito durante un plazo de como máximo un año. Todos estos factores hicieron de *AWS* la opción más adecuada.

[7] `https://www.w3.org/2005/03/DOM3Core-es/introduccion.html`

Para llevar a cabo la sincronización y comunicación de los jugadores en el modo multijugador se ha utilizado *Boreas*, una implementación *open source* del *framework Linda* desarrollado en *C++* con licencia *GPL-3.0*.[8]

A.5. Control de versiones

El desarrollo de *Multi Race Driving*, ha sido sin duda, uno de los proyectos que más ha costado implementar. Para simplificar los posibles problemas que pudiesen ocurrir, desde un principio se ha llevado una clara organización de las diferentes tareas que había que hacer, así como del tiempo para poder hacerlas. El código del proyecto ha estado integrado en todo momento a un sistema de control de versiones. Como plataforma se ha escogido *Github*[9]. De este modo, el código ha permanecido tanto en el propio computador local del desarrollador como en su correspondiente repositorio en *Github*. En el repositorio solamente ha existido una única rama, la rama *Master*. Dado que el proyecto ha sido realizado por un solo programador, no se ha considerado necesaria la creación de más ramas adicionales. Como herramienta auxiliar a *Github* se ha utilizado *GitHub Desktop*[10], que ofrece una interfaz gráfica desde la que se pueden observar los últimos cambios que se han subido al repositorio.

A.6. Planificación y control de tareas

Para llevar un control exhaustivo de todas las tareas que se debían realizar, y lo más importante, de monitorizar cuánto tiempo real se ha tardado en realizarlas, se ha hecho uso de la herramienta gratuita *Clockify*[11]. Es una sencilla aplicación de seguimiento del tiempo que, haciendo uso de una interfaz minimalista le permite al usuario tanto definir las tareas que tiene que realizar, especificando una descripción de la tarea, a que proyecto está vinculada y estimar cuánto tiempo va a durar dicha tarea, como poder monitorizarlas en tiempo real. Además, se ha empleado *GanttProject*[12] , una herramienta *open source*, con el fin de poder construir de manera sencilla un diagrama de *Gantt* que muestre como se ha llevado a cabo la administración del proyecto de *Multi Race Driving*.

[8] `https://www.gnu.org/licenses/gpl-3.0.`
[9] `https://github.com/`
[10] `https://desktop.github.com/`
[11] `https://clockify.me/`
[12] `https://gantt-project.uptodown.com/windows`

Anexo B

Configuración general de *Multi Race Driving*

En este anexo se explica con más detalle como los jugadores pueden gestionar tanto la configuración de la arquitectura como la del modo multijugador. Dicha configuración, es completamente independiente de la gestión de otros aspectos configurables como por ejemplo los menús de la *GUI* o los escenarios.

B.1. Configuración de la arquitectura

En los ajustes, los jugadores pueden realizar cambios relacionados con los volúmenes del *SFX* y de las bandas musicales, pueden ajustar la calidad de gráficos y la resolución de pantalla, e incluso, modificar los controles. El fichero de configuración *XML* donde reside toda esta información se localiza, al igual que el fichero anterior, en el directorio *Settings*, presente en la carpeta de archivos principal *Data*. Dicho fichero, denominado *Configuration.xml,* presenta la estructura que a continuación se comenta.

En primer lugar, se especifica la versión de *XML* y la codificación de caracteres del fichero. Seguidamente se encuentra el nodo principal del documento, llamado *Settings*. Los nodos hijos de éste son los que condicionan cada uno de los aspectos configurables de *Multi Race Driving*, y son los siguientes:

- Nodo *Difficulty*: permite gestionar el nivel de dificultad. El contenido de dicho nodo tiene como rango de valores *Easy*, *Normal* y *Hard*, para representar los tres modos posibles de dificultad.
- Nodo *Volume_Soundtracks*: sirve para controlar el nivel de audio con el que deben reproducirse las bandas musicales. Si el valor es 0, no se escucha nada y si es 100, es el volumen máximo.

- Nodo *Volume_SounEffects*: sirve para controlar el nivel de audio con el que debe reproducirse el *SFX*. Si el valor es 0, no se escucha nada y si es 100, es el volumen máximo.
- Nodo *PixelArt*: controla si los gráficos se deben dibujar o no con efecto *pixel art*. Puede tomar como valores *Enabled* o *Disabled*, *pixel art* activado y desactivado, respectivamente.
- Nodo *Full_screen*: verifica si se debe jugar o no en modo pantalla completa. Al igual que el nodo anterior puede tomar como valores *Enabled* o *Disabled*.
- Nodo *Resolution_x*: marca la anchura que debe tener la pantalla.
- Nodo *Resolution_y*: especifica la altura que debe tener la pantalla.
- Nodo *Controller_left*: contiene el identificador de la tecla que debe pulsarse para mover al vehículo del jugador hacia la izquierda cuando se está en partida.
- Nodo *Controller_right*: representa el identificador de la tecla que debe pulsarse para mover al vehículo del jugador hacia la derecha cuando se está en partida.
- Nodo *Controller_accelerate*: refleja el identificador de la tecla que debe pulsarse para acelerar cuando se está en partida.
- Nodo *Controller_braking*: informa del identificador de la tecla que debe pulsarse para frenar cuando se está en partida.
- Nodo *Controller_soundtrack*: muestra el identificador de la tecla que debe pulsarse para cambiar de pista musical en partida.

La navegabilidad en los menús de la *GUI* siempre es la misma, es decir, no es configurable. De este modo se garantiza la uniformidad. Así, para seleccionar o aceptar se pulsa la tecla *ENTER* (botón *START*). Para salir de un menú o pausar la partida se debe de pulsar la tecla *ESCAPE* (botón *ESC*). Para navegar en los menús, por ejemplo para seleccionar un botón, se usan las flechas del cursor *UP* y *DOWN*.

Para poder modificar la configuración del software, y en última instancia, el fichero *XML*, el jugador solamente debe ir al menú de opciones y efectuar los cambios pertinentes. Tras salir de dicho menú, *Multi Race Driving* guardará los cambios hechos y los conservará incluso después del apagado, hasta que el jugador desee volver a modificarlos. La otra opción es tocar a mano el fichero *XML* en cuestión, pero dada la

presencia de la otra alternativa, ésta última es muy poco recomendable ya que puede introducir configuraciones inválidas, poner algo mal escrito, etcétera. A continuación, se muestra el fichero *XML* anteriormente descrito:

```
<?xml version="1.0" encoding="UTF-8"?>
<Settings>
  <Difficulty>Hard</Difficulty>
  <Volume_Soundtracks>100</Volume_Soundtracks>
  <Volume_Effects>100</Volume_Effects>
  <PixelArt>Enabled</PixelArt>
  <Full_screen>Disabled</Full_screen>
  <Resolution_x>1366</Resolution_x>
  <Resolution_y>768</Resolution_y>
  <Controller_left>LEFT CURSOR</Controller_left>
  <Controller_right>RIGHT CURSOR</Controller_right>
  <Controller_accelerate>LCTRL</Controller_accelerate>
  <Controller_brake>LALT</Controller_brake>
  <Controller_soundtrack>Z</Controller_soundtrack>
</Settings>
```

B.2. Configuración del modo multijugador

Para configurar el modo multijugador, se ha creado un fichero *XML* donde los jugadores pueden ajustar los parámetros del servidor. El fichero, denominado *Multiplayer.xml* está en la carpeta *Settings* y tiene la siguiente sintaxis. Primero, se especifica la versión de *XML* y la codificación de caracteres del fichero. Seguidamente se encuentra el nodo principal del documento, llamado *Settings*. Los nodos hijos condicionan la configuración, y son los siguientes:

- Nodo *Domain*: dominio o dirección *IP* del servidor donde el jugador tiene alojado *Boreas* para poder conectar desde su cliente.
- Nodo *Port*: puerto del servidor donde el jugador tiene alojado a *Boreas* para poder establecer comunicación desde su cliente.

Mediante esta configuración, los jugadores pueden ajustar el modo multijugador a su gusto. De este modo, cualquier jugador puede tener su servidor personal y configurarlo sin compilar el código. A continuación, se muestra el fichero descrito:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Example configuration file for the multiplayer server of Boreas-->
<Settings>
        <Domain>INSERT BOREAS DOMAIN HERE!</Domain>
        <Port>PORT OF BOREAS P.E 20000</Port>
</Settings>
```

Anexo C

Modos de juego de un solo jugador

En esta sección del informe se procede a describir como se han diseñado los distintos modos de juego disponibles para un solo jugador.

C.1. *World Tour*

En este modo de juego, el jugador corre en carreras que tienen lugar en diversas partes del mundo. El jugador dispone de un tiempo de juego inicial por defecto que varía según el modo de dificultad (fácil, normal y difícil). En el trayecto hay tres puntos de control distribuidos de un modo más o menos equidistante. Cada vez que el jugador pasa por un punto de control se incrementa el tiempo de juego. Si el tiempo expira antes de llegar a la meta el jugador es descalificado.

Mientras corre, el jugador deberá sortear a los vehículos del tráfico y vencer a un total de siete rivales que corren con él. Para poder promocionar al siguiente nivel debe quedar en primera posición, en caso contrario debe repetir la carrera. Este modo de juego cuenta con un total de cuatro escenarios de apariencia muy diversa que representan lugares icónicos del mundo. Estos niveles son *Sherwood Forest*, *Egypt*, *Vancouver* y *Tokio*.

El sistema de puntuación es muy simple. Por cada nivel y dificultad se guarda el tiempo del jugador más rápido en ese nivel y esa dificultad. Si el jugador ha quedado primero, se compara su tiempo invertido con el del récord actual. Si es un tiempo inferior, se le solicita a ese jugador sus credenciales para sustituir el récord actual por el suyo. En caso contrario, el tiempo del jugador es desechado. A continuación, se ofrece una comparativa entre el modo de juego *World Tour* (véase figura C.1) y el videojuego en el que se ha inspirado, *Super Hang On* (véase figura C.2).

Figura C.1: Modo *World Tour*

Figura C.2: *Super Hang On* para *Sega Megadrive*

Paralelamente, el jugador puede competir en este modo de juego de tres maneras posibles, atendiendo al tipo de vehículo conducido por los rivales. En base a este criterio, el jugador puede seleccionar jugar a *World Tour* en las siguientes submodalidades:

– *Classic mode*: en esta submodalidad los rivales conducen el tipo de vehículo propio del videojuego original en el que *World Tour* está inspirado. Dado que está basado

en el juego *Super Hang On*, y que en dicho videojuego es la moto el único vehículo jugable, todos los rivales conducen motos.

– *User mode*: en esta submodalidad los coches conducidos por los competidores son del mismo tipo que el vehículo escogido por el jugador.

– *Random mode*: en esta modalidad el tipo de vehículo que conducen los oponentes es totalmente aleatorio, por lo que los bólidos son de distintos tipos (véase figura C.3).

Figura C.3: Modo *World Tour* jugado en *Random Mode*

C.2. *Out Run*

Esta modalidad de juego tiene como objetivo pasar por cinco escenarios distintos, alcanzando los diferentes puntos de control, para conseguir más segundos de juego y lograr llegar a la meta antes de que el tiempo se acabe y con la mayor puntuación posible. A más velocidad, más puntos.

A lo largo de los distintos escenarios, el jugador debe intentar tanto de sortear los vehículos del tráfico que circulan en ese momento por la carretera como de no chocarse contra ningún obstáculo del escenario. El jugador alcanza la meta cuando consigue

llegar al final del quinto mapa. Pues al final de los cuatro anteriores escenarios, se le presenta al jugador una bifurcación por la que puede optar ir a la izquierda o la derecha. En función del camino que escoja pasará a un escenario u a otro. La bifurcación es el lugar donde reside el punto de control. De este modo, cuando la pasa, se incrementa el tiempo de juego.

Out Run tiene un total de 15 escenarios distintos, estructurados a modo de árbol. El escenario donde está la salida se identifica con el nodo raíz del árbol y los escenarios donde está la meta son los nodos hojas del árbol (véase figura C.4). La bifurcación, es por tanto, el elemento que une el nodo de un nivel con sus dos hijos en el nivel inferior.

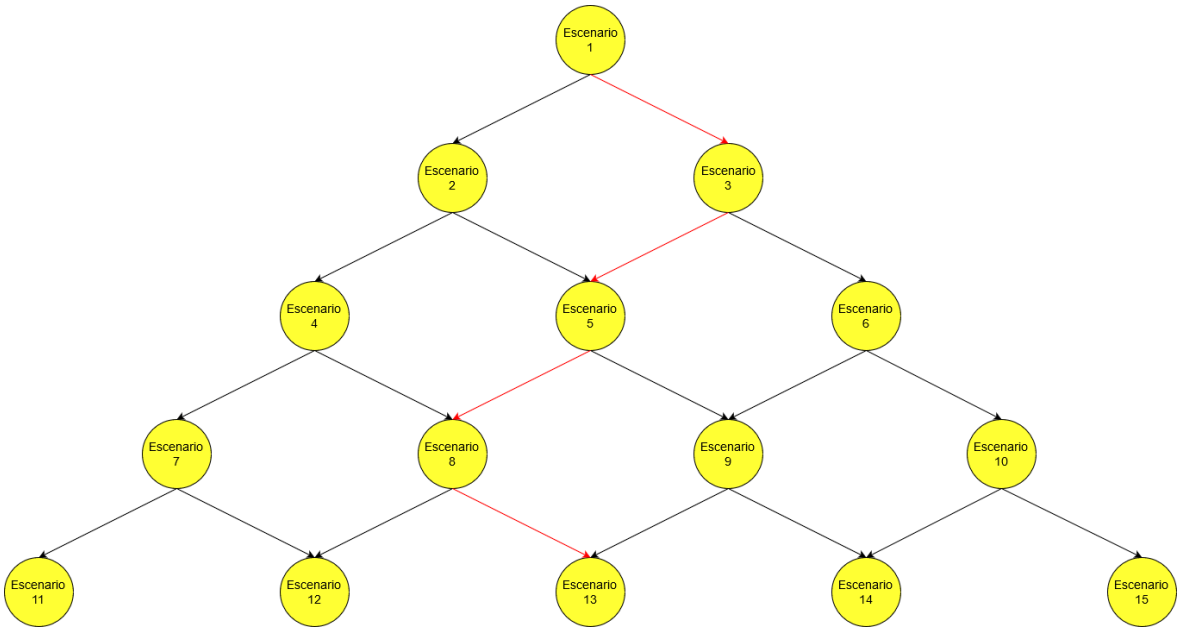


Figura C.4: Escenarios de *Out Run* con ruta posible marcada con flechas rojas

El sistema de puntuación es muy sencillo, es igual que en la versión para la recreativa de 1986. Existe un *ranking* que almacena los siete mejores jugadores ordenados por orden decreciente de puntuación. Para cada jugador se almacena su nombre, su puntuación y el tiempo que le ha llevado completar la partida, expresado en minutos, segundos y centésimas de segundo. El criterio es el orden de puntuación. De este modo, jugadores que tienen puntuaciones mayores irán por delante de los que tienen puntuaciones más bajas. El jugador con mayor ratio de puntos va en primera posición, y así sucesivamente. Cuando un jugador completa una partida, si dicho jugador ha batido un récord de puntuación y se ha colocado entre los siete mejores, se mostrarán los jugadores del *ranking* actuales junto con dicho jugador en la posición que le corresponde dentro de la clasificación, desechando así, al jugador que tenía la

menor puntuación de entre los anteriores. Además, se le solicitan sus credenciales, que serán guardadas en el *ranking* para futuras partidas. Seguidamente se ofrece una comparativa visual entre el modo *Out Run* de *Multi Race Driving* (véase figura C.5) y el videojuego *Out Run* de la recreativa de 1986 (véase figura C.6).

Figura C.5: Modo *Out Run*

Figura C.6: *Out Run* para la máquina recreativa

C.3. *Pole Position*

En esta modalidad, el jugador debe de forma previa seleccionar el circuito donde correr, cuántas vueltas dar y contra cuántos rivales correr. El objetivo del modo de juego consiste en dar vueltas al circuito escogido atravesando los puntos de control existentes antes de que se acabe el tiempo de juego para incrementarlo y quedar en primera posición venciendo a los rivales. Tras concluir la carrera, si el jugador no queda primero se le solicita si quiere volver a intentarlo.

Para este modo de juego hay un total de cuatro escenarios distintos, en los que se puede correr entre una y cinco vueltas, y con número variable de rivales que oscila desde cero, útil para vueltas de reconocimiento al circuito, hasta como máximo siete. Los posibles circuitos de *Pole Position* son *Poland*, *Fuji*, *Suzuka* y *Seaside*. El sistema de puntuación sigue el mismo esquema que el modo *World Tour*.

A continuación, se ofrece una comparativa entre el modo juego *Pole Position* de *Multi Race Driving* (véase figura C.7) y el juego original en el que se ha inspirado, *Pole Position 2*, lanzadado para *Atari* por *Namco* en 1982 (véase figura C.8).

Figura C.7: Modo *Pole Position*

Figura C.8: Versión *arcade* de *Pole Position 2* para *Atari*

Paralelamente, el jugador puede competir en este modo de juego de tres maneras posibles, atendiendo al tipo de vehículo conducido por los rivales. En base a este criterio, el jugador puede seleccionar jugar a *Pole Position* en las siguientes submodalidades:

- *Classic mode*: en esta submodalidad los rivales conducen el tipo de vehículo propio del videojuego original en el que *Pole Position* está inspirado. Dado que está basado en el juego *Pole Position 2*, y que en dicho videojuego es el coche de Formula 1 (*F1*) el único vehículo jugable, todos los rivales conducen coches de *F1*.

- *User mode*: en esta submodalidad los coches conducidos por los competidores son del mismo tipo que el vehículo escogido por el jugador.

- *Random mode*: en esta modalidad el tipo de vehículo que conducen los oponentes es totalmente aleatorio, por lo que los bólidos son de distintos tipos (véase figura C.9).

Figura C.9: Modo *Pole Position* jugado en *Random mode*

C.4. *Driving Fury*

En esta modalidad, el jugador debe detener a un vehículo criminal fugado colisionando contra él antes de que se acabe el tiempo de juego. El vehículo criminal parte a una cierta distancia del vehículo del jugador y con todo el nivel de vida completo. De este modo, por cada colisión, la vida del vehículo criminal va descendiendo. El número de golpes a dar depende de la dificultad. Esta modalidad también cuenta con 15 escenarios, totalmente diferentes, estructurados en árbol. El jugador debe ir conduciendo y tratar de llegar a los puntos de control, cruzando las bifurcaciones y aumentando el tiempo de juego, al mismo tiempo que trata de detener al criminal.

Si el jugador se queda sin tiempo o llega a la meta sin detener al rival pierde la partida y vuelve a repetir el nivel. Si logra detenerlo pasa al siguiente nivel. Consta de un total de tres niveles. Cuando el jugador completa un nivel, se regresa al mapa inicial y se vuelve a perseguir a otro vehículo. La novedad estriba en que el jugador puede escoger rutas nuevas para explorar, y además, que el vehículo criminal aparece un escenario más cerca de la meta. Así, el jugador tiene menos espacio para poder detenerlo. El sistema de puntuación es idéntico al de *World Tour* y *Pole Position*. A continuación, se muestra una comparativa entre el modo Driving Fury de *Multi Race Driving* y el videojuego original en el que se ha inspirado, el *Chase HQ* para la máquina *arcade* lanzado en 1988 (véase figura C.10).

(a) Modo *Driving Fury* (b) *Chase HQ* para la máquina recreativa

Figura C.10: Comparativa entre*Driving Fury* y *CHase HQ*

C.5. *Demarrage*

Es el único modo de juego que está inspirado en un videojuego reciente. El modo *Demarrage* de *Multi Race Driivng* (véase figura C.11) ha sido inspirado en un submodo de juego de *Need For Speed Undercover*, lanzado para *PS2* en 2008 (véase figura C.12). El esquema de juego es muy parecido al de *Driving Fury* con la diferencia de que tras encontrar al vehículo objetivo, en lugar de golpearle hasta detenerlo, el jugador debe intentar adelantarlo y colocarse por delante de él una distancia igual o superior a los 400 metros, haciéndole demarraje (arrancada o acelerón brusco que realiza un corredor o un vehículo a motor para dejar atrás a sus contrincantes). Si el jugador se queda sin tiempo o se alcanza la meta, el vehículo logra evadirse y se pierde la partida. Al igual que los modos de juego *Out Run* y *Driving Fury*, este modo cuenta con otros 15 escenarios distintos, estructurados de la misma manera.

En esta modalidad, el tránsito vial y la resistencia del vehículo a ser adelantado son las adversidades a las que el jugador debe enfrentarse. Consta de igual forma que el modo *Driving Fury* de tres niveles en los que el vehículo a lograr hacer demarraje aparece cada vez más cerca de la meta, teniendo así, menos espacio para poder hacerlo. El sistema de puntuación es idéntico al de *World Tour*, *Pole Position* y *Driving Fury*.

Figura C.11: Modo *Demarrage*

Figura C.12: Modo *Demarrage* de *Need ForSpeed Undercover*

Anexo D

Manual de usuario

Este anexo proporciona una guía de comunicación técnica destinada a dar asistencia a los jugadores. Se describen los menús de la *GUI* y qué acciones puede realizar el jugador en cada uno. Además, también se detallan las animaciones y las *HUD's* de los modos de juego que componen la arquitectura de *Multi Race Driving*.

D.1. Menú de presentación

El programa arranca mostrando al jugador una animación donde figuran el logotipo y nombre de la distribuidora, y la marca comercial de *Multi Race Driving* (véanse figuras D.1 y D.2). Dichos nombres son ficticios y se han creado para poder representar al desarrollador. Esta animación no puede saltarse, es decir, debe ser vista obligatoriamente.

Figura D.1: Logotipo de la empresa distribuidora

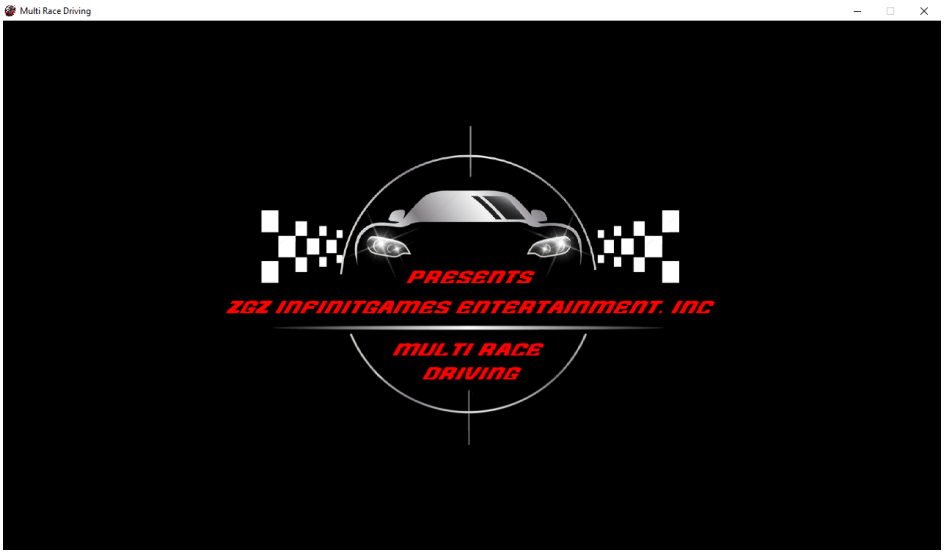


Figura D.2: Marca comercial

Tras concluir esta animación se muestra el menú de presentación de *Multi Race Driving* (véase figura D.3). Presionando *START* se avanza al siguiente menú y presionando *ESC* se cierra el programa. Por último, presionando *SPACE* se avanza al menú de créditos.

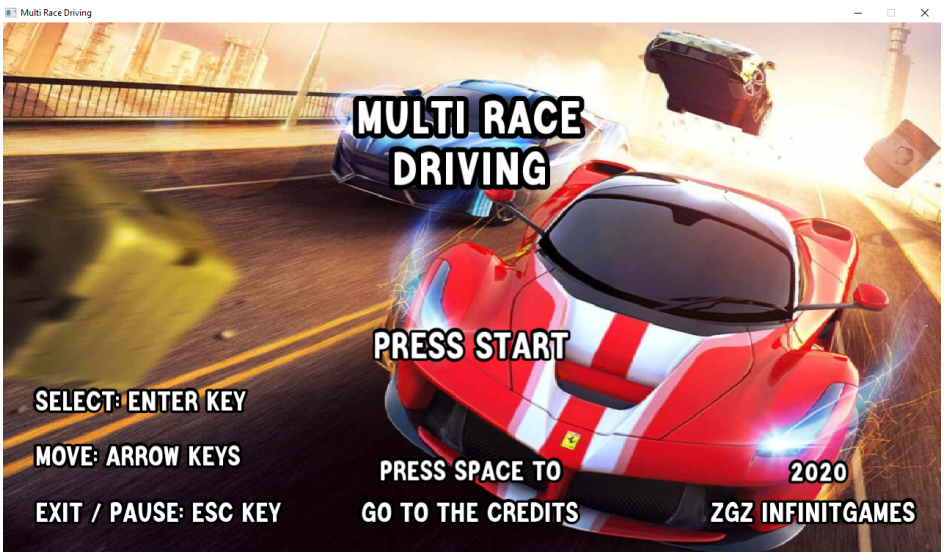


Figura D.3: Menú de presentación

D.2. Menú principal

Este menú presenta a los jugadores las opciones para jugar tanto al modo de un jugador como al modo multijugador. Además, presenta la opción para poder acceder a la sección de ajustes y configuración (véase figura D.4). Haciendo uso de las flechas

del cursor *UP* y *DOWN* el jugador puede moverse por los distintos botones del menú. Si se presiona *ESC* se retorna al menú principal. Si se pulsa *START*, en función del botón seleccionado, se avanza al menú de los modos de juego de un solo jugador, al menú de multijugador en red o al menú de opciones.

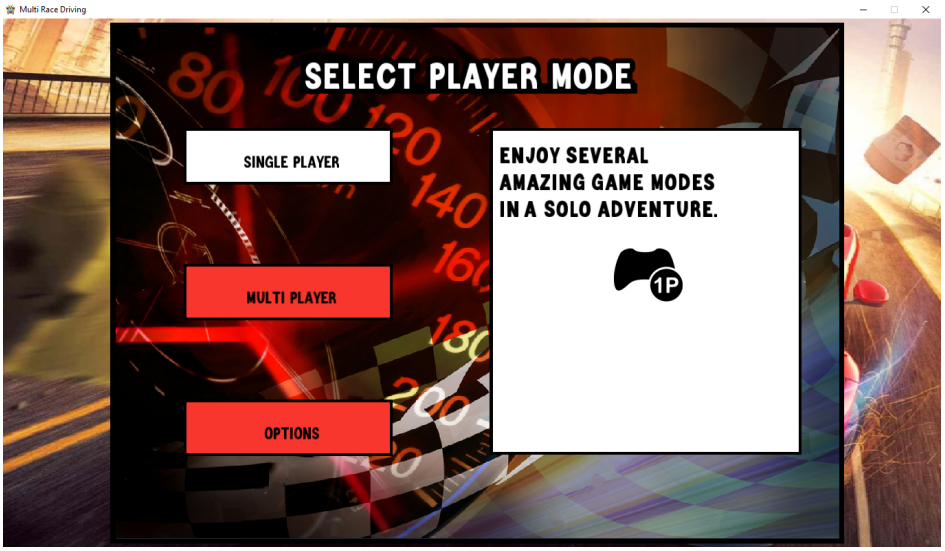


Figura D.4: Menú principal

D.3. Opciones

Este menú puede ser accesible tanto desde el menú principal teniendo marcada la opción de *OPTIONS* y pulsando *START*, o desde el menú de pausa seleccionando el botón de *OPTIONS* y presionando *START*. En este menú de configuración es donde se lleva a cabo la gestión de todos los parámetros de configuración. Las flechas del cursor *UP* y *DOWN* permiten al jugador moverse entre las distintas opciones de configuración.

Al entrar en el menú de opciones, por defecto, siempre se encuentra marcada la opción de dificultad. Usando las flechas del cursor LEFT y RIGHT el usuario puede seleccionar el modo de dificultad que más le guste (fácil, normal y difícil).

Las tres opciones siguientes representan menús para cambiar el ajuste de las bandas musicales y efectos de sonido, de los gráficos (resolución de pantalla y *pixel art*) y los controles (véase figura D.5).

A diferencia de las opciones anteriores, para poder acceder a los submenús que permiten modificar estos parámetros se debe pulsar la tecla *START*.

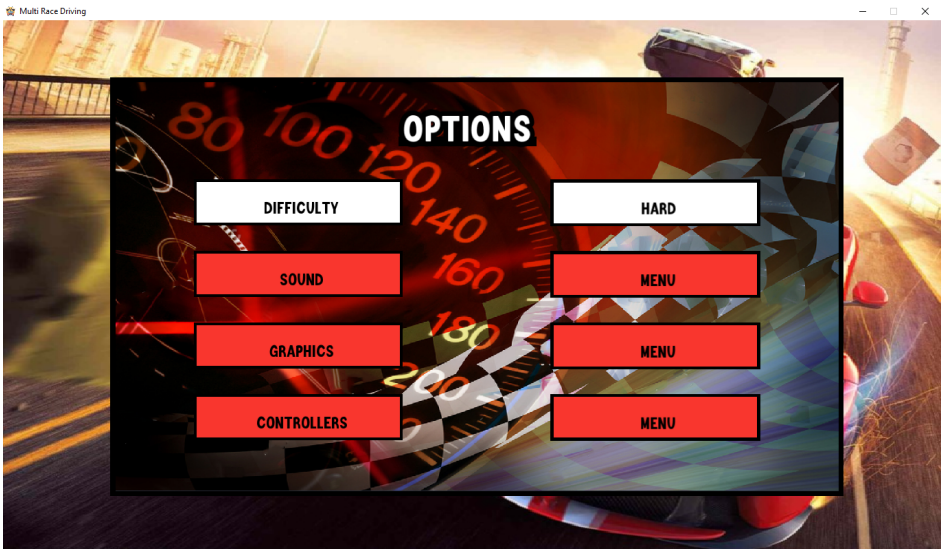


Figura D.5: Opciones

D.3.1. Ajustes de sonido

Este menú de la interfaz solamente puede ser accesible desde el menú de opciones si y solo si la opción de *SOUND* está marcada. Al presionar *START*, aparece un submenú en el que el jugador puede modificar dos posibles parámetros, el volumen de efectos y el volumen de bandas musicales (véase figura D.6). Usando las teclas del cursor *LEFT* y *RIGHT* se determina el volumen de efectos y de música con los que jugar.

Una funcionalidad interesante que se ha incluído en este punto ha sido la opción de que los volúmenes se incrementen o decrementen de uno en uno con el fin de que el jugador tenga un mayor control de los rangos de valores permitidos. Además, el volumen es ajustado de manera automática en directo, es decir, mientras el jugador modifica ambos parámetros, se van reproduciendo los sonidos con la nueva magnitud de volumen. Una vez que el jugador está conforme con los valores seleccionados, pulsando *ESC* vuelve al menú de opciones con la nueva configuración de sonido guardada.

D.3.2. Ajustes de gráficos

El menú de ajuste de gráficos solo es accesible desde el menú de opciones teniendo marcada la opción *GRAPHICS*. Al presionar *START*, aparece un submenú en el que el jugador puede modificar dos posibles parámetros, la resolución de la pantalla y

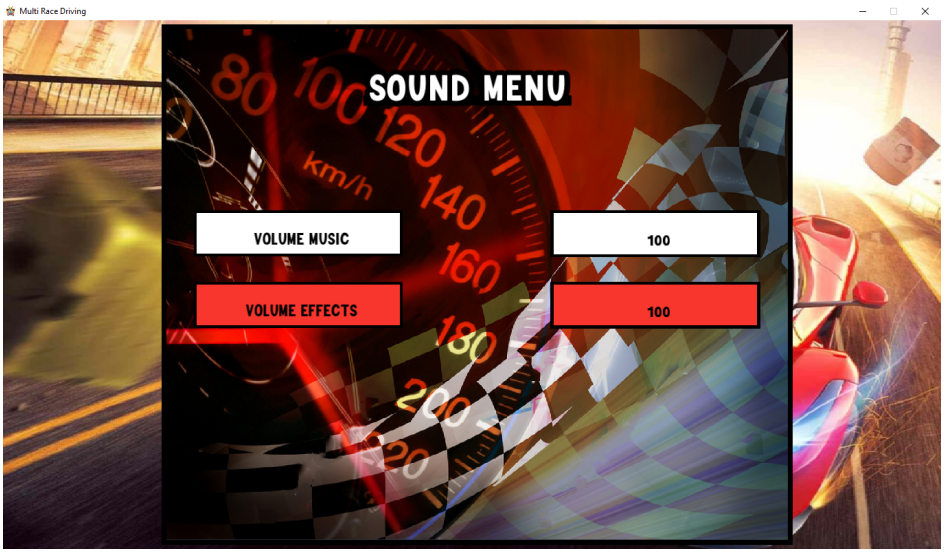


Figura D.6: Ajustes de sonido

la utilización del *pixel art* (véase figura D.7). Usando las teclas del cursor *LEFT* y *RIGHT* el jugador puede modificar ambos parámetros. Cabe destacar que existen una gran cantidad de resoluciones. Por ello, se han escogido las más comunes (pantalla completa, 921 x 691, 1280 x 720, 1366 x 768, 1920 x 1080, 2560 x 1440 y 3640 x 2160).

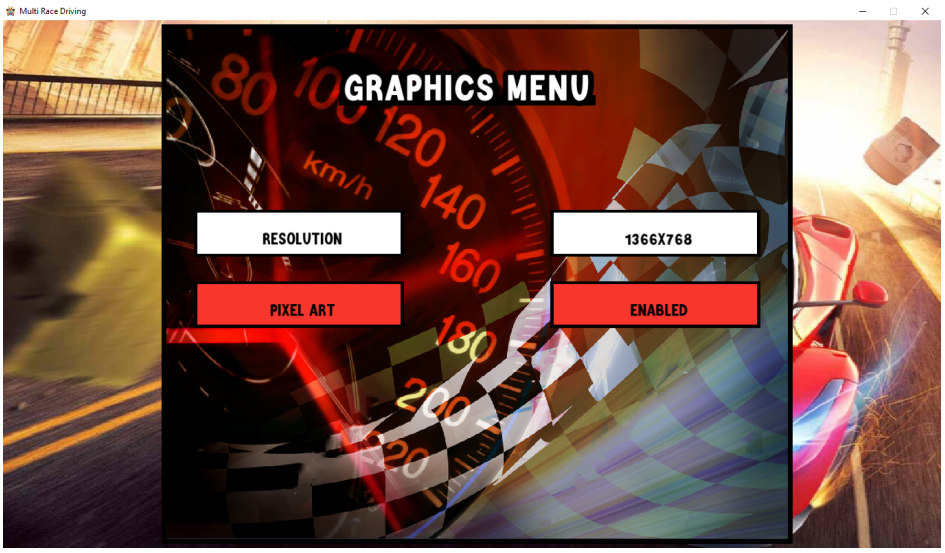


Figura D.7: Ajustes de gráficos

Las resoluciones están en píxeles. Las tres últimas han sido pensadas para situaciones en las que se desea jugar en dispositivos con un mayor grado de resolución que las pantallas de los computadores normales. Por ejemplo, son muy prácticas en casos en los que se desea conectar el computador a una *TV* (véase figura D.8). Una vez que el jugador está conforme con los valores seleccionados, pulsando *ESC* vuelve al menú de opciones con la nueva configuración de gráficos.

Figura D.8: *Multi Race Driving* jugado en *TV* en pantalla completa

D.3.3. Configuración de controles

Esta pantalla es accedida desde el menú de opciones. Para ello, el jugador debe tener seleccionada la opción *CONTROLLERS* y presionar *START*. Tras dicha acción se abre un nuevo submenú en el que el jugador puede determinar los controles para acelerar, frenar, girar el vehículo tanto a la izquierda como a la derecha, y controlar el cambio de pista musical en partida (véase figura D.9). Para poder optar a cambiar un parámetro de control se debe presionar *SPACE*. Seguidamente se debe presionar la tecla que se desea asignar para poder modificar el controlador.

Es importante resaltar que no se han hecho jugables todas las teclas del teclado, solamente aquellas que son reconocidas de acuerdo a la *API* de *SFML*. De este modo, cuando el jugador presiona una tecla pueden ocurrir uno de los siguientes tres casos:

– Si la tecla presionada es válida, es decir, es reconocida por *SFML* y es jugable, se emite un sonido de cambio para así poder alertar al jugador de que la nueva tecla ha sido aceptada.

– Si ha metido una tecla que no es conocida por la propia librería de *SFML*, como por ejemplo son las teclas *F1*.. *FN*, la tecla de impresión de pantalla, la tecla de mayúsculas, etcétera, no se emite ningún tipo de sonido. Si ocurre esta situación es porque las teclas que está metiendo el jugador están siendo ignoradas.

– Si ha metido una tecla que es inválida, osea que es reconocida por *SFML* pero no es jugable se emite un sonido de incorrección para alertar al jugador de que debe escoger otra.

Para poder visualizar el nuevo controlador, se debe soltar la tecla *SPACE*. Así, se confirma la nueva configuración y se actualizan los cambios. Si la tecla no se suelta, no se hacen visibles los nuevos cambios puesto que el controlador todavía está siendo editado por el jugador. Para salir del menú se debe presionar *ESC*, regresando al menú de opciones.

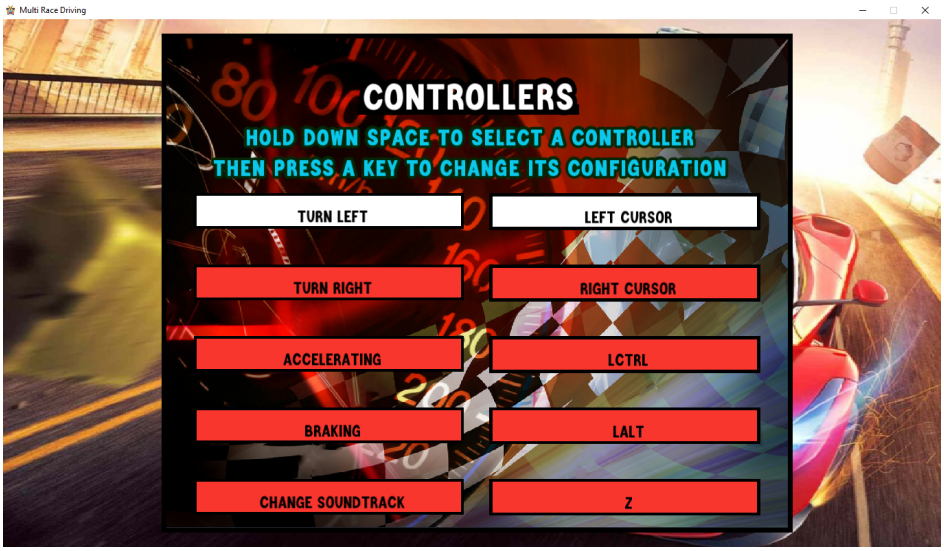


Figura D.9: Configuración de controles

D.4. Configuración del modo de un solo jugador

A esta modalidad de juego solamente se puede acceder desde el menú principal si el jugador presiona *START* teniendo marcada la opción *SINGLE PLAYER*. De este modo el jugador accede al menú de selección de juego para un solo jugador.

D.4.1. Selección de modo de juego para un jugador

Este menú presenta las diferentes modalidades de juego para un único jugador (véase figura D.10). Recurriendo al uso de las flechas del cursor *UP* y *DOWN* el jugador puede moverse por los distintos botones del menú. Si se pulsa *ESC* se regresa al menú principal. Al presionar *START*, si se tiene marcado el modo *Pole Position* o el modo *World Tour* se avanza al menú de selección de vehículo rival. Para los modos restantes se avanza al menú de selección de vehículos.

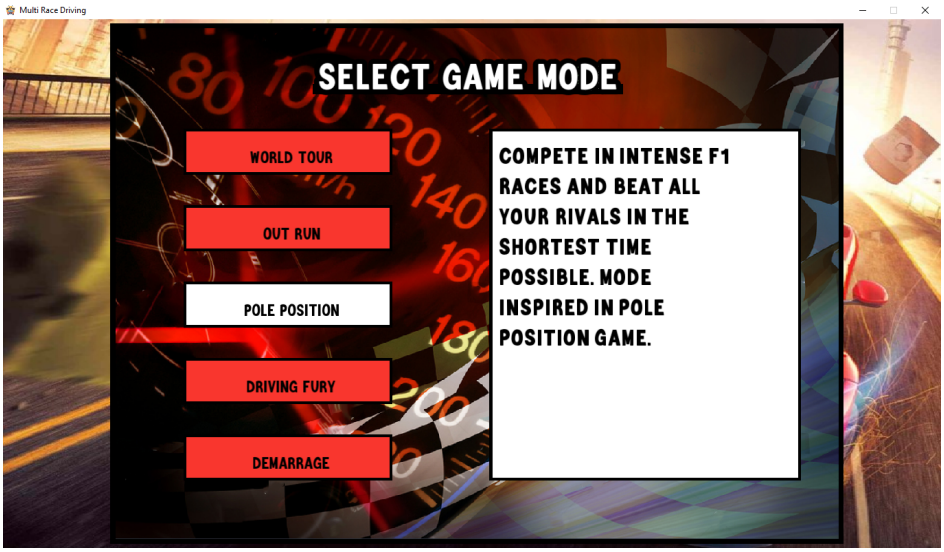


Figura D.10: Modos de juego disponibles para un jugador

D.4.2. Selección de vehículo rival

Este menú solo es accesible si el jugador selecciona jugar al modo *World Tour* o al modo *Pole Position*. En dicho menú el jugador selecciona la submodalidad en la que quiere jugar a dichos juegos. Estas submodalidades son *Classic mode*, *User mode*, *Random mode* (véase figura D.11). Haciendo uso de las flechas del cursor *UP* y *DOWN*, el jugador puede navegar por los distintos botones del menú y seleccionar presionando *START* de entre las opciones disponibles la que más desee. Tras seleccionar una de ellas presionando *START*, el jugador avanza al menú de selección de vehículos. Si por el contrario pulsa *ESC*, el jugador regresa al menú de selección de modalidad de juego para un jugador.

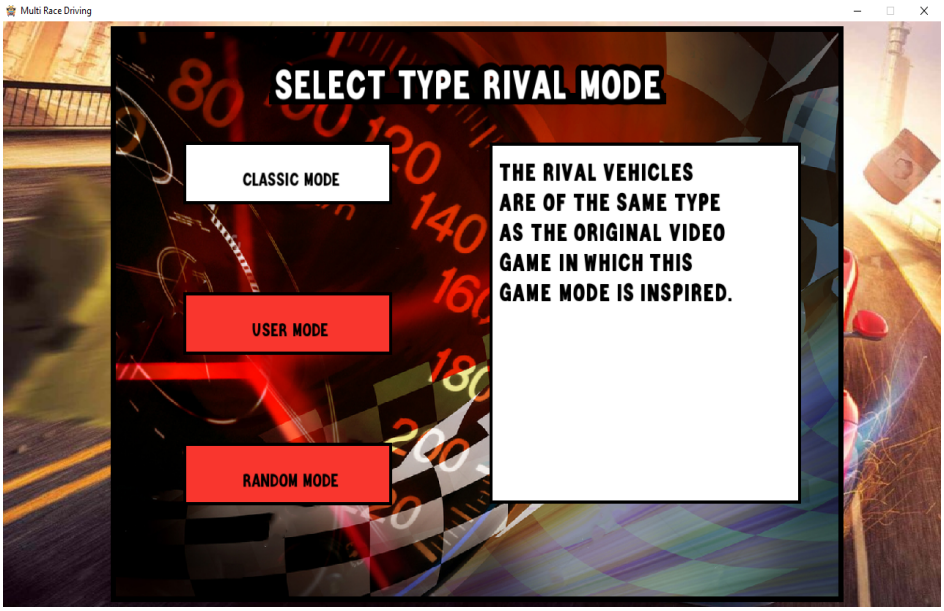


Figura D.11: Menú de selección de tipo de vehículo rival

D.5. Configuración del modo multijugador *online*

A esta modalidad de juego solamente se puede acceder desde el menú principal si el jugador presiona *START* teniendo marcada la opción *MULTI PLAYER*. De este modo, el jugador accede al menú de selección de testeo de conexión al servidor remoto.

D.5.1. Testeo de conexión al servidor remoto

Este es el primer menú que le aparece al jugador al entrar en el modo multijugador. En dicho menú se realiza un *test* de conexión al servidor remoto que contiene el *framework Linda*. Tras ejecutar el *test*, se muestran estadísticas sobre los tiempos de latencia (mínima, media y máxima) obtenidos, y mensaje informativo sobre cómo le va a resultar la experiencia de juego al jugador (véase figura D.12).

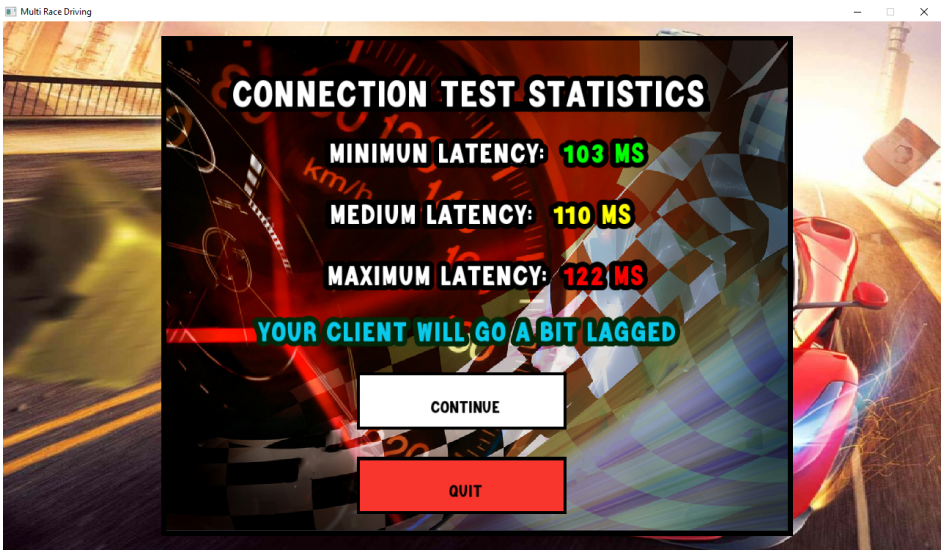


Figura D.12: Resultados del *test* de conexión al servidor remoto

Con las flechas del cursor *UP* y *DOWN*, el jugador navega por los botones del menú. Si pulsa *ESC* vuelve al menú principal. Si pulsa *START* avanza al menú de gestión de grupos si tiene marcada la opción *CONTINUE*, o regresa al menú principal si la opción marcada es *QUIT*. Durante la realización del *test* de conexión, el jugador visualiza una animación de espera (véase figura D.13), que termina en el mismo momento en el que el se muestran los resultados del *test*.

Figura D.13: Espera durante el *test* de conexión al servidor remoto

D.5.2. Gestión de grupos en modo multijugador

Este menú aparece cuando el jugador confirma su deseo de jugar tras ejecutarse el *test* de conexión. Con las flechas del cursor *UP* y *DOWN*, el jugador puede decidir si crear o unirse a un grupo (véase figura D.14). Si presiona *ESC* se vuelve al menú principal, no al menú de testeo de conexión al servidor remoto. Si presiona *START* avanzará al menú para introducir su apodo en el modo multijugador, independientemente de tener marcada la opción CREATE GROUP o JOIN GROUP.

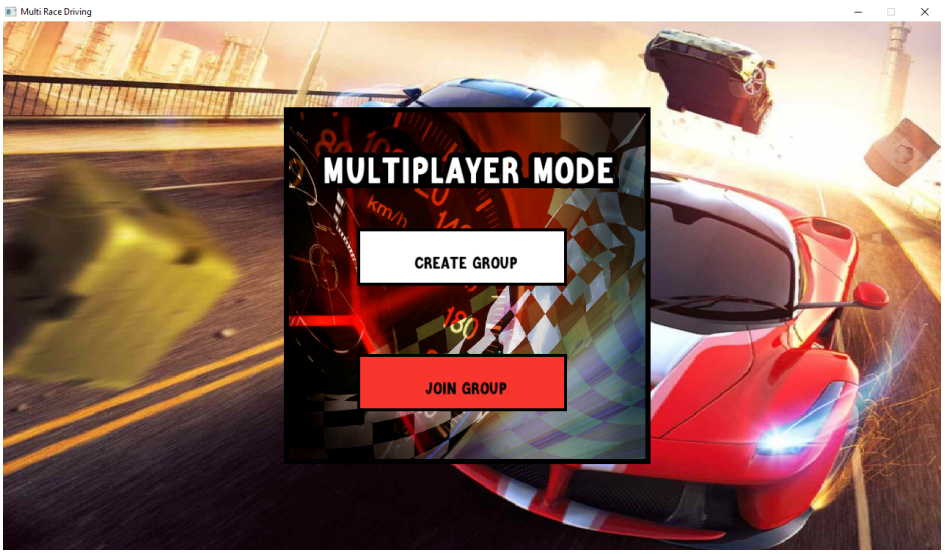


Figura D.14: Gestión de grupos en modo multijugador

D.5.3. Validación de nombre de jugador

En esta pantalla se le solicita al jugador que introduzca su apodo o alias para poder ser identificado en el modo multijugador. (véase figura D.15). A través de los caracteres alfanuméricos, el jugador debe introducir un nombre de como máximo ocho caracteres.

Si el jugador pulsa *ESC* retorna al menú de gestión de grupos, pero si presiona *START*, se guardará el nombre hasta entonces introducido y se comprobará si es válido, es decir, si no hay otro jugador con ese mismo alias. Si el apodo es válido se avanza al siguiente menú, pantalla de validación de nombre de grupo o pantalla de modo de unión a grupo según el jugador halla escogido crear o unirse a grupo, respectivamente. En caso contrario se vuelve a solicitar el nombre. Paralelamente, el jugador puede presionar la tecla BACKSPACE para eliminar letras del nombre en caso de equivocarse al teclear.

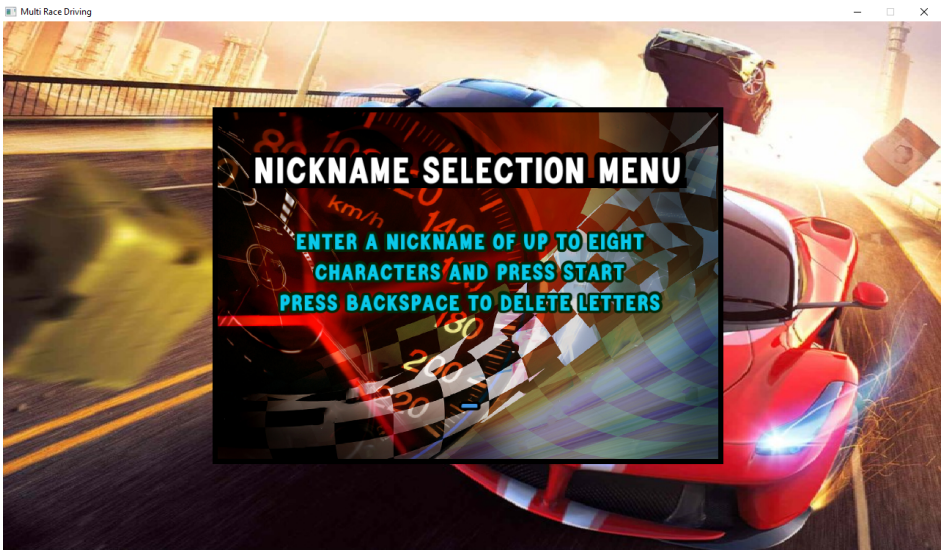


Figura D.15: Validación de apodo de jugador

D.5.4. Validación de nombre de grupo

Esta pantalla solo les aparece a los jugadores propietarios de grupos (aquellos que han marcado la opción de crear grupo) o a los usuarios que han decidido unirse a un grupo particular (véase figura D.16). El nombre del grupo se introduce y valida igual que el del jugador. Se deben introducir como máximo ocho caracteres alfanuméricos. Si se presiona *START*, se comprueba el apodo del grupo, si existe se solicita de nuevo el nombre y si no, se avanza al menú de plantilla de grupo. Haciendo uso de la tecla *BACKSPACE*, el jugador puede eliminar letras en caso de equivocarse al teclear.

Figura D.16: Validación de grupo

D.5.5. Modo de unión a grupo

Este menú les aparace a los jugadores que han marcado la opción de unirse a grupo. Usando las flechas del cursor *UP* y *DOWN* pueden decidir unirse a un grupo particular, para lo que posteriormente, deberán validar el nombre, o unirse a un grupo aleatorio (véase figura D.17). Si el jugador escogió unirse a un grupo aleatorio, el cliente y el servidor se encargan de que el emparejado funcione correctamente. Si el jugador pulsa *ESC* regresará al menú de validación de apodo de jugador. Si presiona *START*, será automáticamente introducido en un grupo si tiene marcada la opción *RANDOM GROUP* o se avanzará al menú de validación de nombre de grupo si tiene marcada la opción *PARTICULAR GROUP*.

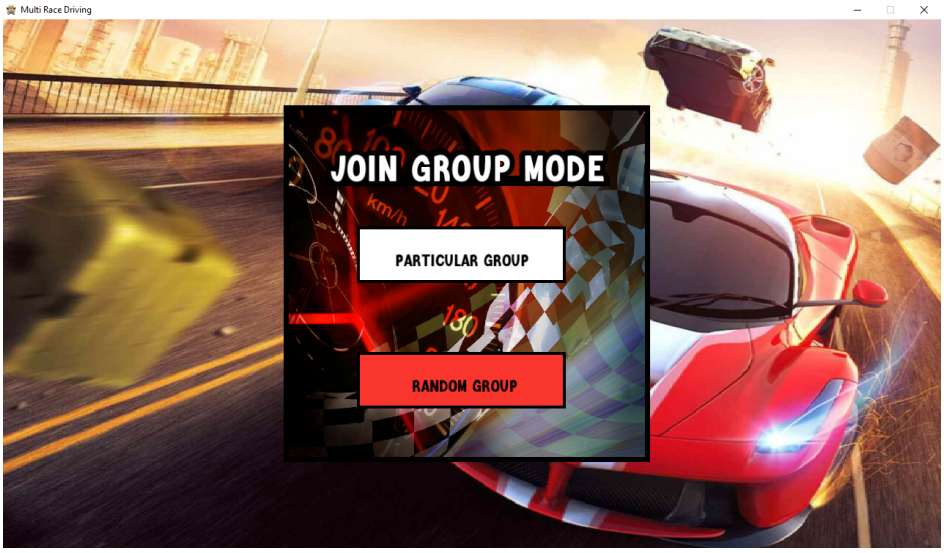


Figura D.17: Modo de unión a grupo

D.5.6. Plantilla de grupo

Este menú le aparece al jugador después de que la creación o unión a un grupo, ya sea aleatorio o particular, ha resultado un éxito. Si el jugador es el propietario del grupo, puede pulsar *ESC* o *START*. Los miembros del equipo se muestran de manera que en la primera posición se encuentra el propietario del grupo, y en las posiciones sucesivas, el resto de jugadores invitados al grupo por orden de validación de petición (véanse figuras D.18 y D.19).

Figura D.18: Vista de los miembros de grupo para el propietario del grupo

Figura D.19: Vista de los miembros de grupo para un invitado al grupo

Si el propietario pulsa *ESC*, vuelve al menú anterior y notifica a todos los integrantes, si los hay, que el grupo se cancela. Por el contrario, si pulsa *START* o el grupo se llena por completo, se notifica a todos los miembros que el grupo está formado y se avanza al menú de selección de modo de juego. Los jugadores invitados permanecerán en este menú hasta que el propietario confirme o cancele el grupo. Si pulsan *ESC* cancelarán su pertenencia al grupo y volverán al menú anterior siempre y cuándo su petición todavía no haya sido aceptada por el propietario.

D.5.7. Selección de modo de juego multijugador

Este menú aparece tras cerrarse o completarse un grupo. solo le aparece al jugador que es propietario del grupo. Es en este menú donde selecciona a qué modalidad de juego jugar (véase figura D.20). Haciendo uso de las flechas del cursor *UP* y *DOWN*, se marca cualquiera de los modos disponibles de juego, *World Tour* o *Pole Position*.

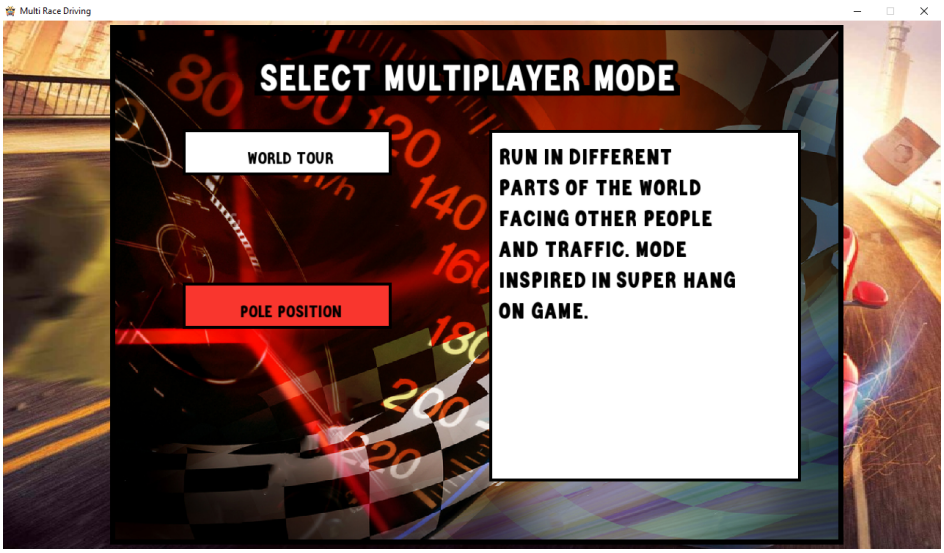


Figura D.20: Selección de modo de juego en modo multijugador

Presionando *START* se selecciona el modo en cuestión. Si se selecciona el modo *World Tour*, propietario e invitados avanzan al menú de selección de vehículos. Si se selecciona *Pole Position*, el propietario avanza al menú de selección de circuitos y los invitados esperan. Si pulsa *ESC* se deshace el grupo. En tal caso, el propietario regresa al menú de validación de nombre de grupo. Los invitados, por su parte, si se unieron a un grupo validando un alias determinado regresan al menú de validación de nombre de grupo, pero si se unieron aleatoriamente retornan al menú de modo de unión a grupo.

Mientras el propietario selecciona el modo de juego, los invitados visualizan una animación de espera (véase figura D.21) que termina cuando el modo de juego al que se va a jugar (*World Tour* o *Pole Position*), es seleccionado por el propietario del grupo.

Figura D.21: Espera de selección de modo multijugador

D.6. Selección de circuitos

Este menú aparece tanto en la configuración del modo para un solo jugador como en la configuración del modo multijugador cuando el modo de juego seleccionado es *Pole Position*. En ambos menús, con pequeñas variaciones, se selecciona en qué circuito correr y cuántas vueltas se va a dar. Si se va a jugar al *Pole Position* para un solo jugador, además del circuito y de las vueltas a dar, se puede configurar contra cuántos rivales se va a competir. En el modo multijugador este factor no se puede configurar porque los rivales son los miembros del grupo (véanse figuras D.22 y D.23).

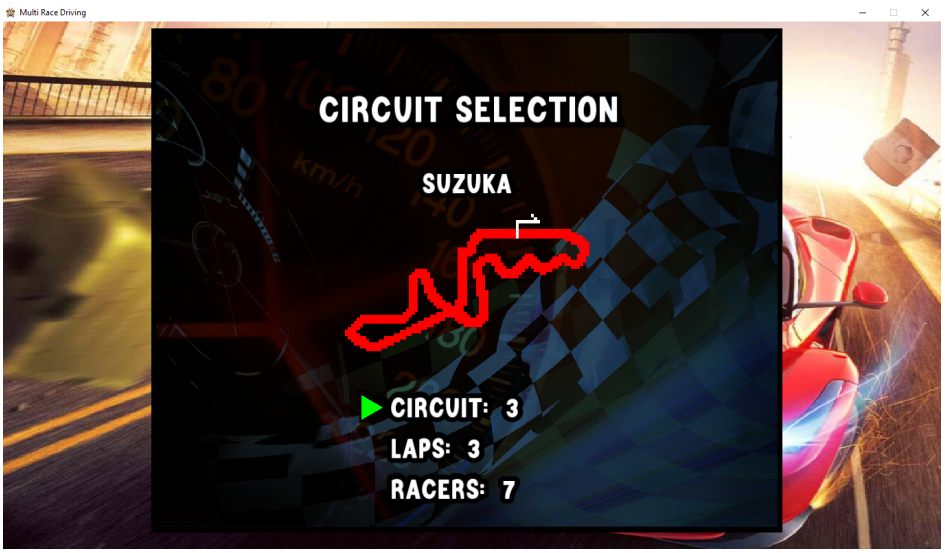


Figura D.22: Selección de circuito en modo de un solo jugador

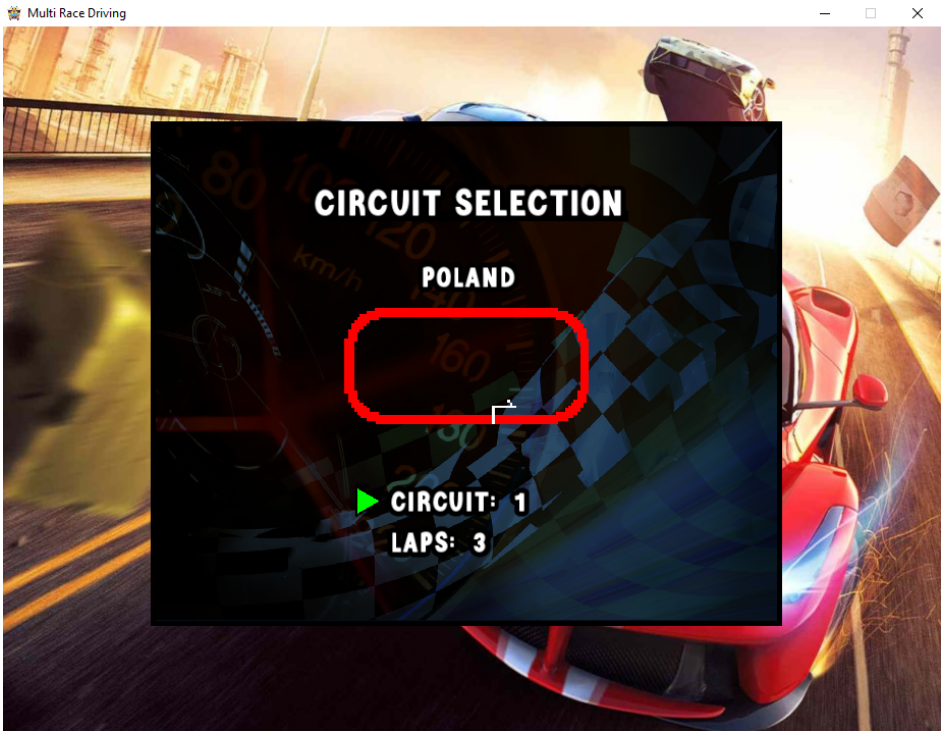


Figura D.23: Selección de circuito en modo multijugador

En modo multijugador, mientras el propietario selecciona el circuito donde correr, los invitados visualizan una animación de espera (véase figura D.24) que termina cuando el circuito en cuestión es seleccionado por el propietario del grupo.

Figura D.24: Espera de selección de circuito en modo multijugador

Con las flechas del cursor *UP* y *DOWN*, se pueden ajustar las diferentes opciones de configuración (circuito, vueltas y número de rivales). Con las teclas del cursor *LEFT* y *RIGHT* el jugador puede moverse por las distintas opciones disponibles.

Si el jugador presiona *ESC*, y no está en modo multijugador, regresa al menú de selección de modos de juego para un único jugador. Si el jugador pulsa *ESC* estando en modo multijugador, si es el propietario regresa al menú de selección de modo de juego multijugador. Si es un invitado, abandona el grupo, y regresa al menú de modo de unión a grupo. Si se pulsa *START* se fija el circuito donde correr con todos los parámetros de configuración seleccionados y se pasa al menú de selección de vehículos.

D.7. Selección de vehículos

Este menú es accesible en todos los modos de juego de un solo jugador y en el modo multijugador. En esta pantalla se muestran todos los vehículos disponibles para cada modelo juego. Haciendo uso de las flechas del cursor *LEFT* o *RIGHT* el jugador puede ver los distintos tipos de vehículos disponibles, y con las flechas del cursor *UP* y *DOWN*, puede cambiarles el color (véase figura D.25).

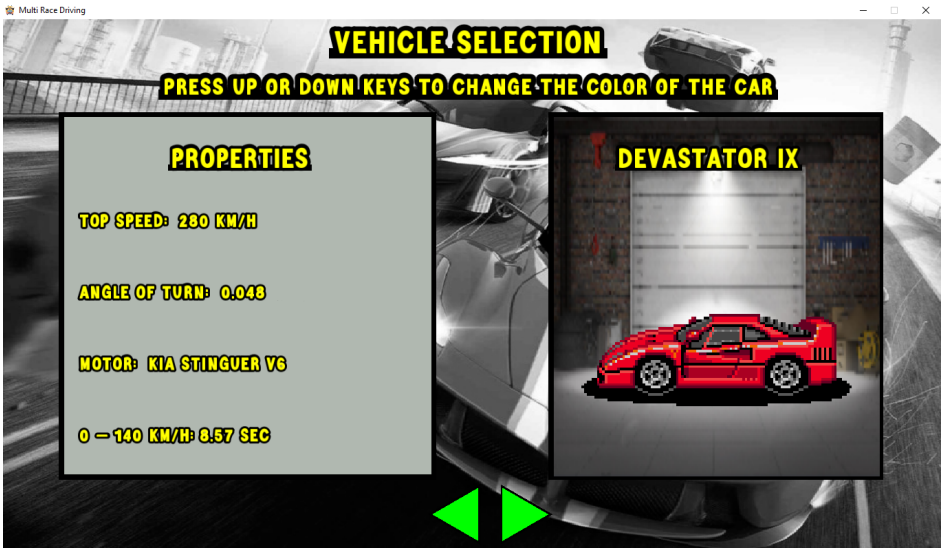


Figura D.25: Selección de vehículos

Si se presiona *ESC* en modo de un jugador, se vuelve según el modo de juego o al menú de selección de circuitos (*Pole Position*) o al menú de selección de modos de juego de un solo jugador (*World Tour*). Si se pulsa *ESC* en modo multijugador y no se ha seleccionado vehículo, si es el propietario regresa al menú de selección de modo de juego multijugador. Si es un invitado, abandona el grupo, y regresa al menú de modo de unión a grupo. Si se presiona *START* se selecciona el vehículo marcado, y si se está en modo multijugador, se espera a que el resto de jugadores seleccionen el suyo.

Si a la hora de cambiar el tipo de bólido, alguno de los triángulos inferiores centrales se vuelve rojo, se indica que no hay más vehículos si se pulsa esa tecla de nuevo. Tras seleccionar vehículo, si se está jugando al modo multijugador el participante deberá esperar a que el resto de integrantes también escojan bólido.

Tras seleccionar vehículo, ya sea en modo de un solo jugador o en modo multijugador con todos los rivales ya habiendo elegido, al jugador se le muestra una animación de carga de partida. Esta pantalla es esencial ya que muestra a los jugadores cuáles son los controles actuales (véase figura D.26). La razón por la que se pusó esta animación es porque los controles solo podían verse al acceder al menú de configuración de controles desde el menú de opciones. De esta forma, los jugadores que aceptasen la configuración por defecto comenzarían a jugar sin saber como manejarlo, lo que supondría perder tiempo de juego averiguando las teclas. Con esta animación se logra que el jugador conozca antes de comenzar a jugar los controles de juego.

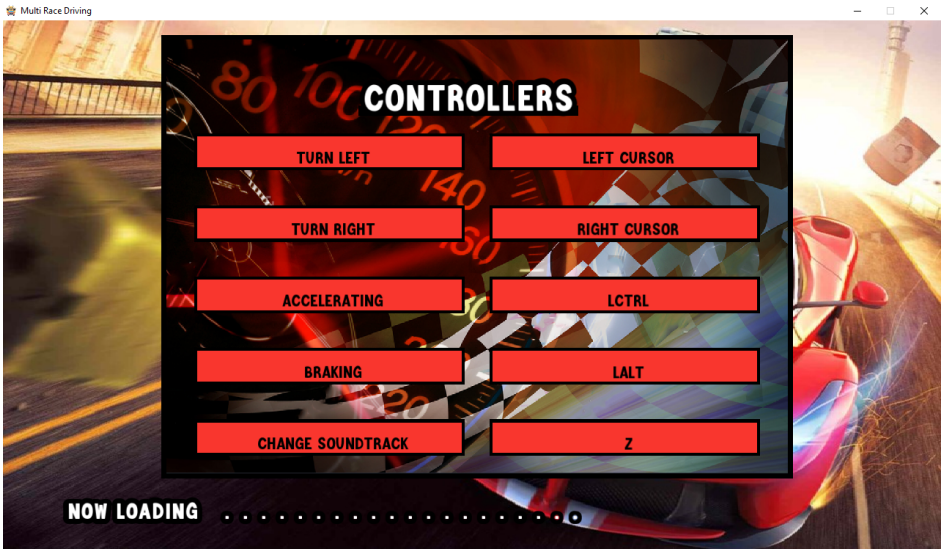


Figura D.26: Pantalla de carga de partida

D.8. Desarrollo de partida

Esta fase abarca todo el núcleo de acción que el jugador observa durante el transcurso de la partida, la cuál se puede dividir en los siguientes fragmentos.

D.8.1. Inicio de partida

Las partidas comienzan con unas animaciones que sirven como marco introductorio de los eventos de competición. Estas animaciones dependen en gran medida de la modalidad de juego seleccionada por el usuario ya que las temáticas de los modos de juego son muy diferentes y los elementos a visualizar (*HUD's*, tráfico, rivales, etcétera), de igual modo, también lo son.

En los modos *World Tour* y *Pole Position* estas animaciones se emplean para posicionar los corredores en la parrilla de salida, mostrar la *HUD* de estos modos de juego y para dar la salida al jugador (véase figura D.27).

Figura D.27: Salida del modo *World Tour*

En los modos *Out Run* y *Driving Fury* se coloca el bólido escogido por el jugador en posición de salida, se muestran las *HUD's* correspondientes, y se le informa al jugador de cuando comenzar a jugar.

En el modo *Demarrage* se coloca el vehículo del jugador en posición de salida, se muestra la *HUD* del modo de juego, se le informa de cuánta distancia debe ponerse por delante del vehículo objetivo y, por último, se le da la salida (véase figura D.28).

Figura D.28: Salida del modo *Demarrage*

D.8.2. Transcurso de partida

Durante la partidas, existen dos factores importantes que el jugador puede visualizar, las *HUD's* y el menú de pausa.

D.8.2.1. *HUD's* de los modos de juego

Las interfaces de juego se han definido como el conjunto de elementos visibles para el jugador mientras se encuentra en transcurso de una partida. Por tanto, las interfaces de juego son, en útima instancia, *HUD's* (*Head-Up Display*).

Las *HUD's* han sido diseñadas de manera acorde tanto al estilo (temática) de los modo de juego como al de los gráficos diseñados, Es muy importante mantener la consonancia de estilos para no confundir a los jugadores. Están formadas por los elementos necesarios para que los jugadores puedan jugar cómodamente a los modos de juego (véanse figuras D.29, D.30, D.31, D.32 y D.33). Además, los indicadores que las constituyen han sido colocados en posiciones estratégicas de las pantalla para no causar problemas de visibilidad de juego al jugador.

Figura D.29: *HUD* del modo *World Tour*

Figura D.30: *HUD* del modo *Out Run*

Figura D.31: *HUD* del modo *Pole Position*

Figura D.32: *HUD* del modo *Driving Fury*

Figura D.33: *HUD* del modo *Demarrage*

Las interfaces de los modos *World Tour* y *Pole Position* en modo multijugador son muy semejantes a los de modo de un solo jugador, con la excepción de que no aparece el indicador de tiempo de juego.

Los elementos correspondientes a las *HUD*'s de los modos de juego pueden encontrarse en el directorio *Hud*, localizado dentro de la carpeta principal *Data*. Los componentes que forman las interfaces de juego se hallan identificados con números correlativos para facilitar su procesamiento.

En vista de las figuras anteriores se puede observar que existe un contador de tiempo en la esquina superior izquierda que contabiliza la duración de la partida (visible en todos los modos de juego incluido el modo multijugador). Paralelamente, existe otro contador, localizado en la parte superior central de la pantalla que contabiliza el tiempo de juego del que dispone el jugador (solo visible en los modos de juego para un jugador).

La gestión de estos contadores de tiempo, así como del contador de puntos, solo presente en el modo *Out Run*, se ha efectuado por medio de *multithreading*. De este modo, para cada contador, tanto de tiempo como de puntos, existe un proceso paralelo que se encarga de controlarlo.

La razón por la que estas tareas se realizan aplicando *multithreading* es la siguiente. El hilo de ejecución principal tiene una carga de trabajo considerable puesto que tiene que monitorizar y controlar todo lo que el jugador ve en pantalla. Los contadores comienzan al inicio de los eventos de competición siempre a cero, además, cada contador tiene su propio intervalo de medida. El contador de la esquina superior izquierda cuenta centésimas de segundo y el central contabiliza segundos. De este modo, cuando pasa una centésima y un segundo de tiempo, respectivamente, ambos contadores se actualizan.

Estas comprobaciones se hacen de manera sucesiva y cada muy poco tiempo. Es muy difícil contar segundos exactos, y mucho más las centésimas de segundo porque es una magnitud temporal mucho más pequeña. Cuanto más pequeño es el intervalo de tiempo transcurrido entre mediciones, más exactas son. Si no se hiciese así, se contaría menos tiempo del que en verdad ha pasado, desencadenando un error de precisión acumulativo que provocaría un desfase de tiempo entre ambos contadores, es decir, registrarían los segundos de manera diferente, lo cuál es un error. Delegando estas tareas a hilos de ejecución a parte, se logra que estos problemas no ocurran. Además, si se juega al modo *Out Run*, existe un contador de puntos que se localiza en la esquina superior derecha de la interfaz. Este contador también es gestionado por un hilo aparte. De este modo se ha conseguido que los puntos se incrementen a un ritmo muy parecido a como lo hacían en los juegos *arcade* antiguos.

D.8.2.2. Pausa

Esta pantalla muestra el menú de pausa del jugador cuando está en transcurso de una partida y pulsa la tecla *ESC* si y solo si no está en modo multijugador. Si se pulsa *START* estando seleccionado el botón *RESUME* se reanuda la partida, si está marcado el botón *OPTIONS* se accede al menú de opciones, si se halla seleccionado el botón *HOME* se vuelve al menú de presentación y estando marcado el botón *QUIT* el programa se cierra.

La razón por la que se ha suprimido dicho menú en el modo multijugador es que esta modalidad necesita ser jugada en tiempo real. Si un jugador pausase el juego fastidiaría a todos los demás que están jugando en partida con él, sobre todo, si hubiesen muchos jugadores en partida. Por consiguiente, para evitar este posible problema se ha decidido eliminarlo.

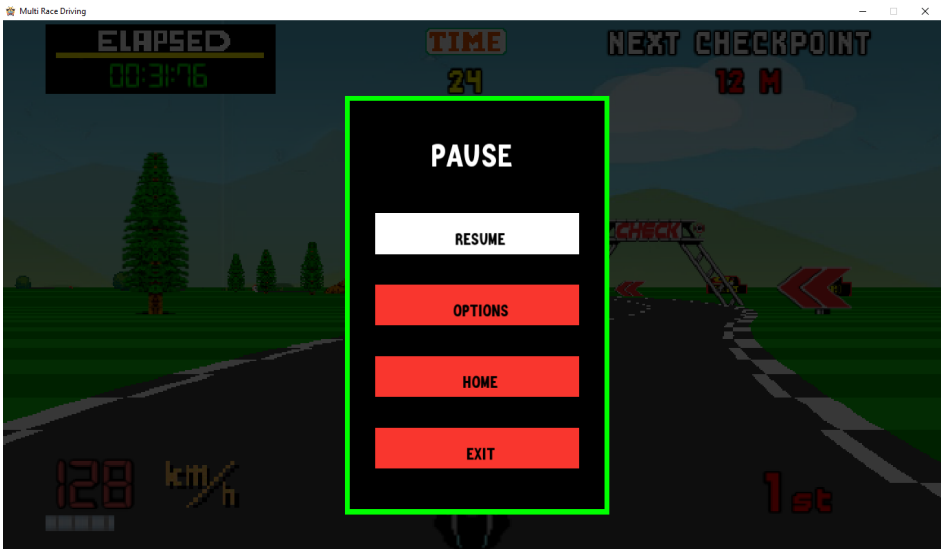


Figura D.34: Pausa

D.8.3. Final de carrera

Del mismo modo que el comienzo, las carreras también concluyen con animaciones que varían según el tipo de juego al que se está jugando. Estas animaciones se pueden clasificar en dos grupos, las de estadísticas de nivel y las de fin de modo de juego.

D.8.3.1. Estadísticas de nivel

Estas animaciones son visualizadas por el jugador en todos lo modos de juego a excepción del *Out Run*, concretamente, después de haber completado un nivel, si se trata de los modos *Driving Fury* o *Demarrage*, o después de completar una carrera si se trata de los modos *World Tour* o *Pole Position*.

Al terminar estas animaciones, se evalúa si el jugador ha batido o no un nuevo récord, solicitándole sus credenciales en caso afirmativo (véase figura D.35). Para salir de todas estas animaciones, el jugador debe presionar *START*. Tras concluir la animación, se regresa al menú de selección de circuito si se está jugando al *Pole Position*. Si se está jugando al modo *World Tour*, se ha quedado en primera posición, y no se han pasado todos los niveles (cuatro) se promociona a la siguiente carrera. Si se ha completado el

cuarto nivel se regresa al menú de presentación. Si no ha quedado primero, se le pregunta si desea repetir la carrera. En caso de jugar a los modos *Driving Fury* o *Demarrage* se pasa al siguiente nivel si no se han pasado todos (tres). En caso contrario se retorna al menú de presentación.

Figura D.35: Fin de carrera en modo *Pole Position*

D.8.3.2. Fin de modo de juego

Estas animaciones muestran al jugador la completitud de un modo de juego. Este tipo de animación aparece única y exclusivamente en un modo de juego, el *Out Run*.

Cuando el jugador alcanza la meta en el modo de juego *Out Run*, al jugador se le muestra la bonificación de puntos que se le suma al total de puntos acumulados durante toda la partida. Esta bonificación añadida varía en función del tiempo que le ha sobrado para completar el nivel (véase la figura D.36). Tras concluir esta animación se visualiza un *ranking* con los siete mejores corredores, ordenados por orden descendente de puntuación (véase figura D.37). Es aquí donde se evalúa si el jugador ha batido o no un récord. Si lo hace se le solicitan sus credenciales y será mostrado en la clasificación en la posición que le corresponde. Si se pulsa *START*, se sale del *ranking* y se regresa al menú de presentación.

Figura D.36: Bonificación tras completar el modo *Out Run*

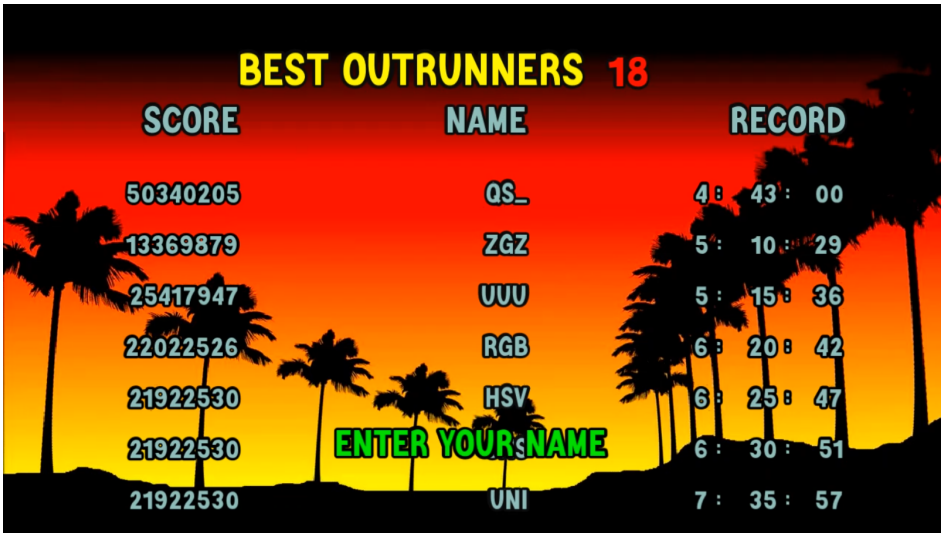


Figura D.37: *Ranking* de clasificación del modo *Out Run*

D.9. Créditos

Los créditos finales constituyen una parte esencial de cualquier producto puesto que en ellos se reflejan quiénes son los creadores que lo han realizado, las tareas de las que se han ocupado los distintos desarrolladores, en caso de haber varios, así como de referenciar todo *software* de terceros que se haya integrado en la creación del producto (*sprites*, bandas sonoras y efectos de sonido).

Los créditos de *Multi Race Driving* son un conjunto de pantallas, accesibles desde el menú de presentación pulsando la tecla *SPACE*. Dichas pantallas muestran, en

primer lugar, los datos generales del *software* (nombre, desarrollador, año de creación y licencia *Creative Commons*[1] a la que está sometido. Seguidamente se informa de la atribución de la propiedad de los derechos de los recursos empleados a sus correspondientes propietarios. Cabe enfatizar, que no hay intención de conseguir beneficios lucrativos a través de *Multi Race Driving*. En dichos créditos se referencian, entre otras cosas, los autores de todas las bandas sonoras, del *SFX*, de los fondos y de los *sprites* utilizados para el desarrollo de la arquitectura, y del *framework Linda* para construir el modo multijugador (véanse figuras D.38, D.39, D.40 y D.41).

Figura D.38: Atribución de los derechos de autor

Figura D.39: Referenciación de algunas bandas musicales utilizadas

[1] `https://creativecommons.org/licenses/?lang=es_ES`

Figura D.40: Referenciación de los *sprites* utilizados

Figura D.41: Referenciación al *framework Linda*

Anexo E

Configuración de los menús

En este anexo se describe de una manera más exhaustiva el mecanismo a seguir para poder configurar los menús que componen la *GUI* de *Multi Race Driving*. Se hace énfasis en el hecho de que aunque todos los menús poseen una estética y apariencia muy semejante, siempre hay algunas diferencias entre ellos puesto que no muestran a los jugadores la misma información.

Ante esta realidad, se ha procurado que todos los ficheros *XML* de configuración tengan una sintaxis fácilmente entendible para los jugadores, en especial, para aquellos que desean modificar la *GUI*. A continuación, se explica como ejemplo el fichero de configuración del menú de selección de modos de juego para un solo jugador.

Los menús de configuración se encuentran en el directorio *Menus*, accesible desde la carpeta principal del sistema de archivos de *Multi Race Driving*, llamada *Data*. En dicha carpeta se encuentran diversos subdirectorios. Cada uno de ellos se corresponde con un menú concreto de la *GUI*. El directorio que para este ejemplo se busca es *GameModesMenu*.

En cada una de estas carpetas, inclusive la ahora mencionada, hay dos subcarpetas llamadas *Images* y *Configuration*, donde se almacenan las imágenes a emplear en el menú y el fichero de configuración *XML* de ese menú, respectivamente. El fichero de configuración tiene la sintaxis que a continuación se procede a explicar.

Inicialmente, se comienza escribiendo la versión del formato *XML* y la codificación de los caracteres del fichero por motivos de cumplimiento del estándar. Seguidamente se encuentra el primer nodo *XML* del fichero cuyo alias coincide con el nombre del fichero *XML* en cuestión, en este caso, *GameModesMenu*. Los siguientes nodos, los nodos hijos, guardan las propiedades del menú.

El primero de todos ellos es el nodo *Background* donde debe especificarse la ruta de la imagen que se desea emplear como fondo del menú. A continuación, se encuentra el nodo *MenuPanel*, que refleja el cuadrado principal del menú en cuyo interior aparece todo el contenido del mismo. Este nodo posee dos nodos descendientes, *Background* y *ColorBorder*, en donde se especifica la ruta de la imagen de fondo del panel y el color del borde, en formato *RGB*.

Seguidamente, se encuentra el nodo *Title*, que contiene toda la información referente al título del menú, localizado en la parte superior central. Este nodo presenta los siguientes aspectos de configuración:

- *Content*: este nodo permite modificar el contenido del título del menú.
- *Font*: este nodo controla la fuente con la que se escribe el texto del título del menú.
- *ColorText*: contiene el color en formato *RGB* de las letras del título.
- *ColorBorder*: gestiona el color del borde del título.

Posteriormente, se encuentra el nodo *DescriptionPanel*, presente en todos aquellos menús en los que aparece un panel informativo, como es este caso o por ejemplo el menú principal. En dicho panel se puede configurar tanto el color interno como el color del borde, marcados por los nodos *ColorPanelInside* y *ColorPanelBorder*, ambos en formato *RGB*, respectivamente.

Después, se encuentra el nodo *MenuButtons* donde reside toda la información referente a los botones del menú, obviamente este nodo y todos los aspectos de configuración que de él se derivan, solo están disponibles en los menús con botones. Mediante el nodo *Font* se configura la fuente en la que se va a escribir el texto de los distintos botones, y con el nodo *ColorFont* se codifica en formato *RGB* el color del texto.

Una vez procesados los atributos generales de los botones, se van a explicar los atributos específicos de cada uno. Los distintos botones están integrados dentro del nodo *Buttons*, a modo de lista, donde los nodos del tipo *BX* representan los distintos botones del menú, siendo *X* un código numérico correlativo para identificar a cada botón. Los aspectos concretos a modificar de cada botón son los que a continuación se detallan:

– *Content*: al igual que en el título, este campo representa el contenido del texto del botón.

– *InitialState*: este nodo representa el estado inicial del botón (seleccionado o no seleccionado), expresado mediante un uno o un cero respectivamente.

– *Colors*: en este nodo se marcan los dos posibles colores que puede tener un botón en base a su estado. Así, el nodo *C1* marca el color cuando el botón no está marcado, y el nodo *C2* marca el color cuando el botón está seleccionado. Los colores son en formato *RGB*.

– *Description* es un nodo opcional que representa la descripción del significado de cada botón. solo aparece en los menús que tienen panel descriptivo, de ahí, que sea opcional. La descripción está dividida en diferentes partes, marcadas por el nodo *DX*, donde *X* es el identificador de cada fragmento de descripción. Estos nodos solo se usan para poder estructurar mejor la descripción en líneas de tamaño más pequeño en vez de ponerla en una única línea más larga. El *software* procesa las diferentes partes y las junta en una única de manera automática.

Aspectos como el tamaño de las fuentes, el grosor de los bordes, las posiciones de los distintos elementos o incluso sus dimensiones no pueden configurarse. La razón estriba en que como existen distintas resoluciones para configurar la pantalla, todos estos factores son modificados automáticamente dejando siempre los menús simétricos y cuadrados.

Si los jugadores desean modificar los aspectos de algún menú solo deben acceder al fichero de configuración *XML* correspondiente, cambiar los valores que toman los nodos y guardar los cambios del fichero. A la hora de arrancar, *Multi Race Driving* se ocupará de procesarlos automáticamente a través del módulo de generación de menús. En este punto, se muestra el contenido del documento *XML* recientemente explicado:

```
<?xml version="1.0" encoding="UTF-8"?>
<GameModesMenu>
  <Background>Data/Menus/GameModesMenu/Images/cover.png</Background>
  <MenuPanel>
    <Background>Data/Menus/GameModesMenu/Images/crono.png</Background>
          <ColorBorder>
        <R>0</R>
        <G>0</G>
        <B>0</B>
      </ColorBorder>
  </MenuPanel>
```

```
<Title>
  <Content>SELECT GAME MODE</Content>
      <Font>Data/Fonts/Hetikademo.otf</Font>
      <ColorText>
        <R>255</R>
        <G>255</G>
        <B>255</B>
      </ColorText>
      <ColorBorder>
        <R>0</R>
        <G>0</G>
        <B>0</B>
      </ColorBorder>
</Title>
<DescriptionPanel>
  <ColorPanelInside>
        <R>255</R>
    <G>255</G>
    <B>0</B>
      </ColorPanelInside>
      <ColorPanelBorder>
        <R>0</R>
        <G>0</G>
        <B>0</B>
      </ColorPanelBorder>
</DescriptionPanel>
<MenuButtons>
  <Font>Data/Fonts/Hetikademo.otf</Font>
  <ColorFont>
        <R>0</R>
        <G>0</G>
        <B>0</B>
      </ColorFont>
      <Buttons>
        <B0>
          <Content>World Tour</Content>
          <InitialState>1</InitialState>
          <Colors>
            <C1>
              <R>249</R>
              <G>54</G>
              <B>46</B>
            </C1>
            <C2>
              <R>255</R>
              <G>255</G>
              <B>255</B>
            </C2>
          </Colors>
          <Description>
            <D1>Run in different parts of the world</D1>
            <D2>facing other rivals and traffic.</D2>
            <D3>Only the first one is worth it.</D3>
            <D4>Mode inspired in Super Hang on game.</D4>
          </Description>
        </B0>
        <B1>
          <Content>Out Run</Content>
```

```
    <InitialState>0</InitialState>
    <Colors>
      <C1>
        <R>249</R>
        <G>54</G>
        <B>46</B>
      </C1>
      <C2>
        <R>255</R>
        <G>255</G>
        <B>255</B>
      </C2>
    </Colors>
    <Description>
      <D1>Circulate through five different scenarios</D1>
      <D2>reaching the checkpoint before time runs out.</D2>
      <D3>Mode inspired in Out Run game.</D3>
    </Description>
  </B1>
  <B2>
    <Content>Pole Position</Content>
    <InitialState>0</InitialState>
    <Colors>
      <C1>
        <R>249</R>
        <G>54</G>
        <B>46</B>
      </C1>
      <C2>
        <R>255</R>
        <G>255</G>
        <B>255</B>
      </C2>
    </Colors>
    <Description>
      <D1>Compete in intense F1 races and beat</D1>
      <D2>all your rivals in the shortest time possible.</D2>
      <D3>Mode inspired in Pole Position game.</D3>
      </Description>
  </B2>
  <B3>
    <Content>Driving Fury</Content>
    <InitialState>0</InitialState>
    <Colors>
      <C1>
        <R>249</R>
        <G>54</G>
        <B>46</B>
      </C1>
      <C2>
        <R>255</R>
        <G>255</G>
        <B>255</B>
      </C2>
    </Colors>
    <Description>
      <D1>Chase and stop a criminal vehicle with</D1>
      <D2>a police car before time runs out.</D2>
```

```
            <D3>Mode inspired in Chase HQ game.</D3>
          </Description>
        </B3>
        <B4>
          <Content>Demarrage</Content>
          <InitialState>0</InitialState>
          <Colors>
            <C1>
              <R>249</R>
              <G>54</G>
              <B>46</B>
            </C1>
            <C2>
              <R>255</R>
              <G>255</G>
              <B>255</B>
            </C2>
          </Colors>
          <Description>
            <D1>Chase a rival vehicle until you get</D1>
            <D2>some distance ahead of it.</D2>
            <D3>Mode inspired in Need for Speed Undercovergame.</D3>
            </Description>
         </B4>
    </Buttons>
  </MenuButtons>
</GameModesMenu>
```

Anexo F

Gráficos y renderización

En este anexo se profundiza en cómo se ha llevado a cabo el renderizado de los gráficos de *Multi Race Driving*.

F.1. Cámara y renderizado

Los escenarios están divididos en rectángulos, explicados en la generación de los escenarios a través de los ficheros *XML*. Dichos rectángulos a su vez están integrados por la unión de diversas líneas, concretamente, un rectángulo se compone de cinco líneas. Las unidades que componen los mapas se han confeccionado así para que, en primer lugar, los elementos decorativos de los escenarios no queden disjuntos unos de otros, y en segundo lugar, que los vehículos, ya sea el coche del jugador, o los rivales, o los coches de tráfico no avancen dando saltos de un rectángulo a otro, sino que avancen poco a poco dentro de cada rectángulo. Si la disgregación de los rectángulos en líneas no se hubiese efectuado, al jugar la partida los coches se moverían dando saltos creando una falsa sensación de *lag*, lo que sería pésimo de cara a la jugabilidad.

Se ha posicionado una cámara detrás y encima del coche del jugador para tener una buena paronamizacion de todo lo que ocurre en escena. Se enfatiza en el hecho de que nunca es el vehículo del jugador quién se mueve, sino que es la cámara la que avanza a lo largo de la carretera. Esta cámara tiene una distancia de renderizado (punto de fuga) de 450 líneas, es decir, se muestran todos los vehículos, objetos, carretera y suelo que estén a 450 líneas o menos de la posición del jugador. El renderizado comienza por la línea más alejada y termina en la línea del jugador.

Según la distancia a la que se encuentra la línea a renderizar, se escala todo su contenido a un valor u otro. De esta manera se consigue que las líneas más alejadas se muestren con una escala más pequeña y las líneas más cercanas con una escala más grande, es decir, se consigue una sensación de profundidad.

Debido a que la carretera y algunas imágenes tienen una gran resolución, se ha diseñado un efecto de *pixel art* que consiste en renderizar la imagen en una resolución menor y escalarla a la resolución de pantalla. De esta manera se consigue que los píxeles sean más visibles y dé la impresión de un efecto *retro*. En concreto, para la resolución 921 x 691 se renderiza la imagen en 460 x 345, y para las demás resoluciones se renderiza la imagen en 640 x 360. La opción de jugar con los gráficos teniendo efecto de *pixel art* es completamente configurable por el jugador. Los menús siguen siendo generados para la resolución nativa de la pantalla (alta calidad) ya que si se realiza el mismo proceso de escalado la letra resulta ilegible. Además, se ha forzado la renderización de 60 imágenes por segundo (60 fps). De esta manera, los tiempos y lógica utilizada siempre serán los mismos (si no se llegara a estos fps la jugabilidad se vería reducida).

F.2. *Sprites*

Los *sprites* y fondos empleados son imágenes formato *.png* debido a que es un formato de imagen sin pérdida. Algunos de los *sprites* se han extraído de depósitos de imágenes *online* gratuitos, y otros han sido creados por el propio desarrollador. Los depósitos de imágenes *online* de donde los *sprites* han sido extraídos son los siguientes:

- *Vectezzy*: es un depósito de imágenes muy variado. Ofrece imágenes sometidas a varios tipos de licencias. Las imágenes no requieren ningún tipo de atribución.
- *PngTree*: es un depósito de imágenes que a diferencia del anterior requiere membresía. Las imágenes no requieren ningún tipo de atribución.
- *SpriteResource*: es una página web completamente gratuita que almacena una amplia colección de hojas de *sprites* para distintos juegos y distintas plataformas.
- *Vexels*: es un depósito de imágenes muy amplío que se caracteriza por el hecho de que sus imágenes requieren atribución para poder usarse.
- *VectorStock*: es un depósito de imágenes muy variado. Posee imágenes gratuitas y de pago, aunque en este proyecto solo se han empleado las del primer

tipo. Requiere membresía, y además, las imágenes facilitadas por este depósito requieren acreditación.

– *Pngocean*: es un depósito con descarga gratuita e ilimitada de imágenes en formato *.png*.

Los fondos empleados para modelar los distintos escenarios proceden de los siguientes sitios:

– *Vectezzy*: explicado anteriormente.

– *Pngocean*: referenciado con anterioridad.

– *SpriteResource*: comentado anteriormente.

– *PngTree*: ya documentado.

– *VectorStock*: detallado con anterioridad.

– *Compight*: es un depósito gratuito de imágenes que requiere membresía y en el que todas las imágenes están sujetas a una licencia *Creative Commons*. No requieren atribución.

F.3. Geometría de los escenarios

Cada objeto, vehículo (y la cámara) está representado mediante un sistema de coordenadas en 3D del mapa, este sistema de coordenadas está formado por los ejes X, Y y Z que se describen a continuación:

– El eje Z representa la línea en la que se encuentra dicho objeto o vehículo (desde la línea de salida, hasta la línea de conexión con el siguiente mapa).

– El eje Y representa la elevación de dicha línea (tanto positiva como negativa).

– El eje X representa la ubicación horizontal en la línea en la que se encuentra, siendo 0.0 el centro de la carretera, mayor que 0.0 la parte derecha de la carretera y menor que 0.0 la parte izquierda de la carretera. La carretera ocupa desde -1.0 hasta 1.0.

Gracias a este sistema de coordenadas en 3D se puede representar la profundidad del mapa, y escalar las líneas más alejadas.

F.3.1. Curvas

Una curva está representada por un sumatorio de los índices de curvatura de las líneas adyacentes, es decir, si la cámara se encuentra en una línea con índice de curvatura mayor que 0.0 y todas sus líneas posteriores (dentro de la distancia de renderizado) tienen un índice de curvatura mayor que 0.0, significa que el jugador está dentro de una curva hacia la derecha que perdura durante todo el campo de visión del jugador.

En definitiva, si la cámara se encuentra en una línea con índice de curvatura distinto de 0.0 significa que el jugador está en una curva, y si es 0.0 es que está en una recta. Además, si alguna de las líneas (varias para que se aprecie) dentro del campo de visión del jugador tiene un índice de curvatura distinto al de la línea en la que se encuentra la cámara, se apreciará cómo varía la trayectoria de la carretera.

F.3.2. Elevaciones

Cada escenario tiene una elevación inicial que viene determinada por la elevación final del escenario anterior (0.0 en el inicial). Así, se cambia de mapa sin producirse saltos. La elevación de los mapas se calcula con la función coseno, ya que así se produce de manera progresiva simulando una colina. Cada rectángulo tiene una elevación determinada por la ecuación de cambio de rasante (véase figura F.1).

$$elevation = startElevation + elevationIndex + elevationIndex \cdot \cos\left(\pi + \frac{\pi \cdot lineIndex}{numLineas}\right)$$

Figura F.1: Ecuación del cambio de rasante

Las líneas de los rectángulos realizan un incremento de la elevación de manera equitativa, por ejemplo, un rectángulo incrementa su elevación en 100 respecto a su rectángulo predecesor, y como el rectángulo está formado por cinco líneas, cada línea incrementa su elevación en 20 con respecto a su predecesora. La elevación inicial (*startElevation*) corresponde a la elevación que había antes de comenzar esta zona de elevación. El índice de elevación (*elevationIndex*) es establecido en el generador de mapas, positivo en subidas y negativo en bajadas. El valor (*numLineas*) es el número de líneas que tiene la zona de elevación, y el índice de línea (*lineIndex*) identifica a cada línea de la elevación. A continuación, se puede observar un ejemplo con la elevación inicial = 0, el índice de elevación = 1.5 y el número de líneas = 10 (véase figura F.2). Como se puede ver, en la línea 10 se alcanza el punto máximo.

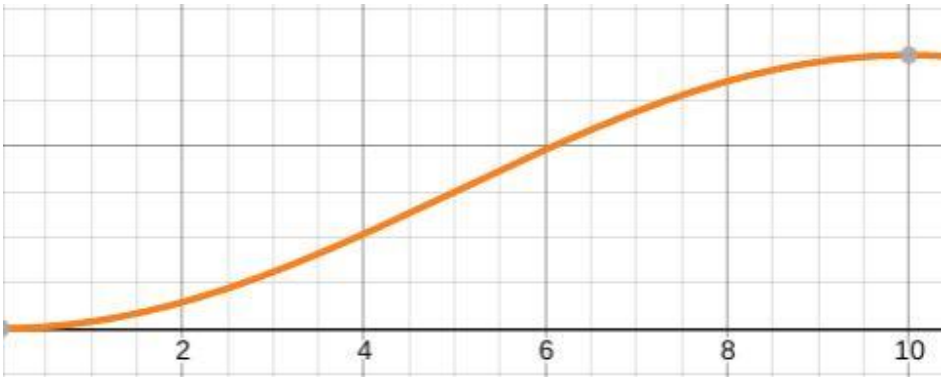


Figura F.2: Gráfica que muestra el modelado de cambios de rasante

F.4. Escenarios

En esta sección se describen con sumo detalle los distintos tipos de escenarios diseñados y como se han generado.

F.4.1. Generación de escenarios de nivel

A continuación, se explican los ficheros *XML* donde se hallan definidos los escenarios de nivel de los distintos modos de juego, así como el formato de fichero que se ha creado para poder plasmar toda la información referente a su contenido, y por último, cómo los jugadores pueden modificar dichos ficheros.

El generador automático de mapas crea los escenarios a partir de las carpetas *LandScape1*, *Landscape2*, ..., *LandscapeN*, ubicadas en los directorios *World Tour*, *Out Run, Pole Position*, *Driving Fury* y *Demarrage*, accesibles desde el directorio *GameModes*, presente en el directorio principal *Data*. Cada una de estas carpetas contiene la información relativa a cada escenario, estructurada de la siguiente manera.

F.4.1.1. Objetos del escenario

Los objetos están definidos tanto por las imágenes de los *sprites*, identificadas con un dígito correlativo que va desde el valor uno hasta el número de objetos de ese mapa (1.png, 2.png, . . . , y así sucesivamente), como por la información opcional de cada elemento. Dicha información se encuentra en ficheros *XML* cuyos nombres coinciden con los identificadores de cada objeto (1.xml, 2.xml, ..., y así sucesivamente).

Los ficheros relativos a las propiedades de cada elemento son simples ficheros *XML* de existencia opcional que tienen la sintaxis que a continuación se explica. En primer lugar, se especifica el tipo de codificación de los caracteres y la versión empleada del lenguaje *XML*. No es información obligatoria pero se ha colocado para cumplir la normativa del estándar *XML*.

Seguidamente, se encuentra el nodo *XML* principal del fichero, que representa las propiedades, identificado como *Properties*. Son los hijos de este nodo los que explican las características del objeto. Para poder representarlas se utilizan los tipos de nodos opcionales *Scale*, *Hit_center*, *Hit_left*. *Hit_right* y *Hit_sides*.

Los valores de todos estos nodos son números decimales. El nodo *Scale* puede tomar valores positivos desde 0.0 hasta 3.40282e+038 y multiplica el tamaño del objeto por ese valor. Los restantes tipos de nodos *Hit_center*, ..., *Hit_sides* indican el porcentaje de la imagen que es colisionable con los vehículos del jugador. Los valores decimales que acompañan a estos tipos de nodos van desde 0.0 hasta 1.0. El valor 0.0 establece que el objeto es completamente atravesable y 1.0 indica que toda la imagen es sólida y el coche se puede chocar.

Los nodos *Hit_center*, ..., *Hit_sides* definen la parte sólida del objeto desde el centro del mismo, por ejemplo para definir que sólo se permiten los choques con los troncos de los árboles. *Hit_right* y *Hit_left* definen la parte sólida del objeto desde la derecha y la izquierda del mismo respectivamente. Un ejemplo de su utilidad sería para poder definir que solo se permiten choques con el poste de las farolas en *L*. El nodo *Hit_sides* es la unión de *Hit_right* y *Hit_left* de manera simultánea, empleado por ejemplo para indicar que solo se permiten choques con los extremos de los puntos de control. El siguiente texto en XML refleja las propiedades para los puntos de control:

```
<?xml version="1.0" encoding="UTF-8"?>
<Properties>
        <Hit_sides>0.25</Hit_sides>
</Properties>
```

F.4.1.2. Fondo

El fondo del mapa está definido en el fichero *bg.png* y consiste en una imagen sólida de alta calidad y resolución.

F.4.1.3. Trazado del escenario

El trazado del escenario está definido en los ficheros *LandScape.xml*, que contienen toda la información referente al mapa, desde sus colores hasta su modelado, formado por rectángulos horizontales. Cada rectángulo contiene el trazado de la carretera, curvado o recto, llano o con elevación, y los posibles elementos decorativos a la derecha, a la izquierda, o en ambos lados de la carretera.

El primer nodo *XML*, denominado *LandScape* es el nodo raíz del documento. Los nodos hijos de este nodo definen las propiedades generales del escenario. Siguiendo una secuencia cronológica, los atributos del escenario vienen definidos por los siguientes nodos:

- *Name*: este nodo indica el nombre del circuito, es un campo que puede estar o no presente en el fichero. Para los escenarios del modo *World Tour* este nodo sí que figura pero para el resto de modalidades no.
- *Type*: este nodo representa el tipo de terreno que tiene el paisaje. Este nodo es utilizado para poder determinar el tipo de terreno que tiene que levantar el vehículo cuando está fuera de la carretera. Sus posibles valores pueden ser *Grass* (hierba), *Land*, (tierra) y *Snow* (nieve).
- *Sprites*: este nodo cuenta con la lista de todos los elementos decorativos presentes en el escenario. Para cada uno de estos elementos se indica en el fichero la ruta de la imagen a cargar dentro del sistema de archivos.
- *Terrain*: este es el nodo más importante del fichero ya que es donde figura todo el trazado del paisaje. El circuito se halla dividido en diferentes partes o fragmentos representados por los nodos con código de identificación PX, donde P es una abreviatura de *Parte* y X es el número de fragmento del escenario.
- *Color_grass*: este nodo representa los colores con los que se pinta el terreno del escenario que no es carretera.
- *Color_road*: este nodo representa los colores con los que se pinta la calzada.
- *Color_rumble*: este nodo representa los colores con los que se deben pintar los bordes de la carretera.
- *Color_dash*: este nodo refleja el color con el que se deben pintar los carriles en los que se haya dividida la calzada.

Todos estos colores están especificados en formato *RGB*. La decisión de que los escenarios estén divididos en partes o fragmentos deriva en que los tamaños de los ficheros son mucho más pequeños que si la construcción de los escenarios se hubiese hecho de manera paulatina, es decir, rectángulo a rectángulo. Los intervalos son, por tanto, conjuntos de rectángulos del escenario que comparten unas propiedades comunes.

Cada uno de los intervalos en los que se divide el circuito puede ser una recta, una curva o un cambio de rasante (subida o bajada). Para cada uno de estos tramos, se especifica su posición de origen y su posición de acabado. De acuerdo a la sintaxis del fichero, todo intervalo se define con el nodo *Road* seguido de las posiciones de comienzo y fin de ese intervalo. Tales límites son marcados por los atributos *startPosGE*, *startPosGT*, *finalPosLT* o *finalPosLE* seguido del número de réctangulo donde empieza o acaba. Las terminaciones *GE*, *GT*, *LT* y *LE* son abreviaturas de los comparadores lógicos *Greater Equal*, *Greater Than*, *Lower Than* y *Lower Equal*.

Para poder representar si ese fragmento de carretera contenido en el nodo *Road* es una recta, una curva o un cambio de rasante, se hace uso de los siguientes nodos:

- *Straight*: indica que el intervalo marcado desde la coordenada de inicio, reflejada por los atributos *startPosGE*/*startPosGT*, hasta la coordenada de acabado, marcada por los atributos *finalPosLT*/*finalPosLE* es una recta.

- *Curve*: indica que el intervalo marcado desde la coordenada de inicio, reflejada por los atributos *startPosGE*/*startPosGT*, hasta la coordenada de acabado, marcada por los atributos *finalPosLT*/*finalPosLE* es una curva. Paralelamente a los atributos anteriores, existe otro atributo adicional, el atributo *direction*, cuyo valor es un número decimal que indica la dirección de la curva, izquierda si es negtivo y derecha si es positivo. Cuánto más positivo o negativo sea el valor decimal, más pronunciado será el radio de la curvatura en un sentido u otro.

- *Climbing*: indica que el intervalo marcado desde la coordenada de inicio, reflejada por los atributos *startPosGE*/*startPosGT*, hasta la coordenada de acabado, marcada por los atributos *finalPosLT*/*finalPosLE* es un cambio de rasante ascendente (subida). De igual manera que para las curvas, se usa un atributo *direction* para determinar la posible trayectoria curva. Si el decimal es positivo la subida es a la derecha, si es negativo la subida es a la izquierda y si toma valor 0.0 la subida es recta. Además, se encuentra el atributo *Elevation* que indica como de pronunciada es la pendiente, marcada con un entero positivo.

– *Dropping*: indica que el intervalo marcado desde la coordenada de inicio, reflejada por los atributos *startPosGE/startPosGT*, hasta la coordenada de acabado, marcada por los atributos *finalPosLT/finalPosLE* es un cambio de rasante descendente (bajada). La dirección de la bajada y la pendiente se codifican exactamente igual que en la subida.

A diferencia de las rectas y las curvas, los cambios de rasante pueden estar compuestos de varios fragmentos internos. Así, en una misma subida o bajada puede haber curvas y rectas. Dichos fragmentos se hallan codificados con los identficadores *PX*, donde *P* es una abreviatura de *Parte* y *X* es el número de fragmento.

Finalmente, para cada uno de los fragmentos del circuito, se especifica cúales son los elementos decorativos a dibujar. Dicha información se representa con los siguientes nodos:

– *SpriteNearLeft*: representa a los elementos que aparecen próximos y a la izquierda de la carretera.

– *SpriteNearRight*: refleja los elementos que aparecen próximos y a la derecha de la calzada.

– *SpriteFarLeft*: representa a los elementos que aparecen lejanos y a la izquierda de la carretera.

– *SpriteFarRight*: refleja los elementos que aparecen lejanos y a la derecha de la carretera.

Los nodos anteriores, para representar los elementos decorativos, cuentan con las propiedades siguientes:

– El nodo *Each*: refleja cada cuánto tiempo debe mostrarse ese elmento en el intervalo.

– El nodo *Offset*: indica la distancia a la que aparece el elemento de la carretera. Si el offset toma valor positivo, se aleja de la carretera. Si toma valor negativo se acerca al centro de la carretera y si toma valor 0.0 aparece pegado al borde. Si toma valor *Random* aparece en una posición aleatoria a izquierda o derecha según indique el tipo de nodo.

– El nodo *Code*: indica el código indentificador del elemento decorativo del mapa.

Para poder indicar los puntos de control en los escenarios se emplea el nodo *CheckPoint*. Al igual que en los elementos decorativos, el nodo *Offset* indica a que distancia aparece el punto de control de la carretera. El nodo *Position* marca el punto exacto del intervalo donde aparece el punto de control. A diferencia de los elementos decorativos normales, los puntos de control sólo aparecen una vez. Finalmente el nodo *Code* marca el código de identificación del punto de control en el mapa. El siguiente fragmento de fichero *XML* muestra la especificación de los elementos decorativos del mapa:

```
<?xml version="1.0" encoding="UTF-8"?>
<LandScape>
<Type>Land</Type>
  <Sprites>
  <S1>Data/GameModes/OutRun/LandScapes/LandScape2/1.png</S1>
  <S2>Data/GameModes/OutRun/LandScapes/LandScape2/2.png</S2>
  <S3>Data/GameModes/OutRun/LandScapes/LandScape2/3.png</S3>
  <S4>Data/GameModes/OutRun/LandScapes/LandScape2/4.png</S4>
  <S5>Data/GameModes/OutRun/LandScapes/LandScape2/5.png</S5>
  <S6>Data/GameModes/OutRun/LandScapes/LandScape2/6.png</S6>
  <S7>Data/GameModes/OutRun/LandScapes/LandScape2/7.png</S7>
  <S8>Data/GameModes/OutRun/LandScapes/LandScape2/8.png</S8>
  <S9>Data/GameModes/OutRun/LandScapes/LandScape2/9.png</S9>
  <S10>Data/GameModes/OutRun/LandScapes/LandScape2/10.png</S10>
  <S11>Data/GameModes/OutRun/LandScapes/LandScape2/11.png</S11>
  <S12>Data/GameModes/OutRun/LandScapes/LandScape2/12.png</S12>
  <S13>Data/GameModes/OutRun/LandScapes/LandScape2/13.png</S13>
  <S14>Data/GameModes/OutRun/LandScapes/LandScape2/14.png</S14>
  <S15>Data/GameModes/OutRun/LandScapes/LandScape2/15.png</S15>
  <S16>Data/GameModes/OutRun/LandScapes/LandScape2/16.png</S16>
  <S17>Data/GameModes/OutRun/LandScapes/LandScape2/17.png</S17>
  <S18>Data/GameModes/OutRun/LandScapes/LandScape2/18.png</S18>
  <S19>Data/GameModes/OutRun/LandScapes/LandScape2/19.png</S19>
  <S20>Data/GameModes/OutRun/LandScapes/LandScape2/20.png</S20>
  <S21>Data/GameModes/OutRun/LandScapes/LandScape2/21.png</S21>
  <S22>Data/GameModes/OutRun/LandScapes/LandScape2/22.png</S22>
  <S23>Data/GameModes/OutRun/LandScapes/LandScape2/23.png</S23>
  <S24>Data/GameModes/OutRun/LandScapes/LandScape2/24.png</S24>
  <S25>Data/GameModes/OutRun/LandScapes/LandScape2/25.png</S25>
  <S26>Data/GameModes/OutRun/LandScapes/LandScape2/26.png</S26>
</Sprites>
```

Seguidamente se muestra un ejemplo de un tramo de escenario correspondiente a una recta:

```
<P4>
  <Straight>
    <Road startPosGT="99" finalPosLE="110">
      <SpriteNearLeft>
        <Each>74</Each>
        <Offset>0.4</Offset>
```

```
        <Code>7</Code>
      </SpriteNearLeft>
      <SpriteNearRight>
        <Each>75</Each>
        <Offset>Random</Offset>
        <Code>20</Code>
      </SpriteNearRight>
      <SpriteFarLeft>
        <Each>63</Each>
        <Offset>1.8</Offset>
        <Code>23</Code>
      </SpriteFarLeft>
      <SpriteFarRight>
        <Each>52</Each>
        <Offset>Random</Offset>
        <Code>23</Code>
      </SpriteFarRight>
    </Road>
  </Straight>
</P4>
```

A continuación, se puede observar la especificación de un fragmento de escenario correspondiente a una curva:

```
<P8>
  <Curve direction="1.1">
    <Road startPosGE="291" finalPosLE="315">
      <SpriteNearLeft>
        <Each>63</Each>
        <Offset>0.0</Offset>
        <Code>18</Code>
      </SpriteNearLeft>
      <SpriteNearRight>
        <Each>69</Each>
        <Offset>0.0</Offset>
        <Code>18</Code>
      </SpriteNearRight>
      <SpriteFarLeft>
        <Each>30</Each>
        <Offset>Random</Offset>
        <Code>21</Code>
      </SpriteFarLeft>
      <SpriteFarRight>
        <Each>52</Each>
        <Offset>Random</Offset>
        <Code>21</Code>
      </SpriteFarRight>
    </Road>
  </Curve>
</P8>
```

El siguiente fragmento de código XML muestra la especificación de un cambio de rasante ascendente (subida):

```
<P9>
  <Climbing elevation="5200">
    <P1>
      <Direction>0.3</Direction>
      <Road startPosGE="316" finalPosLE="365">
      <SpriteNearLeft>
        <Each>62</Each>
        <Offset>0.2</Offset>
        <Code>3</Code>
      </SpriteNearLeft>
      <SpriteNearRight>
        <Each>68</Each>
        <Offset>1.0</Offset>
        <Code>4</Code>
      </SpriteNearRight>
      <SpriteFarLeft>
        <Each>54</Each>
        <Offset>Random</Offset>
        <Code>6</Code>
      </SpriteFarLeft>
      <SpriteFarRight>
        <Each>66</Each>
        <Offset>Random</Offset>
        <Code>20</Code>
      </SpriteFarRight>
    </Road>
    </P1>
  </Climbing>
</P9>
```

Seguidamente se muestra un ejemplo de un tramo de escenario correspondiente a un cambio de rasante descendente (bajada):

```
<P11>
  <Dropping elevation="9000">
    <P1>
      <Direction>0.0</Direction>
      <Road startPosGT="399" finalPosLT="500">
        <SpriteNearLeft>
          <Each>85</Each>
          <Offset>Random</Offset>
          <Code>23</Code>
        </SpriteNearLeft>
        <SpriteNearRight>
          <Each>102</Each>
          <Offset>0.2</Offset>
          <Code>22</Code>
        </SpriteNearRight>
        <SpriteFarLeft>
          <Each>30</Each>
          <Offset>Random</Offset>
          <Code>21</Code>
        </SpriteFarLeft>
```

```
        <SpriteFarRight>
          <Each>52</Each>
          <Offset>Random</Offset>
          <Code>1</Code>
        </SpriteFarRight>
      </Road>
    </P1>
  </Dropping>
</P11>
```

También se permite poder configurar todos los escenarios a la vez de un modo concreto de juego. Dentro de las carpetas *World Tour*, *Out Run*, *Pole Position*, *Driving Fury* y *Demarrage*, localizadas en el directorio *GameModes*, accesible desde la carpeta principal *Data*, existe una carpeta denominada *Configuration* que contiene un fichero en formato *XML* donde se guardan los aspectos de configuración generales de todos los escenarios de un modo.

La sintaxis del fichero *XML* es sumamente simple. Primero se especifica la versión de *XML* y la codificación de caracteres del fichero. Seguidamente se encuentra el nodo principal del fichero, denominado *Configuration*, donde se especifican las propiedades como tal. A continución, se encuentran los nodos siguientes:

- *Width_road*: representa la anchura de la carretera. Debe tomar valores positivos, preferiblemente entre 1000 y 3000. Las unidades no expresan magnitud de longitud ni distancia, es una métrica inventada.
- *Number_tracks*: representa el número de carriles en los que se debe dividir la calzada. Todos ellos de igual tamaño.
- *Rumble_limit*: representa el grosor de las líneas divisoras de los carriles.

Es importante tener en cuenta, que aunque el rango de valores para cada factor es bastante amplio, no todas las combinaciones renderizan un paisaje correcto, sino que puede salir deformado. Por ello, se recomienda usar configuraciones similares a la proporcionada por defecto. A continuación, se muestra un ejemplo de la sintaxis de este tipo de fichero para el modo de juego *World Tour*:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Width_road>2000</Width_road>
  <Number_tracks>2</Number_tracks>
  <Rumble_limit>0.1</Rumble_limit>
</Configuration>
```

F.4.2. Escenarios de salida, llegada y vuelta intermedia

Debido a las limitaciones del generador de mapas, la salida y la meta (véanse figuras F.3 y F.4, así como los escenarios intermedios de la salida cuando se dan vueltas a un circuito (véase figura F.5) son escenarios añadidos de manera automática al inicio y final, respectivamente, de los escenarios de nivel. El escenario inicial está formado por un conjunto de rectángulos que contienen todos los *sprites* de la salida, que varían en función del tipo de modo elegido.

Figura F.3: Ejemplo de mapa de salida

El mapa final está constituido por un conjunto de rectángulos que contienen todos los *sprites* de la meta.

Figura F.4: Ejemplo de mapa de llegada

El escenario que se visualiza cuando el jugador completa una vuelta al circuito, es un mapa exactamente idéntico al escenario inicial. De este modo al escenario del circuito, se le concatena como mapa sucesor el escenario de salida, y a éste se le concatena de nuevo el escenario del circuito. Este proceso se repite un total de N - 1 veces, siendo N el número de vueltas a dar al circuito. Al comenzar la última vuelta no se concatena el mapa inicial, sino el mapa de llegada.

Figura F.5: Ejemplo de mapa intermedio

F.4.3. Bifurcaciones

Las bifurcaciones son elementos que no están presentes en todos los modos de juego, dado que *World Tour* y *Pole Position* no las admiten por su temática.

Las bifurcaciones sólo están presentes en los modos *Out Run*, *Driving Fury* y *Demarrage*, concretamente al final de recorrer cada uno de los cuatro primeros escenarios escogidos, como ya se explicó en el diseño de los modos de juego. Al final de cada uno de estos escenarios se genera una bifurcación que conecta el mapa actual con sus dos vecinos de la siguiente etapa.

Esta bifurcación está modelada mediante la unión de los fragmentos de dos circunferencias con centros y radios precalculados (véase figura F.6). Sólo se calcula la bifurcación izquierda, ya que la bifurcación derecha es su reflejo (simetría). En la bifurcación a calcular, existen dos puntos de corte, el primero conecta la primera circunferencia con la segunda, y el segundo conecta la segunda circunferencia con el resto de la carretera recta hasta comenzar el siguiente escenario. De esta manera y

sin girar la cámara, se divide la carretera en dos tramos idénticos (con los colores del mapa con el que se conecta). Cuando el jugador está en un lado u otro de la bifurcación se carga un mapa u otro (véase figura F.7).

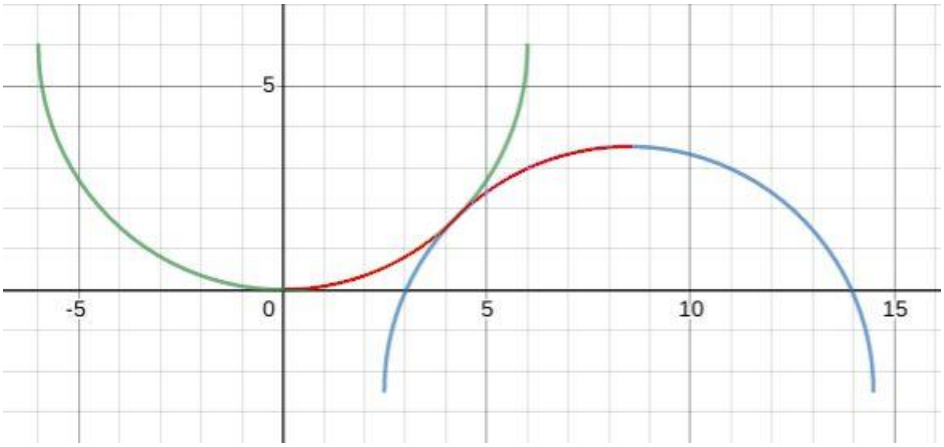


Figura F.6: Gráfica que muestra la apariencia de las bifurcaciones

Figura F.7: Renderizado de una bifurcación

La semicircunferencia verde representa la primera circunferencia, la semicircunferencia azul representa la segunda circunferencia, y el trazado rojo representa el trazado de la carretera en la bifurcación izquierda. Considerando sólo el trazado rojo y creando un reflejo simétrico se obtiene el trazado real de la bifurcación. (véase figura F.8).

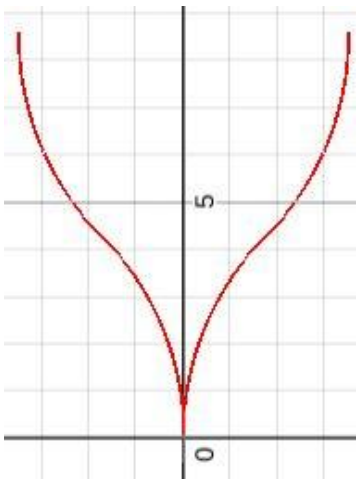


Figura F.8: Representación de las bifurcaciones

La primera circunferencia está modelada por una ecuación (véase figura F.9), que permite que la calzada se curve hacia la izquierda desde el origen (0, 0) creando la primera parte de la bifurcación.

$$x = \sqrt{R^2 - index^2} + R$$

Figura F.9: Ecuación de la primera circunferencia de la bifurcación

Donde x es el centro de la carretera, R el radio de la circunferencia (con valor 10) e *index* es el índice de la línea.

La segunda circunferencia está modelada por una ecuación (véase figura F.10), que permite curvar la calzada de manera que conecta la primera parte de la bifurcación con el resto de la carretera recta hasta comenzar el nuevo mapa.

$$x = \sqrt{R^2 - (index - A)^2} + B$$

Figura F.10: Ecuación de la segunda circunferencia de la bifurcación

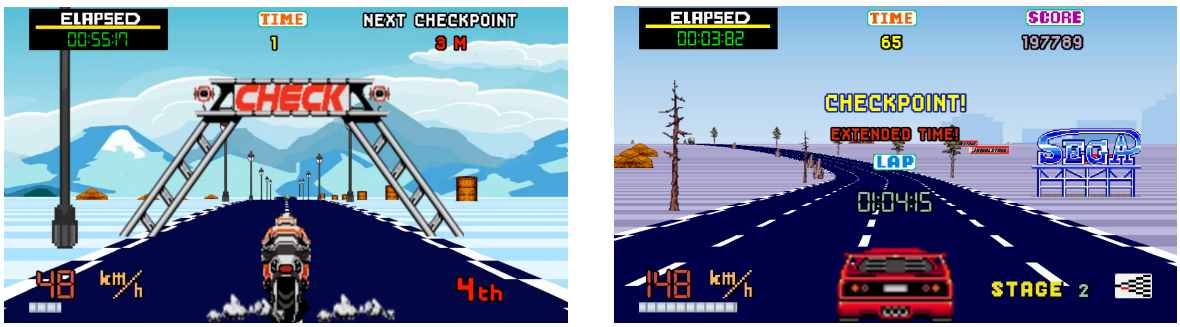
Donde A y B son valores constantes que se han precalculado para que el primer punto de corte entre las dos circunferencias coincida con los primeros 45^{o} recorridos de la primera circunferencia desde el origen, y el segundo punto de corte coincida con los primeros 45^{o} recorridos de la segunda circunferencia desde el primer punto de corte. De esta manera se consigue que no haya una curva imposible y se restablece la trayectoria de la carretera hacia la misma dirección desde la que se partía.

F.4.4. Puntos de control

Los puntos de control (véase figura F.11) están presentes en todos los modos de juego, multijugador incluido. En los modos de un sólo jugador, el punto de control le permite al jugador saber en qué posición se encuentra en ese momento, y además, le incrementa el tiempo de juego. En el modo multijugador, solo está presente la primera funcionalidad ya que no hay contador de tiempo de juego.

En los modos de juego *World Tour* y *Pole Position*, existen tres puntos de control, separados de modo equidistante. En el resto, los puntos de control se alcanzan tras salir de las bifurcaciones que permiten cambiar de un escenario a otro. En los modos de juego *World Tour* y *Pole Position*, la posición de los puntos de control es totalmente configurable ya que estos se encuentran reflejados en los correspondientes ficheros *XML* de los escenarios de nivel. En el resto de modos de juego esto no sucede así dado que los puntos de control se alcanzan cuando el vehículo del jugador cruza los límites de un escenario y pasa al siguiente adyacente.

En los modos de juego *World Tour* y *Pole Position*, cuando el jugador atraviesa un punto de control se reproduce un sonido de aviso, y se incrementa el tiempo de juego, en caso de jugar a la versión para un sólo jugador. Si se juega en el modo multijugador solo se reproduce el sonido. En el resto de modos ocurre lo mismo pero en el centro de la pantalla se muestra un mensaje que anuncia el punto de control, la extensión del tiempo de juego y el tiempo invertido por el jugador para ir desde el comienzo del escenario hasta el final de la bifurcación.



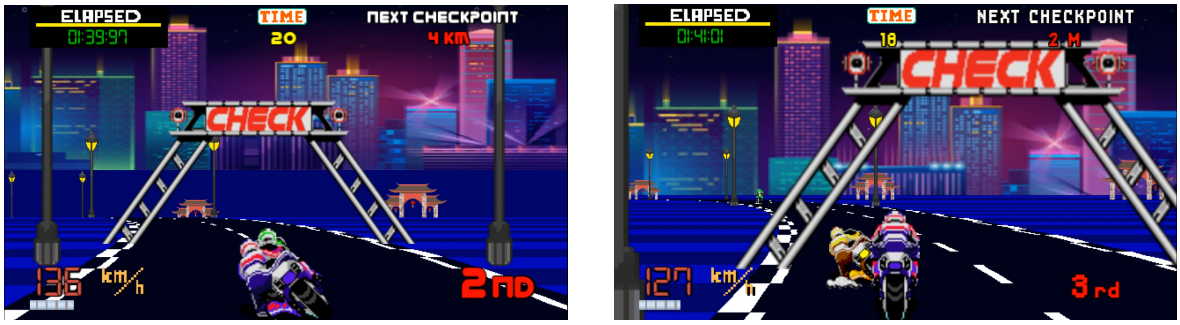
(a) Punto de control en modo *World Tour* (b) Colisión de un todoterreno

Figura F.11: Punto de control en modo *Out Run*

F.5. Comparativa de gráficos

Los gráficos de *Multi Race Driving* han sido diseñados con el fin de transmitir realismo y originalidad a los jugadores. La calidad y presencia que tienen los gráficos en la versión final de la arquitectura no es una tarea que se haya conseguido a la primera por diversas razones (algunos *sprites* no concordoban con la temática del modo de juego ni con el escenario, las *HUD's* tenían un tipo de fuente que no era acorde al estilo *retro* que caracteriza a la arquitectura, los fondos de los escenarios no se ajustaban bien, el punto de fuga para lograr la perspectiva no estaba bien calculado, etcétera). Teniendo presentes las retroalimentaciones de los distintos *beta testers*, fueron necesarios muchos cambios y renderizaciones para poder conseguir los gráficos actuales.

A continuación, se ofrece una comparativa entre los gráficos de *Multi Race Driving* en su versión *beta* y en su versión final (véase figura F.12). En dicha comparativa se pueden apreciar muchos de los detalles citados anteriormente, y al mismo tiempo, la gran diferencia en cuánto a mejora que hay entre ambas versiones.



(a) Versión *beta* de *Multi Race Driving* (b) Versión final de *Multi Race Driving*

Figura F.12: Comparativa de gráficos

Anexo G

Físicas de los vehículos

En este anexo se describe como se han diseñado tanto las físicas de los vehículos (cinemática y dinámica), como la configuración de las propiedades de los mismos.

G.1. Cinemática

El movimiento de los vehículos se logra por medio de un par de variables que regulan la aceleración y la velocidad, respectivamente. Es importante tener en cuenta que la velocidad de los vehículos no está en km/h, sino que es una medida inventada, cuya relación viene expresada por una fórmula matemática (véase figura G.1) que relaciona las dos variables anteriores (velocidad y aceleración).

$$velocidad = \sqrt{aceleracion}$$

Figura G.1: Ecuación que relaciona la velocidad y la aceleración de un vehículo

Esta fórmula matemática permite reflejar el comportamiento de los motores de los vehículos con un alto grado de realismo. Los motores de los vehículos reales funcionan incrementando su velocidad rápidamente cuando parten del reposo y conforme van acelerando se van aproximando a un cierto umbral del cuál no pueden excederse. Dicho umbral es la velocidad máxima del motor. La función matemática refleja de un modo sencillo este patrón de comportamiento ya que para valores pequeños, la función crece rápido (el motor acelera mucho) y para valores más grandes la función tiende a estabilizarse (el motor no sobrepasa su velocidad máxima). Para que el jugador vea la velocidad del vehículo en km/h, la velocidad es multiplicada por un factor que relaciona, de modo ficticio, la magnitud de la velocidad del bólido con los km/h.

En cada movimiento se incrementa o decrementa la aceleración del bólido, y la velocidad es solo el resultado final. Para que la velocidad y la aceleración no crezcan indefinidamente, se ha limitado la aceleración máxima a un valor predefinido. Si el jugador presiona el controlador para acelerar el vehículo, la aceleración de éste se incrementa y si presiona el controlador de frenado, la aceleración se decrementa. Si el jugador no acelera ni frena, la aceleración se reduce por sí sola para simular el efecto de soltar el pie del acelerador y no pisar el freno. Además, si el vehículo se ha salido de la carretera, la aceleración máxima también se ve reducida.

Los giros dependen de la velocidad del vehículo, cuanto más rápido va el vehículo más gira. A la hora de tomar las curvas, si el vehículo no va lo suficientemente rápido, experimenta una fuerza que lo empuja hacia el interior de la curva. Por el contrario, si va demasiado rápido se ve expuesto a la acción de una fuerza que lo empuja en el sentido contrario de la dirección de la curva, es decir, hacia fuera. Estos efectos han sido modelados para representar el efecto físico de la inercia, producido por las acciones de las fuerzas físicas centrípeta y centrífuga (primer caso y segundo caso respectivamente).

G.2. Dinámica

La dinámica ha consistido en modelar tanto los posibles factores (inercia, colisiones, etcétera) que producen alteraciones en el estado de los vehículos, como los mecanismos que permiten describir la evolución de los mismos (los vehículos) al momento de experimentarlos.

Si el vehículo colisiona, se bloquea su control y se reproduce una animación de colisión. Un factor influyente a la hora de modelar las colisiones ha sido la disponibilidad de *sprites* para poder construirlas. Dado que los vehículos no coinciden ni en tipo ni en número de *sprites*, se han tenido que modelar animaciones diferentes.

Los vehículos del jugador pueden experimentar durante la competición tres tipos de colisiones:

- Choques contra los elementos del escenario: estos choques se caracterizan porque, de sufrirlos, el vehículo del jugador pierde toda su velocidad.
- Choques contra los vehículos del tráfico: en este tipo de colisiones los bólidos también pierden toda su velocidad al completo.

– Choques contra los vehículos rivales: este tipo de colisión es la más leve de todas. Cuando suceden este tipo de golpes la velocidad prácticamente no se ve alterada, pero la trayectoria del vehículo sí.

Para proporcionar un mayor realismo a las animaciones, se han realizado los siguientes cálculos:

– Se guarda la velocidad del vehículo en el instante inicial de la colisión.

– Se almacena la posición en la que tiene lugar el choque.

– Se define un coeficiente de deceleración que hace perder velocidad al bólido mientras está chocando.

– En base a velocidad antes del golpe, a la posición en la que tiene lugar el siniestro y a dicho factor de deceleración, se estima la trayectoria del vehículo y cuánta es la distancia que debería recorrer desde el momento del golpe hasta que se detenga porque se queda sin aceleración.

Para la moto (véase figura G.2) la animación es la misma tanto si choca con elementos del escenario como con los coches del tráfico. En esta animación se puede observar tanto al piloto como a la moto deslizarse por la superficie de la calzada durante un pequeño tiempo. Cuando la moto se queda sin velocidad, es reubicada automáticamente en el centro de la carretera.

Figura G.2: Fotograma de la animación de colisión de la moto

Para los coches de *F1* (véase figura G.3), la colisión también es la misma si se choca contra un coche de tráfico o con un elemento del mapa. En esta animación se puede ver cómo el bólido explota. Tras acabar la explosión el vehículo es ubicado en el centro de la carretera.

Figura G.3: Fotograma de la animación de colisión del coche de *F1*

Con el resto de vehículos (coche deportivo, policía, todoterreno y camión) las animaciones al chocar con un obstáculo del escenario o con un vehículo de tráfico también son las mismas (véase figura G.4). Para todos ellos se sigue el mismo esquema. En dichas animaciones se puede observar a estos vehículos haciendo trompos.

Figura G.4: Fotograma de la animación de colisión del todo terreno

Para los choques contra los vehículos rivales de la carrera no se han realizado animaciones de choque, sino que han sido modelados como pequeños golpes en los que los vehículos son desviados ligeramente de su trayectoria.

Es importante mencionar que la utilización de ciertos recursos adicionales como la presencia de humo al efectuar los trompos, los sonidos de choque y los sonidos de derrape cuando los bólidos giran sin control por la calzada han contribuido a que las animaciones de las colisiones sean aún más realistas.

Otro aspecto muy importante de estos vehículos es su comportamiento cuando están fuera de la calzada. Para dotar a los gráficos de un mayor realismo, se ha diseñado un sistema que permite que los vehículos levanten hierba, tierra o nieve según el tipo de terreno del que sea el paisaje (véase figura G.5). Este sistema se fundamenta en la existencia de un campo, residente en los ficheros de configuración *XML* que especifica de que tipo es el terreno del escenario en cuestión. En base a este dato, el bólido al salirse fuera de la carretera levanta hierba, tierra o nieve.

(a) Hierba (b) Tierra (c) Nieve

Figura G.5: Vehículo circulando por el terreno de los paisajes

G.3. Configuración de los vehículos

Los vehículos jugables se hayan recogidos en las carpetas *Devastators*, *FormulaOnes*, *Minivans*, *Motorbikes*, *Police* y *Trucks*, localizadas dentro del directorio *Vehicles*, accesible desde la carpeta principal de archivos *Data*. En cada una de estas carpetas se pueden encontrar un máximo de ocho subdirectorios que representan los colores disponibles de ese tipo de bólido. Así, dentro de la carpeta *Devastators*, se hallan las subcarpetas *Devastator1*, ..., *Devastator8*. En estos directorios se localizan dos subcarpetas, denominadas *Images* y *Configuration*. Estos subdirectorios guardan el conjunto de *sprites* de ese vehículo y su configuración, respectivamente.

En la carpeta *Configuration*, existe un fichero denominado *Configuration.xml* que describe las propiedades del tipo de vehículo. El fichero en cuestión tiene el formato de sintaxis siguiente.

En primer lugar, se especifica el tipo de codificación de los caracteres y la versión empleada del lenguaje *XML*. No es información obligatoria pero se ha colocado para cumplir la normativa del estándar. A continuación, se encuentra el nodo *XML* principal del fichero, que representa el tipo de vehículo. De este modo, en función de la clase de bólido, el nodo puede tomar como valores *Motorbike*, *Devastator*, *Minivan*, *Truck*, *FormulaOne* o *Police*. Todo depende del tipo de vehículo. Finalmente, en sus nodos hijos se encuentran las características del vehículo en cuestión. Los nodos que representan las propiedades del vehículo son los citados a continuación:

- *Name*: este nodo representa la marca del tipo de vehículo siempre y cuando el bólido sea la moto, el coche deportivo, el todoterreno, el camión o el policía. En los coches de *F1*, simboliza el nombre de la escudería a la que pertenece el coche.
- *MaxSpeed*: indica la máxima velocidad alcanzable por el vehículo en km/h. Debe tomar siempre un valor positivo.
- *Angle*: representa el ángulo de giro con la que el vehículo toma las curvas, medida en unidades (es una métrica ficticia). El intervalo de valores es limitado y solo puede tomar valores positivos, concretamente, entre 0.0 (no gira nada) y 1.0 (gira muchísimo).
- *Motor*: informa del tipo de motor que tiene incorporado ese vehículo.

Para poder modificar estas propiedades el jugador solo tiene que sustiuir los valores que toman estos nodos por los que él desee y guardar el fichero *XML* con los nuevos cambios. *Multi Race Driving* se encargará de hacerlos efectivos cuando lleve a cabo la lectura del fichero en cuestión. A continuación, se muestra como ejemplo la sintaxis de uno de estos ficheros:

```
<?xml version="1.0" encoding="UTF-8"?>
<Devastator>
  <Name>Devastator IX</Name>
  <MaxSpeed>280</MaxSpeed>
  <Angle>0.048</Angle>
  <Motor>Kia Stinguer V6</Motor>
</Devastator>
```

Anexo H

Tráfico y vehículos rivales

En este anexo se procede a comentar los detalles técnicos y los matices de implementación más importantes tanto de los vehículos de tráfico como de los rivales.

H.1. Tráfico

El tráfico está formado por los coches que circulan en modo paisano mientras tienen lugar las carreras, es decir, son vehículos que transitan libremente por la calzada. Están presentes en casi todos los modos de juego para un solo jugador, a excepción del *Pole Position*, puesto que la presencia de tráfico en circuitos de *F1* no tenía base lógica alguna. Además, tampoco están disponibles en el modo multijugador.

Los *sprites* de estos coches se localizan en la carpeta *TrafficCars*, contenida dentro del directorio *Vehicles*, accesible desde la carpeta principal de archivos, *Data*. Esta carpeta contiene a su vez un total de seis subcarpetas, con nomenclatura *TrafficCarX*, donde *X* es un número correlativo cuyo rango es del uno al seis. De este modo, se identifica a cada tipo de coche de tráfico. En cada uno de estos directorios se encuentran los *sprites* de cada vehículo.

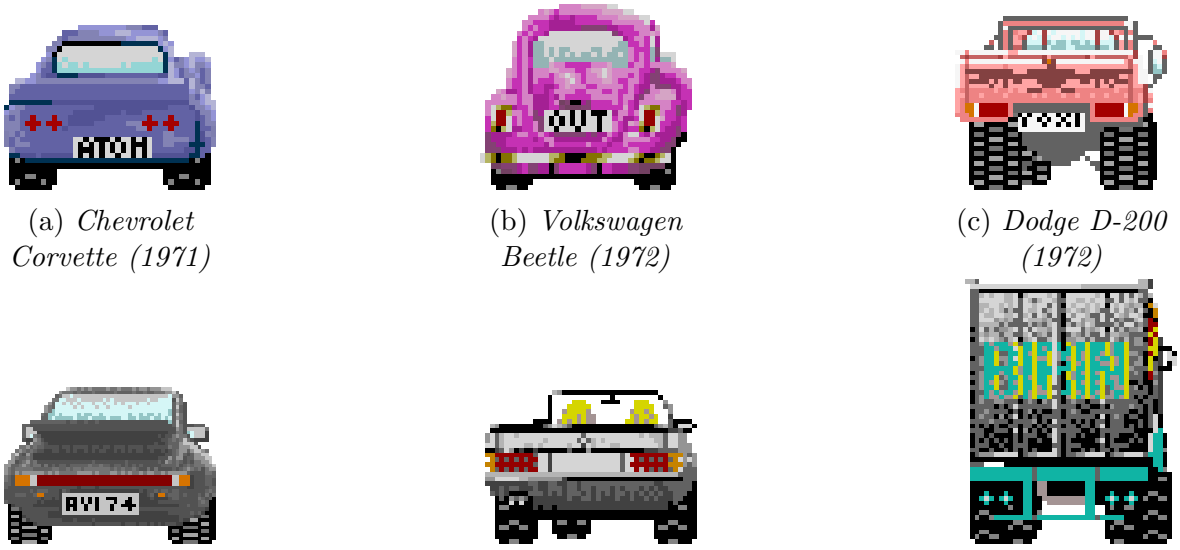
Los *sprites* de estos vehículos son los mismos que los del propio videojuego original para la recreativa del *Out Run* de 1986. Los coches de tráfico pueden ser de uno de los siguientes modelos:

- Chevrolet Corvette (1971).
- Volkswagen Beetle (1972).
- *Dodge D-200 Crew Cab (1972).*

– *Porsche 911 (1985).*

– *BMW Cabrio (1985).*

– *Dodge 800 (1986).*



(a) *Chevrolet Corvette (1971)*

(b) *Volkswagen Beetle (1972)*

(c) *Dodge D-200 (1972)*

(d) *Porsche 911 (1985)*

(e) *BMW Cabrio (1985)*

(f) *Dodge 800 (1986)*

Figura H.1: *Sprites* de muestra de los vehículos de tráfico

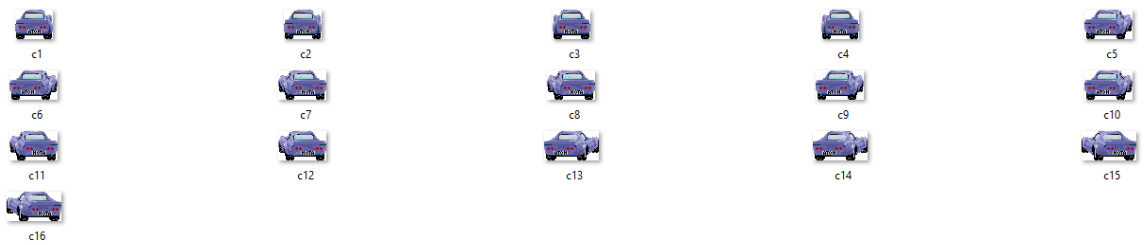
El número de vehículos de tráfico es estático y depende de la dificultad con la que se está jugando. Así, para el modo fácil hay cinco coches de tráfico, para el modo normal hay 10 coches, y para el modo difícil hay 15 coches. Estos vehículos se eliminan cuando se entra en una bifurcación (por simplicidad técnica) y cuando el jugador los ha dejado atrás a una distancia de, al menos, 50 líneas. Los vehículos se generan al inicio de cada mapa o cuando se han eliminado por quedarse atrás. A continuación, se detalla el proceso para poder generarlos:

– La velocidad inicial se establece de manera aleatoria con un valor comprendido entre el 25 % y el 75 % de un valor de velocidad máxima definido por defecto, concretamente, 300 km/h.

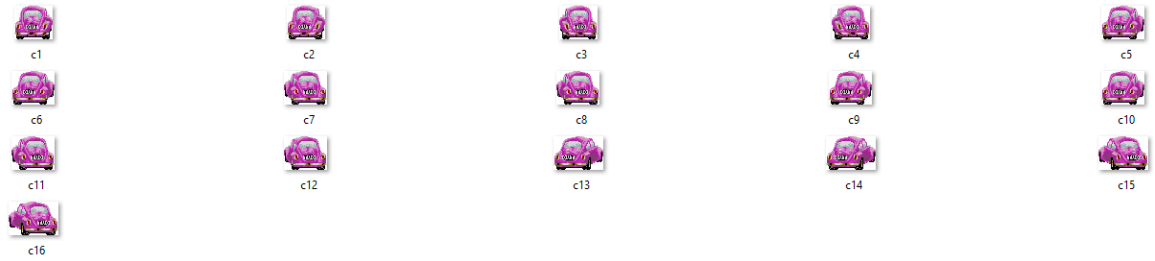
– La posición inicial se establece de manera aleatoria con un valor comprendido entre la línea del último enemigo generado + 25 (o la línea renderizada más alejada si el último enemigo generado está dentro del campo de visión) y 100 líneas más a dicho valor, por ejemplo entre 500 y 600. De esta manera se evita tanto que aparezcan enemigos de la nada dentro del campo de visión del jugador como que aparezcan múltiples enemigos en la misma línea.

– La agresividad de la *IA* se establece de manera aleatoria con un valor comprendido entre la mitad del valor de la agresividad global y el valor de la agresividad global. El tipo de *IA* se establece de manera aleatoria entre los cuatro posibles tipos disponibles.

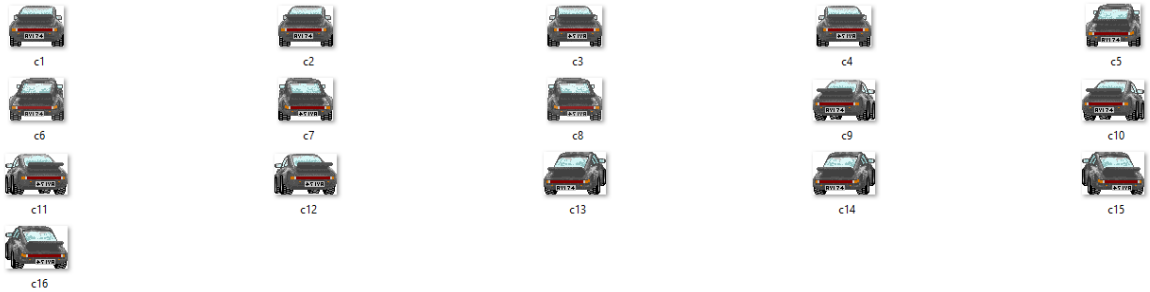
Dada la gran cantidad de vehículos de tráfico utilizados (véase figura H.2), se ha optado por incluir a modo de muestra algunos *sprite sheets* de estos tipos de vehículos.



(a) Sprite sheet del Chevrolet Corvette (1971)



(b) *Sprite sheet* del *Volkswagen Beetle (1972)*



(c) *Sprite sheet* del *Porsche 911 (1985)*

Figura H.2: Ejemplos de *sprite sheets* de vehículos de tráfico

H.2. Vehículos rivales

Los bólidos rivales son todos aquellos vehículos que corren como oponentes del jugador en los diversos eventos de competición para un solo jugador. A diferencia del

tráfico, los rivales están presentes en todos los modos de juego a excepción del *Out Run*. En la modalidad para un jugador de *World Tour*, el jugador debe enfrentarse obligatoriamente a un total de siete coches distintos. En el modo para un único jugador de *Pole Position*, el jugador puede seleccionar contra cuántos rivales correr (mínimo cero y máximo siete). En los modos de juego *Driving Fury* y *Demarrage*, el jugador debe enfrentarse a un único rival por nivel, aquel que debe golpear hasta detener o adelantar y hacer demarraje, respectivamente.

La generación de estos vehículos en los escenarios depende del modo. No es aleatoria como con el tráfico. En los modos de *World Tour* o *Pole Position* los vehículos rivales son posicionados en unas posiciones predefinidas para poder crear la parrilla de salida (véase figura H.3). En los modos *Driving Fury* o *Demarrage*, los vehículos aparecen siempre a una distancia exacta de 1000 líneas respecto del origen del escenario en que aparecen.

Para lograr una mayor expresividad gráfica, para cada vehículo rival se ha elaborado una paleta de colores específica para ese tipo de vehículo. Por consiguiente, cada tipo de coche (moto, *F1*, deportivo, todo terreno y camión) tiene una gama de color diferente al resto, a excepción del vehículo de la policía, que solo tiene un color.

Figura H.3: Ejemplo de parrilla de salida en *World Tour*

El proceso de pintado de coches fue muy laborioso pero bastante mecánico. Consistió en coger los *sprites* originales de cada videojuego y efectuar un proceso de cambio de colores. Se ha procurado que la paleta de cada coche tenga una amplía diversidad de color, con la finalidad de lograr una mayor vistosidad.

Estos vehículos pueden encontrarse en las carpetas *Devastators*, *Motorbikes*, *Minivans*, *Trucks*, *FormulaOnes* y *Police*. Dichas carpetas se localizan dentro del directorio *Vehicles*, accesible desde la carpeta principal *Data*. Dentro de cada una, salvo en la del coche policial, se pueden encontrar otros ocho directorios, donde se encuentran los *sprites* de cada uno de estos vehículos. Dada la gran cantidad de vehículos rivales posibles, y por tanto, del elevado número de *sprites*, se ha optado por un incluir para tipo de vehículo (moto, coche deportivo, todoterreno, *F1*, policía y coche de tráfico) un *sprite sheet* de muestra (véanse figuras H.4, H.5, H.6, H.7, H.8 y H.9).

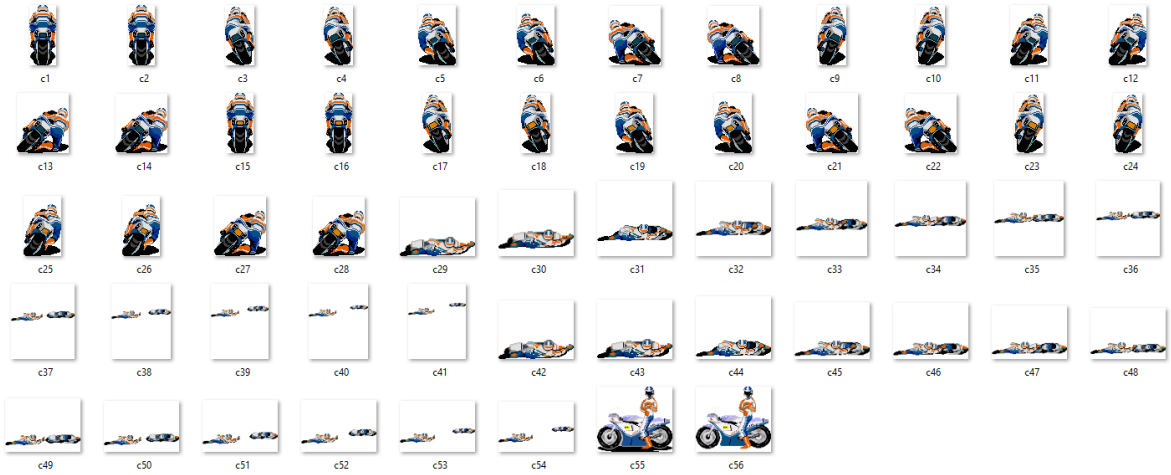


Figura H.4: *Sprite sheet* de una moto

Figura H.5: *Sprite sheet* de un coche deportivo

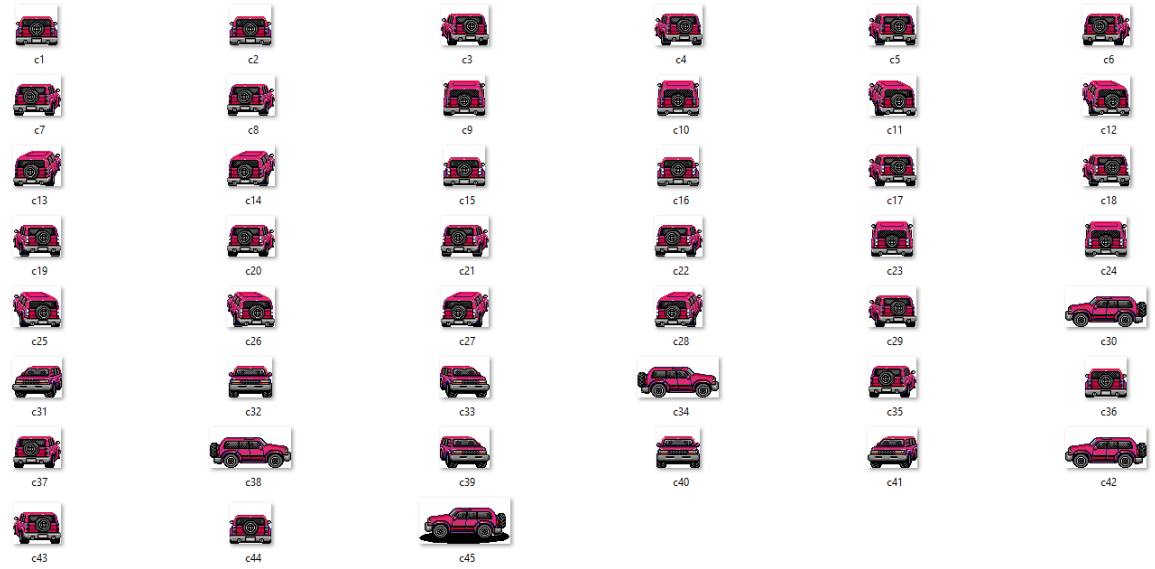


Figura H.6: *Sprite sheet* de un todoterreno

Figura H.7: *Sprite sheet* de un camión

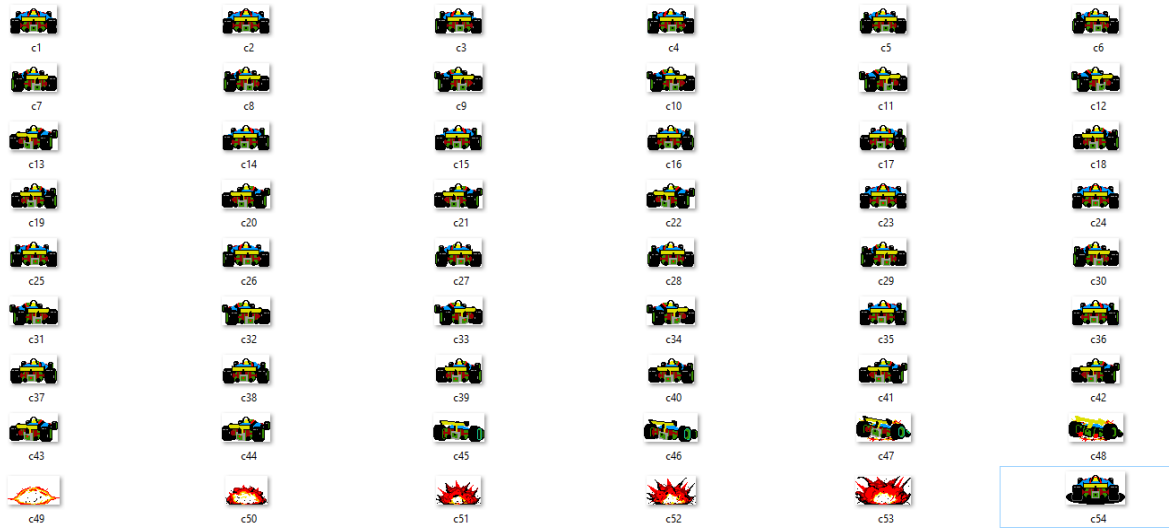


Figura H.8: *Sprite sheet* de un *F1*

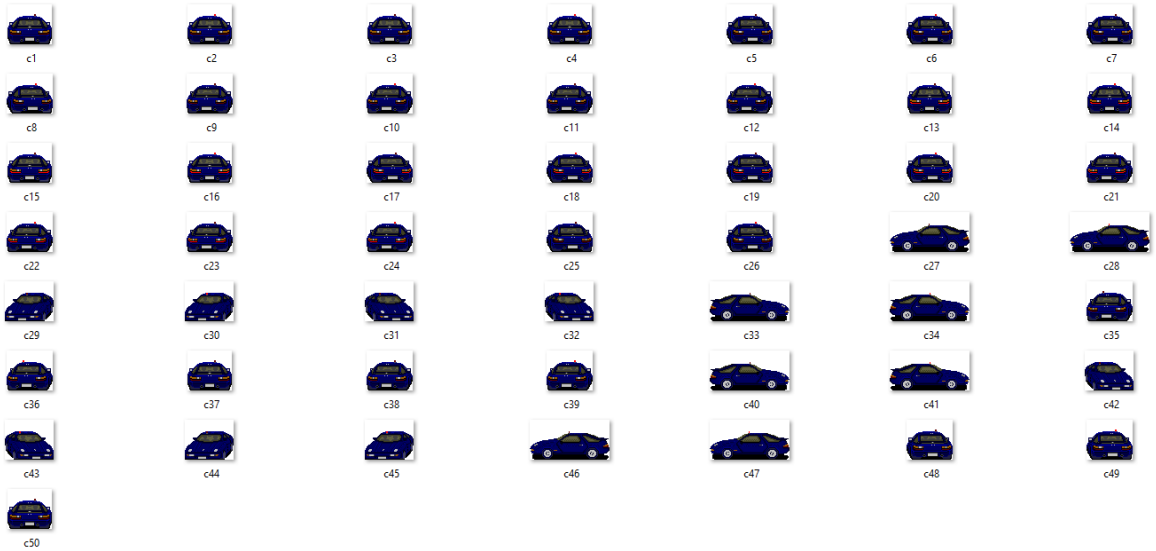


Figura H.9: *Sprite sheet* del coche de policía

Anexo I

Tratamiento de la música y el *SFX*

En este anexo se detallan las distintas canciones y *SFX* empleados en la arquitectura, como se ha cumplido con la política del *copyright*, y por último, también se explica como se ha llevado a cabo la gestión y reproducción de los sonidos (música y *SFX*) a nivel de implementación.

I.1. Pistas musicales y *SFX*

Con el objetivo de evitar cometer delito contra la propiedad intelectual, tanto en el propio programa como en este documento, se agradece y al mismo tiempo se muestra, el reconocimiento a los videojuegos de los que se han extraído las músicas y el *SFX*, reconociendo que los derechos de estas composiciones son propiedad de las compañías desarrolladoras de estos videojuegos. Seguidamente se proceden a citar los videojuegos de los que se han extraído las músicas y el *SFX* junto con las empresas desarrolladoras.

Los videojuegos de los que se han extraído las bandas musicales son los siguientes:

- *Super Hang On*, lanzado en 1989 para *Sega Megadrive*, desarrollado por *Sega AM2* y *Sega*.
- *Alien 3*, lanzado en 1992 para *Sega Megadrive*, desarrollado por *Acclaim Studios London* y *Bits Studios*.
- *Mega Turricane* (1994), para *Sega Megadrive*, desarrollado por *Factor 5*.
- *Cruis'n USA*, tanto la versión *arcade* como la creada para *N64*, ambas lanzadas al mercado en 1994. Las empresas desarrolladoras son *Miday Games* y *WMS Industries*.

– *Cruis'n World*, lanzado en 1996 para *N64*. Fue desarrollado por *Midway Games* y *Eurocom*.

– *Cruis'n Exotica*, lanzado en 1999 para *N64*. Las desarrolladoras de este videojuego son *Gratuitous Game*, *Crawfish Interactive* y *Midway Games*.

– *Mario Kart Super Circuit*, lanzado en 2001 para *Game Boy Advance*. Videojuego desarrollado por *Intelligent Systems*.

– *Mario Kart*, lanzado para *Nintendo DS* en 2005. Fue desarrollado por *Nintendo* y *Nintendo Entertainment Analysis and Development*.

Los videojuegos de los que se han extraído los efectos de sonido (*SFX*) son los siguientes:

– *Out Run*, versión para la máquina recreativa de 1986. Los desarrolladores de este juego fueron *Sega*, *Sega AM2*, *M2*, *NEC Corporation*, *Acclaim Studios London* y *SEGA Mobile*.

– *Chase HQ*, versión *arcade* lanzada en el año 1988. *Taito Corporation*, *Ocean Software*, *The Hit Squad* y *John O'Brien* son los desarrolladores de este videojuego.

– *Super Hang On*, lanzado en 1989 para *Sega Megadrive*, desarrollado por *Sega AM2* y *Sega*.

– *Golden Axe*, lanzado en 1989 para *Sega Megadrive*. Las empresas distribuidoras son *Sega*, *Bandai*, *Virgin Interactive*, *Sega Wow*, *Telenet Japan*, *Acclaim Studios London* y *SEGA Mobile*.

– *Cruis'n USA*, tanto la versión *arcade* como la creada para *N64*, ambas lanzadas al mercado en 1994. Las empresas desarrolladoras son *Midway Games* y *WMS Industries*.

– *Cruis'n World*, lanzado en 1996 para *N64*. Fue desarrollado por *Midway Games* y *Eurocom*.

– *Road Rash* (1999), creado para *N64*. Fue desarrollado por *Pacific Coast Power* y *Light*.

– *Le Tour de France*, lanzado en 2002 para *PS2*, desarrollado por *Konami*.

– *Mario Kart*, lanzado para *Nintendo DS* en 2005. Fue desarrollado por *Nintendo* y *Nintendo Entertainment Analysis and Development*.

– *Mario Kart Wii*, lanzado para *Wii* en 2008. *Nintendo* fue el desarrollador de este juego.

– *Mario Kart Tour*, lanzado para *Android* e *iOS* en 2019. Los desarrolladores de este videojuego son *Nintendo*, *DeNA*, y *Nintendo Entertainment Analysis and Development*.

Todas las canciones y *SFX* citados fueron escogidos por ser muy variados, y además, por encajar muy bien con la filosofía que pretende transmitir *Multi Race Driving* a los jugadores.

A modo de tributo a los videojuegos originales, para las canciones empleadas durante las partidas se ha decidido conservar sus nombres originales. A continuación, se citan por orden de reproducción, las piezas musicales junto con sus compositores y productores.

– *Redline Shuffle*: compuesta por *Vince Pontarelli* y producida por *Orpheus Hanley*.

– *House Special*: compuesta por *Vince Pontarelli* y producida por *Orpheus Hanley*.

– *Winning Road*: compuesta por *Recruit 0*, *Kōichi Namiki* y *Katsuhiro Hayashi*. Producida por la discográfica *Data Discs*.

– *Bluegrass Boggie*: compuesta por *Vince Pontarelli* y producida por *Orpheus Hanley*.

– *RoadKill Jam*: compuesta por *Vince Pontarelli* y producida por *Orpheus Hanley*.

– *Egypt*: compuesta por *Vince Pontarelli* y producida por *Steve Duckworth*.

– *Sprinter*: compuesta por *Kōichi Namiki* y *Katsuhiro Hayashi*. Producida por la discográfica *Data Discs*.

– *Funky sea*: compuesta y dirigida por *Vince Pontarelli*.

– *Booty Shaker*: compuesta y dirigida por *Vince Pontarelli*.

– *Euro House*: compuesta y dirigida por *Vince Pontarelli*.

I.2. Aspectos de implementación

Para poder agilizar el tratamiento de las bandas sonoras y de los efectos de sonido, se ha tomado la decisión de gestionarlos por separado. De este modo, los efectos de sonido están agrupados por un lado y las bandas musicales por otro. Pueden encontrarse en los directorios *SoundEffects* y *Soundtracks* respectivamente, localizados dentro de la carpeta principal de archivos de la plataforma, *Data*.

En cada una de estas carpetas, hay dos subdirectorios para separar la configuración de las pistas y efectos de los propios mismos. Así, la carpeta *Soundtracks* contiene un subdirectorio denominado igual, *Soundtracks*, donde se recogen todas las pistas musicales, y una carpeta denominada *Configuration*, donde reside un fichero de configuración *XML* donde se hallan recogidos todos las bandas musicales que la arquitectura debe utilizar. De igual modo ocurre con los efectos de sonido, es decir, existe una subcarpeta llamada *SoundEffects* y otra denominada *Configuration*, ambas dentro del directorio *SoundEffects*, donde están los archivos de efectos de sonido y la configuración sobre cuáles tiene que usar el programa respectivamente.

Multi Race Driving cuenta con un total de 23 pistas musicales y 127 efectos de sonido. Todos ellos están en formato *Ogg*[1] (*códec* vorbis). Existen numerosas razones de peso por las que se ha optado por escoger este formato de audio frente a otros como por ejemplo *Flac*.
En mayor detalle, las principales razones han sido que *Vorbis* es un *códec* muy recomendable por su alto grado de eficiencia en el *streaming* y en la compresión de archivos, factor que es muy importante ya que hay bandas musicales, aunque no todas, que se ejecutan de manera dinámica e interactiva, al igual que todo el *SFX*, ya que depende tanto de los eventos que ocurren en el escenario (sonidos del semáforo, colisiones de los coches, derrapes) como de acciones controladas por el propio jugador (arrancar y manejar el coche). Además, *Ogg* es un formato de contenedor multimedia de uso libre y soportado por la *API* de *SFML* a diferencia de otros como *MP3*[2].

Para poder reproducir la música y el *SFX* se ha diseñado un módulo interno responsable tanto de cargar todas las canciones y efectos en el momento de arranque de *Multi Race Driving* como de reproducirlos cuando éste último está en ejecución.

[1] `https://es.wikipedia.org/wiki/Ogg_Media`
[2] `https://www.sfml-dev.org/tutorials/2.5/audio-sounds.php`

La carga de las pistas y efectos se efectúa leyendo sendos ficheros de configuración *XML*. El proceso de carga se hace aplicando *multithreading* de manera que existe un hilo encargado de la lectura y carga de los efectos de sonido y otro responsable de lo mismo pero con las pistas musicales. De este modo, el proceso de carga es inmediato. Los efectos y músicas se hallan identificadas en sus respectivas carpetas de archivos con un identificador correlativo para facilitar el proceso de lectura. Los ficheros de configuración *XML*, por su parte, son listas donde se encuentran las rutas de todas las pistas y efectos a cargar. A continuación, se muestra el fichero *XML* que contiene la lista de pistas musicales:

```
<?xml version="1.0" encoding="UTF-8"?>
<SoundTracks>
  <S1>Data/Soundtracks/Soundtracks/1.ogg</S1>
  <S2>Data/Soundtracks/Soundtracks/2.ogg</S2>
  <S3>Data/Soundtracks/Soundtracks/3.ogg</S3>
  <S4>Data/Soundtracks/Soundtracks/4.ogg</S4>
  <S5>Data/Soundtracks/Soundtracks/5.ogg</S5>
  <S6>Data/Soundtracks/Soundtracks/6.ogg</S6>
  <S7>Data/Soundtracks/Soundtracks/7.ogg</S7>
  <S8>Data/Soundtracks/Soundtracks/8.ogg</S8>
  <S9>Data/Soundtracks/Soundtracks/9.ogg</S9>
  <S10>Data/Soundtracks/Soundtracks/10.ogg</S10>
  <S11>Data/Soundtracks/Soundtracks/11.ogg</S11>
  <S12>Data/Soundtracks/Soundtracks/12.ogg</S12>
  <S13>Data/Soundtracks/Soundtracks/13.ogg</S13>
  <S14>Data/Soundtracks/Soundtracks/14.ogg</S14>
  <S15>Data/Soundtracks/Soundtracks/15.ogg</S15>
  <S16>Data/Soundtracks/Soundtracks/16.ogg</S16>
  <S17>Data/Soundtracks/Soundtracks/17.ogg</S17>
  <S18>Data/Soundtracks/Soundtracks/18.ogg</S18>
  <S19>Data/Soundtracks/Soundtracks/19.ogg</S19>
</SoundTracks>
```

Para garantizar un manejo óptimo de los sonidos y del *SFX* se ha consultado la documentación de *SFML*. Esta *API* da soporte a la manipulación de audio por medio de dos tipos de datos predefinidos, *Sound* y *Music*. Ambos hacen la misma tarea pero de forma diferente, pues *Sound* para reproducir los sonidos debe cargarlos en memoria desde el archivo en cuestión mientras que *Music* abre el fichero y reproduce el sonido en *streaming*, es decir, es mucho más rápida y eficiente esta gestión, sobre todo, si se van a reproducir sonidos largos o repetidos de forma periódica. Por consiguiente, se ha usado *Music* para poder gestionar internamente tanto el audio como el *SFX*. Además, dicha gestión ha sido dinámica, recurriendo a los *Smart Pointers* de *C++*, que permiten la gestión automática de la memoria abstrayendo completamente al programador de ciertas tareas como el borrado y la liberación de ésta.

En cuánto a las pistas musicales, se pueden distinguir dos grupos, las pistas dinámicas y las pistas estáticas. Las pistas dinámicas son aquellas que pueden ser cambiadas por el jugador mientras el *software* está ejecutándose. Dichas bandas sonoras son las que suenan durante la partida. El jugador por defecto puede cambiarlas con la tecla *Z*, aunque puede cambiarla desde el menú de selección de controladores si lo desea. Cuando el jugador pulsa el controlador para cambiar la pista musical, se muestra en la parte superior central de la pantalla durante un periodo breve de tiempo el título de la pieza (véase figura I.1). Las pistas estáticas son aquellas que no pueden controlarse, como por ejemplo la de la animación introductoria, la de la pantalla de carga o las músicas de los menús de la *GUI*.

Figura I.1: Cambio de pista musical

La reproducción de las pistas musicales y del *SFX* se realiza en paralelo, es decir, de manera concurrente. Sin embargo, este paralelismo no ha tenido que ser modelado por el desarrollador ya que la *API* de *SFML* lo proporciona a los usuarios.

Anexo J

Diseño del modo multijugador

En este anexo se procede a describir en profundidad cuáles han sido las diferentes etapas en las que se ha estructurado el proceso de construcción del modo multijugador. Para cada una de estas fases se han explicado los objetivos a conseguir y cuáles han sido los pasos seguidos para poder abordarlos. Las etapas por la que ha pasado el diseño del modo multijugador son las siguientes.

J.1. Despliegue del servidor

La primera fase del desarrollo del modo multijugador ha sido la habilitación de un servidor que sirva de soporte físico para que *Boreas* pueda ejecutarse. El proceso de cómo poder montar el servidor no fue tomado a la ligera, de hecho, se discutieron varias alternativas sobre cómo proceder.

La primera opción era la de montar un servidor propio local que fuese gestionado en todo momento por el desarrollador del proyecto. Sin embargo, no era una opción ventajosa ya que el desarrollador no tenía recursos propios suficientes para poder tener el servidor disponible de forma ininterrumpida. Por ello, se escogió la segunda opción, que consistía en recurrir a un servicio externo de *cloud computing*, concretamente *AWS* (*Amazon Web Services*) para poder alojar el *framework*. De este modo, el desarrollador se veía completamente exento de tener que velar por su manteimiento y seguridad. Pues es *Amazon*, quién se encarga de promover tales servicios a los usuarios. Además, otra ventaja de esta perspectiva es que está en la nube, lo cuál, lo hace mucho más sofisticado.

Las fases que se han seguido para poder abordar el despliegue del servidor son las siguientes:

– El primer paso ha sido habilitar, usando la cuenta de correo de la universidad y los recursos proporcionados por la extensión de *GitHub* para estudiantes, *GitHub Student Developer Pack*[1], el acesso al *cloud* de *AWS*.

– Una vez accedido al *cloud* de *AWS* se ha adquirido una instancia EC2[2] de *Amazon* con sede geográfica en *Virginia*, *EEUU*, con una dirección *IP* dinámica, un dominio por defecto, y un fichero donde se almacena la clave pública de la máquina para poder conectar con ella. La razón por la que se ha escogido esta instancia es debido a que, de acuerdo al crédito proporcionado por la universidad, era la alternativa económica más aceptable.

– Seguidamente se ha procedido a configurar la máquina virtual de *EC2*. Para ello, se ha configurado el *firewall* de la máquina para que acepte peticiones *TCP*, conexiones *SSH*, y además, se ha abierto otro puerto donde *Boreas* escucha las peticiones de los clientes de los jugadores.

– El siguiente paso ha sido asignar a la máquina una *IP* estática, y posteriormente, adquirir un dominio gratuito propio para usarlo como alternativa al proporcionado por *Amazon*. Se ha actualizado el campo '*A*' del registro *DNS* del dominio con la dirección *IP* de la instancia de *AWS*. De este modo, se protege el tipo de instancia utilizada ya que, ni se usa la *IP*, ni se usa el dominio por defecto, que evidencia a terceros que es una máquina *EC2*.

– A continuación, se ha establecido conexión con la instancia a través de la clave privada proporcionada por *Amazon*, y una vez dentro se ha procedido a instalar el *software* necesario para poder ejecutar *Boreas* en ella. Se ha instalado *g++* y *make* para poder compilarlo, y posteriormente, se ha instalado *Git*[3] para poder clonar el repositorio de *Boreas* de *GitHub*.

– Tras clonar el repositorio y compilar el código en la máquina virtual *EC2* se ha procedido a activar el *log* de *Boreas*, para poder disponer de un registro de toda la actividad procesada por el *framework*.

– A continuación, dado que *Boreas* es un *framework* que guarda todos los datos en memoria, para evitar excesos de información, se ha configurado un proceso de limpieza automático que lo resetea todos los días a las 5 A.M.

[1] `https://education.github.com/pack/offers#aws-educate`
[2] `https://aws.amazon.com/es/ec2/`
[3] `https://git-scm.com/`

– Finalmente, se han realizado varios programas de prueba para poder comprobar la operatividad y comunicación con el servidor desplegado.

J.2. *Framework Linda*

Linda es un sistema de coordinación muy útil en los sistemas distribuidos. Fue diseñado por *David Gelernter*[4], en 1985. El funcionamiento se basa en la existencia de un espacio lógico de memoria compartida (espacio de tuplas), usado por los procesos para comunicarse. Los procesos se comunican dejando tuplas en el espacio, consultando la existencia de tuplas en el espacio, o retirando de forma selectiva tuplas del espacio.

Las tuplas son definidas como listas de datos planas. Permiten representar secuencias de valores concretos como por ejemplo `["Juan","45","34","88"]`, y al mismo tiempo patrones, es decir, tuplas en las que pueden aparecer variables (véase figura J.1).

El espacio de tuplas detecta las tuplas pedidas por los procesos recurriendo al uso de un algoritmo de *pattern matching*[5]. El algoritmo compara la dimensión de la tupla pedida con cada una de las tuplas del espacio. Si son iguales en tamaño, se comparan las componentes que tomen valores concretos para determinar la coincidencia. Aquellas componentes que codifican patrones son ignoradas. De este modo, se chequean la dimensión y componentes fijas de las tuplas, y las que son variables se sustituyen.

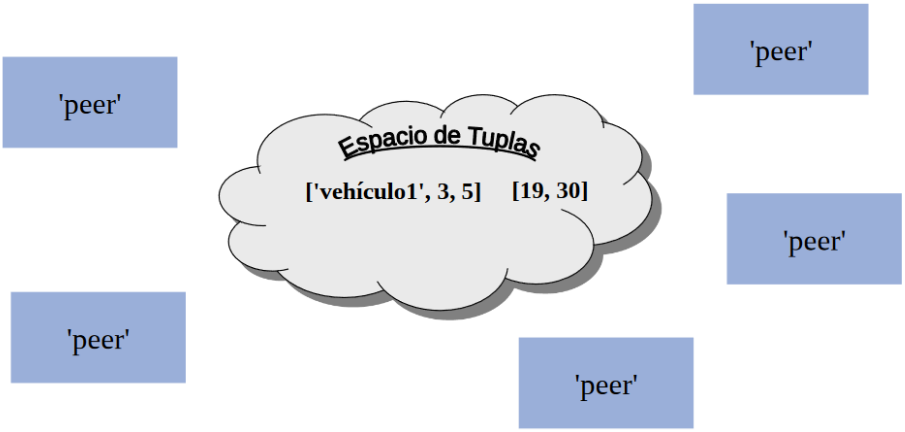


Figura J.1: Modelo de coordinación *Linda*

[4] https://en.wikipedia.org/wiki/David_Gelernter
[5] https://www.genbeta.com/desarrollo/pattern-matching

A continuación, se especifican la operaciones disponibles sobre el espacio de tuplas, todas ellas atómicas[6].

– *Postnote*(`[exp1, exp2, ..., expN]`): también conocida como *Out* o *Write* (véase figura J.2). Mediante esta operación el proceso deposita una tupla en el espacio de tuplas y sigue. Es una operación no bloqueante.

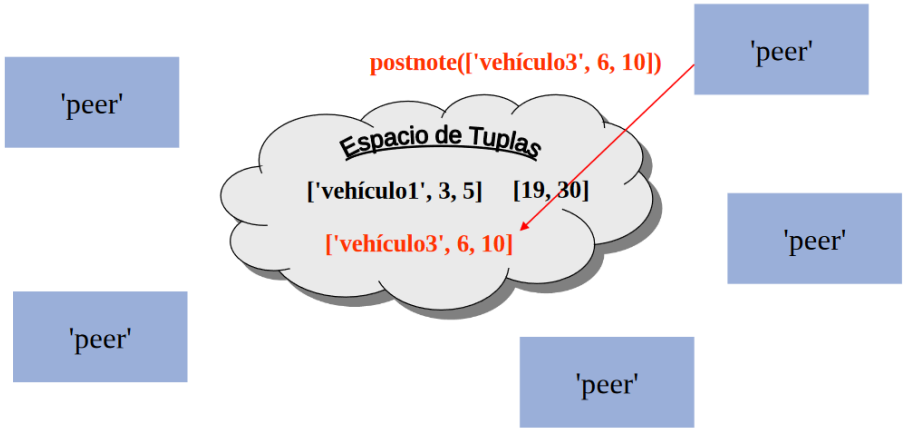


Figura J.2: Operación *Postnote*, *Out* o *Write* de *Linda*

– *Removenote* (`[exp1, exp2, ..., expN]`): también definida con los sobrenombres *In* o *Take* (véase figura J.3). El proceso permanece bloqueado hasta que el espacio de tuplas le asigna una tupla que coincida con el patrón propuesto. La tupla es quitada del espacio de tuplas.

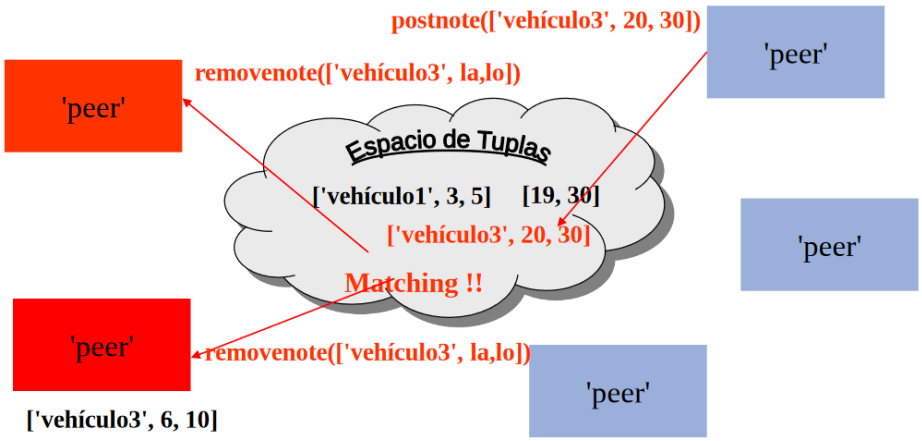


Figura J.3: Operación *Removenote*, *In* o *Take* de *Linda*

[6] `https://es.wikipedia.org/wiki/Instrucci%C3%B3n_at%C3%B3mica`

– *Readnote* (`[exp1, exp2, ..., expN]`): conocida de igual modo como *Read* (véase figura J.4). Esta operación es una versión de la operación anterior en la que la tupla asignada no se elimina del espacio de tuplas.

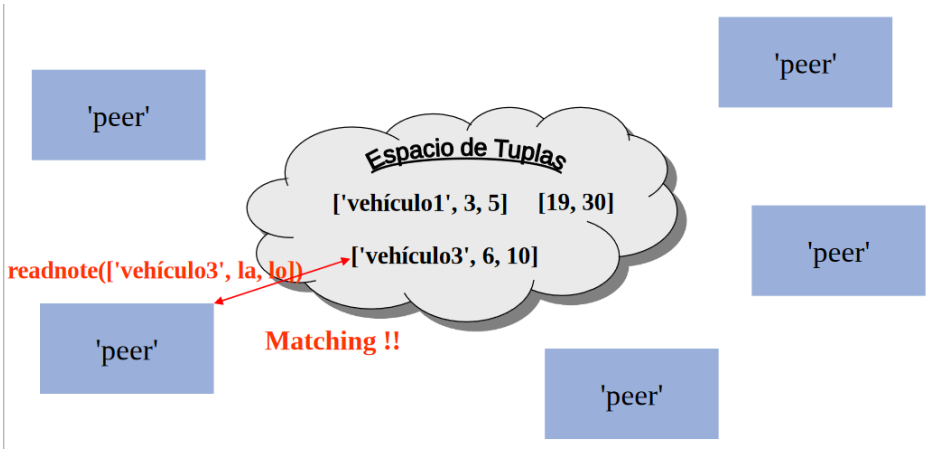


Figura J.4: Operación *Readnote* o *Read* de *Linda*

Aparte de las operaciones anteriores, para dar una mayor usabilidad de *Boreas* por parte de *Multi Race Driving*, se ha implementado una operación adicional, denominada *ReadnoteX*. Esta operación es una versión de *Readnote* en la que si no hay ninguna tupla que coincida con el patrón deseado por el proceso, recibe una tupla con formato `['NOT_FOUND']` por parte del espacio de tuplas y sigue su ejecución en vez de bloquearse como ocurría en el *Readnote*. Esta operación ha sido implementada con el fin de maximizar la velocidad de comunicación entre los jugadores y el servidor ya que no consume ningún recurso de ninguno de los dos, obtiene resultados inmediatos, y además, facilita enormemente la concurrencia.

J.3. Ventajas de *Boreas*

La utilización del *framework Linda* como mecanismo de sincronización y comunicación en el modo multijugador fue una decisión fácil de tomar. Sin embargo, no se tenía muy claro, si programar una versión de cero para *Multi Race Driving* o recurrir a una implementación externa. Tras un minucioso análisis de los pros y los contras de ambos puntos de vista, se decidió proseguir por el segundo camino. Existían diversas versiones candidatas a emplear, siendo *Boreas* la finalmente seleccionada.

La principal razón por la que se ha usado esta implementación ha sido que el desarrollador de *Multi Race Driving* fue también uno de los desarrolladores que

intervinieron en la creación de *Boreas*. Por ello, aspectos como la arquitectura interna del sistema, el almacenamiento interno de la información o la gestión de la concurrencia eran ya conocidos. Otro motivo es que el almacenamiento de los datos es distribuido y está muy optimizado ya que toda la información se guarda en memoria, por lo que la lectura y escritura de las tuplas es muy rápida. Por último, *Boreas* es un sistema muy robusto y eficiente, pues fueron muchas las pruebas a las que se sometió. Una de ellas fue, haciendo uso de un algoritmo de fuerza bruta, cargar el espacio de tuplas con el fin de acotar superiormente el límite de almacenamiento del sistema. Los resultados de esta prueba fueron muy alentadores ya que la experiencia duró 2h y 15 minutos y fue capaz de guardar un total de 30.000.000 de tuplas aproximadamente. Esta prueba es un signo claro de la potencia de diseño de esta implementación. Dichos resultados han sido extraídos del documento de diseño de *Boreas*[7].

J.4. *Test* de velocidad de conexión al servidor

Una parte muy importante en el desarrollo de cualquier juego multijugador es la latencia que experimenta la red a la hora de enviar el tráfico. Del mismo modo, también es conveniente saber que la latencia no es un problema que afecta a todos los clientes (jugadores) de igual modo, pues se verán más afectados aquellos que estén situados a más distancia del servidor que aquellos que están más cerca.

Dada la existencia de un único servidor centralizado, como medida preventiva a este problema latente, cuando el jugador selecciona el modo multijugador, se procede a realizar un *test* de velocidad para poder comprobar la calidad de conexión del cliente con el servidor remoto. Tras efectuarse el *test* de velocidad de conexión, se le muestra al jugador un resumen con las estadísticas de dicho *test*, y al mismo tiempo, se le solicita confirmar su deseo de jugar. La diferencia con respecto a la primera selección es que en ésta se dispone de los datos proporcionados por el *test*.

Para la realización del testeo de conexión se ha aplicado una vez más *multithreading*. Se lanza un proceso auxiliar que se encarga de realizar este *test* mientras que el hilo de ejecución principal muestra al jugador la animación correspondiente a que está teniendo lugar el *test*. El *test* de velocidad de conexión es muy simple, consiste en hacer *ping*

[7] `https://www.naval.cat/opendelta/?=X1mlWr0p9gPXrZ6qYEbq1A==`

al servidor remoto y obtener los tiempos de latencia. Dichos tiempos en cuestión, son los resultados que se le proporcionan al jugador en la pantalla tras concluir el testeo.

Ping es un comando *shell* que, por defecto, devuelve siempre la salida por terminal. Para asegurar la transparencia de todo este proceso, el hilo encargado de ejecutar el *test* crea un proceso hijo y una tubería o *pipe* a la cuál se redirige toda la salida del comando *ping*. El hilo ejecutor del *test* escribe los resultados en la tubería y el proceso hijo los lee y parsea para poder encontrar los tiempos de latencia. Además, dado que la salida del comando *ping* puede recibirse en distintos idiomas (depende de los ajustes de idioma y región del computador de cada cliente), se ha pensado un método de parseo que no tiene en cuenta el idioma de la salida para hallar la información. Así las cosas, el proceso siempre va a estar funcional independientemente del idioma. Además, cabe destacar, que este proceso tiene lugar de manera completamente transparente al jugador.

J.5. Estudio de tiempos

El tiempo medio de latencia a la hora de realizar peticiones al servidor remoto de *Amazon* suele estar situado en torno a los 100 ms. La siguiente gráfica comparativa (véase figura J.5) muestra un análisis de los valores de latencia mínima, media y máxima obtenidos en un total de 20 experiencias.

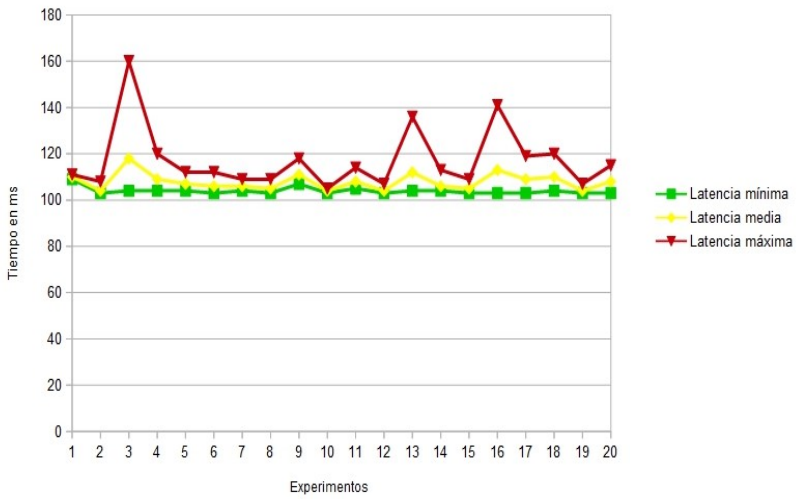


Figura J.5: Estadísticas de los *tests* de velocidad al servidor remoto

En vistas al gráfico anterior, se puede observar que no hay una diferencia desmesurada entre las tres magnitudes, aunque se ha tenido en cuenta que latencias del orden de 100 ms son bastante significativas en cuánto al posible *laggeo* que se pueda experimentar. Además, cabe destacar que como protocolo de transmisión se ha usado *TCP*.

El problema del modo multijugador ha residido fundamentalmente en el envío y recepción de las posiciones y estados de los vehículos de los jugadores durante las carreras. Para abordar este problema se plantearon dos posibles alternativas. El primer enfoque es sencillo ya que consistía en notificar los cambios del estado de los jugadores de manera continua, mientras que el segundo consistía en notificar los cambios de los jugadores cada cierto tiempo (muy pequeño).

El problema del primer enfoque es obvio, cuánta más información se envíe más retardo y posible congestión de tráfico se va a tener, pero tiene una ventaja que es clave, la información es muy exacta porque se conoce casi en todo momento dónde están los coches y su estado (frenado, giro, colisión).

La segunda idea, por su parte, tiene la ventaja de que garantiza un mejor uso de la red, a costa de que entre recepción y recepción, no se tiene información de qué le ocurre a cada jugador. Siguiendo este enfoque, el tiempo entre notificaciones debe ser adecuado para que la información se envíe bien. Si se mandan los datos más rápido de lo que la red puede transportarlos los paquetes pueden ser mezclados unos con otros, derivando en fallos de sincronización. Para poder solventar este problema habría que poner un tiempo de espera suficiente. Sin embargo, este tiempo deriva en otro problema, y es que todos los eventos ocurridos entre notificaciones se perderían (frenados, giros, colisiones, etcétera). Sería necesario desarrollar un algoritmo que pudiese predecir la trayectoria y las acciones de los jugadores durante esos transcursos de tiempo, lo cual es prácticamente imposible.

Ante este problema, se ha optado por recurrir a la primera opción, y además, se ha procurado que la información a enviar (los paquetes) sean lo más concisos y significativos posibles con el fin de agilizar su procesamiento y reducir la posible congestión al máximo.

Otro aspecto importante es si el protocolo *TCP* ha sido la opción más acertada. En la mayoría de juegos *online* actuales se emplea el protcolo *UDP* en vez del protocolo *TCP* dado que no es un protocolo orientado a conexión, y prima la velocidad de envío

de los datos antes que la seguridad de estos. Por ello, situaciones como paquetes con alteraciones en los datos, paquetes recibidos en distinto orden al de envío o incluso paquetes no recibidos no son controlados por el protocolo mientras que *TCP*, al ser un protocolo de transmisión fiable y seguro ya tiene todos estos casos en cuenta, dejando exentos a los programadores de tener que diseñar rutinas de servicio que los traten. Así las cosas, si se hubiese empleado *UDP* la información viajaría más rápido pero hubiese sido necesario introducir muchas comprobaciones en los clientes precisamente para evitar todas las situaciones anteriormente citadas.

J.6. Sincronización y comunicación en el modo multijugador

En esta sección se procede a explicar los protocolos de comunicación que se han desarrollado para poder confeccionar el modo multijugador. Dicha modalidad se halla compuesta por varios puntos críticos en los que se ha tenido que asegurar un correcto control de la sincronización para que el sistema esté en todo momento operativo.

Una clave a la hora de montar el modo multijugador ha sido recurrir al uso de *multithreading* para separar la parte de muestreo de la información en pantalla de la comunicación con la red. El modo multijugador es un proceso que atraviesa distintas fases en las que los datos compartidos entre los jugadores son muy variados. Con vistas a agilizar esta comunicación y a mantener la propiedad de modularidad del modo multijugador, cada jugador crea un hilo de ejecución auxiliar para controlar el envío y/o recepción de datos a través de la red al comienzo de cada una de las fases, explicadas a continuación, y lo destruye al final de éstas. De este modo, aunque se crean y eliminan muchos hilos, las tareas que realizan son distintas, mucho más pequeñas, y además, todos ellos son disjuntos ya que se crean al comienzo de fase y se eliminan al final de cada una. Estos aspectos hacen que la sincronización resultante sea mucho más ágil y fácil de diseñar que si por ejemplo se hubiese hecho con un único hilo para cada jugador que gestionase toda la actividad. Así las cosas, cada jugador posee un hilo encargado de mandar y recibir datos a través de la red, el cuál está sincronizado con el hilo principal, encargado de mostrar dicha información en pantalla.

A continuación, se procede a explicar por orden cronológico las distintas etapas en las que se ha divido el diseño del modo multijugador, describiendo en cada una de ellas el estado del sistema y que información se intercambian los jugadores a través de las

tuplas. Además, para cada tupla se explica su formato y la semántica de cada uno de sus elementos. Antes de comenzar a explicar las distintas fases en profundidad, se enfatiza en el hecho de que salvo durante el transcurso de la partida, el botón *X* de la ventana del programa, no está disponible mientras se lleva a cabo todo el proceso de formación y validación de los grupos con el fin de poder simplificar la sincronización. Siempre que el jugador esté en una fase que no sea la propia carrera y pulse el icono *X* de la ventana, se le notificará que esa acción en ese momento no está disponible.

J.6.1. Fase de grupos

Esta es la primera fase del modo multijugador. En primer lugar cada jugador debe seleccionar si desea unirse a un grupo o crear uno nuevo. Tras esta selección, cada participante debe introducir un alias o apodo con el que va a ser identificado tanto por el sistema como por el resto de miembros del grupo en el que se introduzca (véase figura J.6). Los apodos solo pueden ser combinaciones de hasta ocho caracteres, estrictamente alfanuméricos.

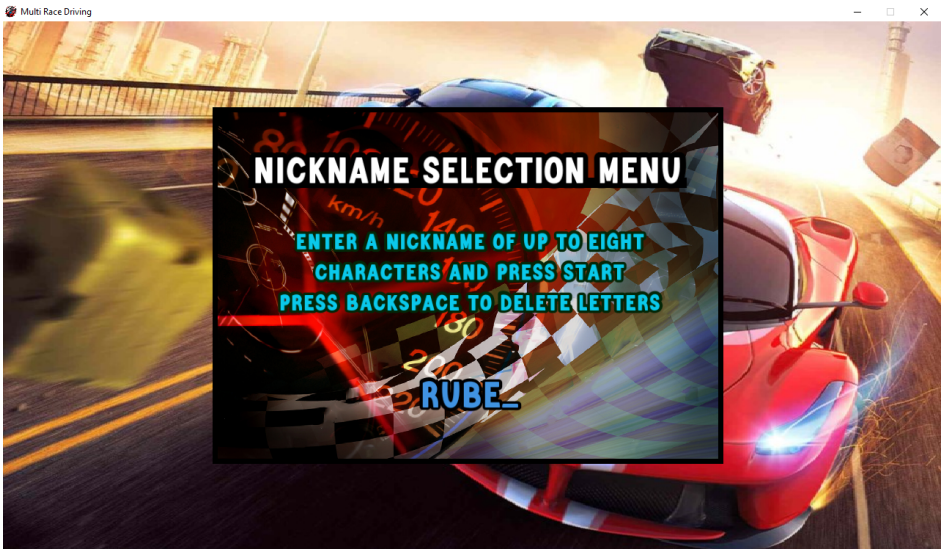


Figura J.6: Jugador validando apodo en modo multijugador

Cuando el jugador introduce el apodo, el proceso de red del jugador invoca a la operación *ReadnoteX* con la tupla `['NAME_PLAYER', nickNamePlayer]`. Así, se busca en el sistema si hay algún jugador que tenga un apodo idéntico al patrón nickNamePlayer (variable que guarda el apodo). En caso de no haber ningún jugador con un alias igual en el espacio de tuplas, el sistema le responde al proceso con la tupla `['NOT_FOUND']`. Si se recibe esa tupla, el proceso postnotea una tupla con el siguiente

formato: `['NAME_PLAYER', nickNamePlayer]`. Esta tupla representa para el sistema la presencia de un nuevo jugador con apodo válido. La primera componente indica al sistema que hay un nuevo jugador vigente y la segunda es un patrón que codifica el apodo en cuestión. En caso de no recibir la tupla `['NOT_FOUND']` se le solicita al jugador un nuevo apodo (véase figura J.7).

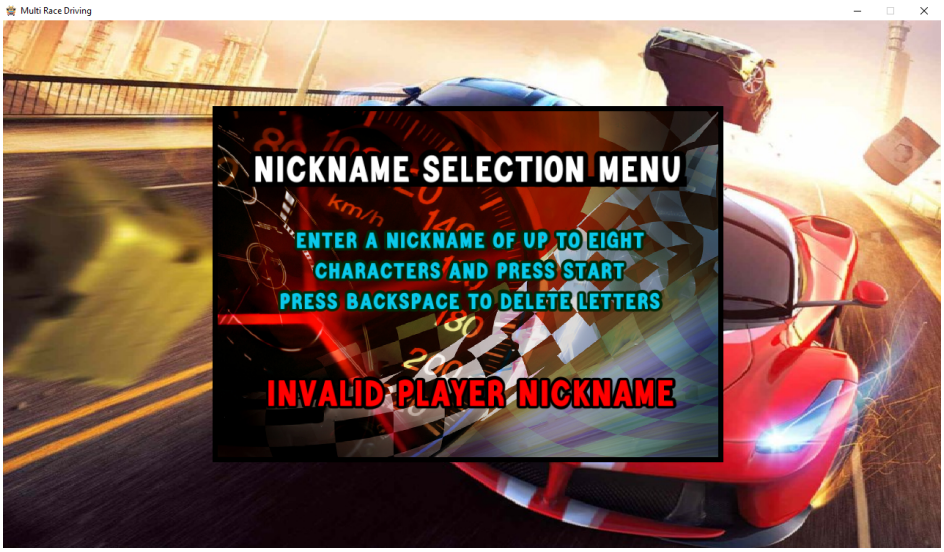


Figura J.7: Error en la validación de apodo de jugador

Tras la validación del apodo, el jugador debe introducir el nombre del grupo a crear si y solo si seleccionó la orden de crear grupo (véase figura J.8). La validación de grupo es muy similar a la validación del jugador. Primero se debe introducir el apodo del grupo (máximo ocho caracteres alfanuméricos). Tras introducirlo, el proceso de red del jugador ejecuta la operación *ReadnoteX* con la tupla `['NAME_GROUP', nicknameGroup]` para verificar si ya existe un grupo con el apodo solicitado. Si existe, se le solicita un nuevo nombre de grupo (véase figura J.9). Si no existe, el sistema devuelve al proceso la tupla `['NOT_FOUND']`, y entonces, postnotea las tuplas siguientes:

`['NAME_GROUP', nicknameGroup]`: esta tupla representa un nuevo grupo en el sistema. El primer elemento simboliza un nuevo grupo vigente y el segundo el patrón que codifica el alias de dicho grupo.

`['CREATE_GROUP', nicknameGroup, 1, 'TRUE']`: esta tupla marca la creación de un nuevo grupo al que se pueden unir más integrantes. El primer elemento marca un nuevo grupo creado con éxito, el segundo marca el nombre del grupo, el tercero refleja el número de integrantes en el momento de la creación (solo está el propietario), y el cuarto representa que el grupo todavía esta abierto para que se unan más miembros.

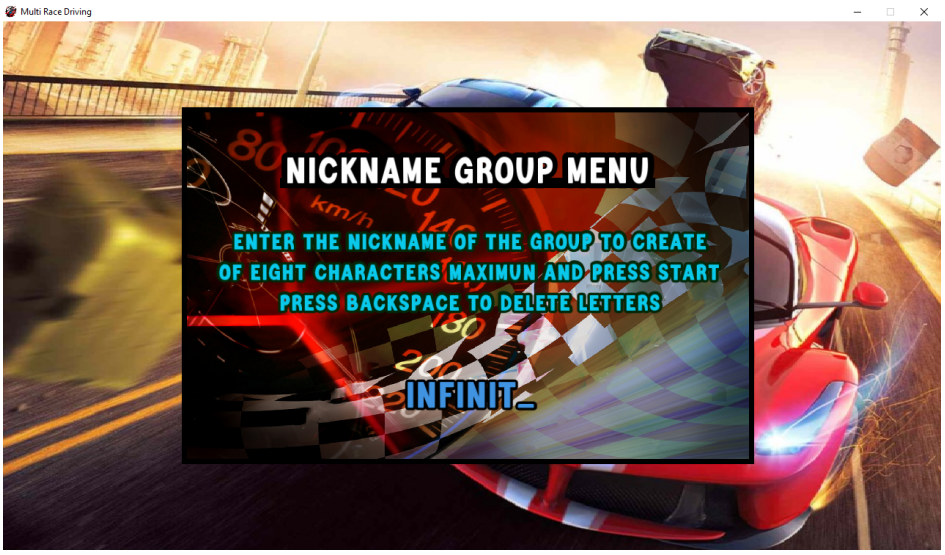


Figura J.8: Jugador validando apodo de grupo en modo multijugador

Figura J.9: Error en la validación de apodo de grupo

En el caso de que el jugador escogiese la opción de unirse a grupo, tras validarse su apodo, se le solicita si se desea unir a un grupo concreto o un grupo aleatorio. Si selecciona un grupo concreto se le solicita que introduzca el apodo del grupo en cuestión al que quiere unirse y el proceso postnotea la siguiente tupla `['JOIN_GROUP', nickNameGroup, numPlayers, nickNameplayer]`. Esta tupla representa la petición de un jugador para poder unirse a grupo con un nombre concreto. La primera componente indica la solicitud de petición, la segunda indica el nombre del grupo, la tercera el número de miembros que actualmente tiene ese grupo y la cuarta es el apodo del jugador que se desea unir. Por el contrario si se desea unir aleatoriamente, el proceso ejecuta un *Removenote* sobre la tupla`['CREATE_GROUP', nicknameGroup, numberPlayers, 'TRUE']` para buscar el primer grupo válido que admita todavía

más jugadores. Tras mandar la petición, si no se tiene respuesta, el jugador puede optar o bien por seguir esperando hasta que su solicitud sea aceptada o cancelarla y probar suerte con otro grupo pulsando la tecla *ESC*. Si su petición es aceptada, para facilitar la formación del grupo, se ha prohibido que el jugador pueda cancelar su solicitud si ya ha sido confirmada por el propietario, es decir, solo puede cancelarla si todavía no ha sido validada por el propietario. Si el jugador intenta hacer está acción se le retroalimenta por pantalla la respuesta (solicitud cancelada o petición inválida porque ya está confirmado en el grupo).

El proceso de red del propietario, por su parte, permanece bloqueado a la espera de recibir peticiones con el formato de tupla anterior ejecutando la orden *Removenote*. Es el algoritmo de *pattern matching* el encargado de llevar a cabo la sustitución de las variables por sus valores, logrando así la sincronización.

Cuando el hilo de red del propietario recibe la petición de un jugador, evalúa si en el grupo todavía hay vacantes. En tal caso le manda la tupla `['ACCEPTED_PLAYER', nickNameGroup, numPlayers, nickNameplayer]` para indicar que el jugador ha sido aceptado. El proceso de red del jugador que se desea unir, por su parte, permanece bloqueado haciendo *Removenote* sobre esa misma tupla. La primera componente de la tupla indica petición aceptada, la segunda componente es el alias del grupo, la tercera es el número de jugadores actualizado, y la cuarta es el nombre del nuevo miembro del grupo.

Paralelamente, el proceso del propietario informa a los procesos de los miembros restantes del equipo de la incorporación del nuevo jugador, mandando para cada uno una tupla con el formato `['PLAYER_GROUP', nickNameGroup, numPlayers, nickNameplayer, newPlayerGroup]` donde la primera componente es una indicación del propietario a cada miembro del equipo de que un nuevo jugador se ha unido, la segunda es el nombre del grupo, la tercera es el número de jugadores del grupo actualizado, el cuarto elemento es el nombre del jugador al que se le manda la notificación, y la quinta es el nombre del nuevo jugador añadido. Cada miembro de equipo posee una estructura interna donde guarda localmente los jugadores de su equipo con todos sus datos (identificador y apodo). Dicha estructura se va modificando por los procesos de red en base al recibimiento y envío de las tuplas anteriores.

El cierre de los equipos solamente puede ser controlado por el propietario del grupo. La clausura del grupo se produce o bien porque se agota la capacidad o porque

el propietario presiona *START*. Si ocurre el primer caso, para cada jugador el proceso del propietario postnotea la tupla `['COMPLETED_GROUP', nickNameGroup, numPlayers, nickNameplayer]` donde la primera componente indica el cierre de grupo, la segunda es el apodo del grupo a cerrar, la tercera es el número actual de jugadores del grupo y la última es el apodo del jugador destinatario de la tupla. En caso de que se cierre el grupo porque se presiona *START*, el propietario postnotea para cada jugador una tupla igual que la anterior pero cambiando `COMPLETED_GROUP` por `CLOSED_GROUP`. Con esta sutil distinción los invitados al grupo logran saber el motivo de cierre. Paralelamente, ejecuta un *Removenote* sobre la tupla `['CREATE_GROUP', nicknameGroup, numberPlayers, 'TRUE']` y la postnotea como `['CREATE_GROUP', nicknameGroup, numberPlayers, 'FALSE']` para alertar que ese grupo ya está cerrado y que no se manden más peticiones a su propietario.

Otro aspecto importante es la cancelación del grupo cuando se está formando. Esta situación ocurre cuando el propietario pulsa *ESC* sin haber cerrado el grupo. En este caso cada jugador del equipo recibe por parte del propiteario la tupla `['CANCELED_GROUP', nickNameGroup, numPlayers, nickNameplayer]`. La semántica de la tupla es igual a las anteriores con la diferencia de que la primera componente marca orden de cancelación de grupo. Además, el propietario retira el nombre del grupo cancelado haciendo *Removenote* sobre la tupla `['NAME_GROUP', nicknameGroup]` para borrarla del espacio de tuplas. La segunda componente es el nombre del grupo que creó.

J.6.2. Fase de selección de modo de juego

Esta es la fase del modo multijugador en la que el propietario escoge la modalidad de juego en la que correr, *World Tour* o *Pole Position*, mientras que los invitados del grupo permanecen en la animación de espera de selección de modo de juego multijugador.

Cuando el propietario selecciona el modo de juego presionando *START*, postnotea para cada invitado una tupla con el formato `['GAME_MODE_SELECTED', nickNameGroup, nameGuestPlayer, gameCodeSelected]`, donde la primera componente es la indicación de selección de modo de juego, la segunda componente es el alias del grupo, la tercera componente es el apodo del jugador invitado al que mandar el mensaje y la cuarta es el identificador del modo de juego.

Si mientras se está en esta fase, el propietario pulsa *ESC*, se cancela el grupo formado y el proceso del propietario, encargado de la red, postnotea para cada invitado una tupla con el formato `['CANCELLED_GROUP', nickNameGroupMultiplayer, codePlayer, nickNameMultiplayer, name]`. La primera componente indica cancelación de grupo, la segunda es el alias del grupo cancelado, la tercera es el código numérico de identificación del jugador, la cuarta su nombre, y la última es el nombre del propietario del grupo. Por su parte los procesos de red de los invitados están cada cierto tiempo ejecutando un *ReadnoteX* sobre las dos tuplas anteriores, para así, enterarse de que o bien se ha seleccionado ya modo de juego o de que el grupo se ha cancelado y deben regresar al menú de validación de nombre de grupo o al menú de modo de unión de grupo, según hubiesen escogido unirse a grupo concreto o aleatorio.

Paralelamente, se ha diseñado un sistema de control de latidos para que el propietario pueda saber si todos los jugadores del grupo están operativos o si de lo contrario hay alguno que se ha caído o desconectado. Para ello, el propietario cuenta con un registro de peticiones fallidas para cada miembro de equipo, concretamente se permiten un máximo de 40 peticiones a modo de *timeout*.

Para cada jugador el proceso de red del propietario ejecuta *ReadnoteX* sobre la tupla `['PULSE_SENDED', nickNameGroup, nickNamePlayer, codePlayerInGroup]`. La primera componente indica envío de latido (pulso) por parte de un miembro del grupo, la segunda es el apodo del grupo, la tercera es el apodo del jugador del que se comprueba el pulso y la cuarta es su identificador.

Si el espacio de tuplas retorna la tupla `['NOT_FOUND']` significa que o bien no ha mandado todavía nada o bien ya se ha caído ese jugador. En cualquier caso se incrementa el número de peticiones fallidas a ese jugador. Si no devuelve la tupla anterior, significa que ya ha mandado latido. En este caso el proceso postnotea en el espacio de tuplas la tupla `['PULSE_RECEIVED', nickNameGroup, nickNamePlayer, codePlayerInGroup]`, y acto seguido, se resetean los posibles intentos de latido fallidos a ese jugador. La tupla tiene la misma semántica que la de envío con la excepción del primer elemento que simboliza latido recibido. Así, el miembro sabe que el propietario ha recibido su señal de latido y puede a mandarle otro. Por el contrario, si se supera el límite de intentos fallidos (expira el *timeout*), el propietario asume que ese jugador se ha caído y manda a todos los procesos de red del resto de miembros la tupla `['DELETE_PLAYER', nickNameGroup, nickNamePlayer, codePlayerInGroup]` para que borren a ese jugador de sus listas locales de miembros de grupo.

Para evitar envíos masivos de latidos, cuando el jugador invitado hace *Postnote* de la tupla `['PULSE_SENDED', nickNameGroup, nickNamePlayer, codePlayerInGroup]` para enviar latido al propietario, el proceso de red de dicho jugador, acto seguido, ejecuta *Removenote* sobre la tupla de confirmación de envío de latido `['PULSE_RECEIVED', nickNameGroup, nickNamePlayer, codePlayerInGroup]`, a modo de *ACK*. De esta manera hasta que no se confirma un latido enviado no se manda el siguiente.

En esta etapa también se ha controlado la posible cancelación de grupo por parte del propietario de la misma manera a como se explicó antes. Además, si el propietario permanece en el grupo pero todos los demás abandonan también el grupo es cancelado. Si el propietario cancela el grupo, a él se le informa de la acción efectuada y al resto de jugadores, los invitados al grupo, se les notifica en pantalla que el grupo ha sido cancelado. De esta manera, todos los jugadores saben en todo momento que es lo que ocurre y en qué estado se halla el modo multijugador.

J.6.3. Fase de selección de circuito

Esta fase del modo multijugador solamente aparece si en el menú anterior el propietario selecciona el modo *Pole Position* En esta etapa, el propietario selecciona el circuito en el que se va a correr mientras que el resto de miembros esperan en la animación de espera de selección de circuito a que confirme.

Cuando el jugador selecciona el circuito en el que se va a correr y el número de vueltas que se van a dar pulsa la tecla *START*. Al presionarla el proceso de red del propietario ejecuta para cada miembro del equipo la operación *Postnote* con la tupla `['CIRCUIT_SELECTED', nickNameGroup, nickNamePlayer, codeCircuit]`. La primera componente indica que el propietario ha escogido un circuito, la segunda es el nombre del grupo, la tercera es el nombre del miembro del grupo al que se le manda el aviso y la cuarta es el código del circuito a cargar. De este modo, cada jugador carga localmente en su cliente el circuito recibido leyendo el fichero *XML* correspondiente.

Por su parte, los procesos de red de los invitados están periódicamente revisando si la tupla anterior está insertada en el espacio de tuplas. Además, es importante tener en cuenta que el sistema de latidos para comprobar que los jugadores invitados siguen operativos, así como la tarea de poder eliminarlos, por parte del propietario y de

los invitados, también sigue operativa durante esta fase. Sin embargo, es importante controlar que el propietario pueda volver atrás porque, por ejemplo, se puede haber equivocado de modo de juego o simplemente se arrepiente de su decisión. En tal caso, el proceso de red del propietario, postnotea para cada cliente una tupla con el formato `['CANCELLED_CIRCUIT', nickNameGroupMultiplayer, codePlayer, nickNameMultiplayer, name]`. La primera componente indica cancelación de circuito, la segunda es el alias del grupo cancelado, la tercera es el código numérico de identificación del jugador, la cuarta su nombre, y la última es el nombre del propietario del grupo. De este modo, los procesos de red de los invitados, que en este punto, ejecutan la operación *ReadnoteX* sobre esta tupla son informados de que la selección de circuito es inválida y deben regresar a la animación de espera de selección de modo de juego. De igual modo que con la cancelación del grupo, si el propietario cancela la selección de circuito, se le visualiza la acción en pantalla, y al mismo tiempo, a los invitados también se les notifica por pantalla que el circuito ha sido cancelado. Así, los jugadores conocen el estado del grupo en todo momento.

J.6.4. Fase de selección de vehículos de los jugadores

Esta etapa es obligatoria en el modo multijugador. Es la etapa más importante de todo el proceso de formación de partida ya que aquí todos los jugadores (propietario e invitados) deben elegir con que vehículo correr. La sincronización y comunicación se ha llevado de la siguiente forma.

Esta fase, por motivos de simplicidad, se ha divido en dos subfases. La primera es aquella en la que los jugadores seleccionan su vehículo, y la segunda, es aquella en la que permanecen a la espera de que el resto de jugadores seleccionen el suyo. Diseñando una barrera, los jugadores quedan bloqueados hasta que todos han seleccionado.

Mientras cada jugador elige el vehículo, el proceso de red de cada jugador está monitorizando si alguno de los jugadores, incluido el propietario, cancela la selección de vehículo. Dichos procesos de los invitados se hayan bloqueados ejecutando *Removenote* sobre una tupla que tiene el formato `[order, nickNameGroupMultiplayer, codePlayer, namePlayerSender, nickNameMultiplayer]` donde la primera componente es el tipo de orden a realizar, la segunda es el nombre del grupo, la tercera es el código identificador del jugador invitado destinatario y la cuarta es el nombre del invitado. La primera componente puede tomar cualquiera de los siguientes valores: `'PLAYER_CLOSED', 'CANCEL_CAR', 'VEHICLE_DONE', 'DELETE_PLAYER`.' La primera

opción indica que un invitado deja el grupo. Si el proceso de red de ese jugador invitado la recibe, vuelve al menú de validación de nombre de grupo o de modo de unión a grupo según corresponda. Si el proceso recibe la segunda orden, significa que el propietario del grupo ha cancelado la selección de vehículo y deben regresar o a la animación de espera de selección de modo de circuito o a la animación de espera de selección de circuito, según el propietario seleccionase como modo de juego *World Tour* o *Pole Position*. Si se recibe la tercera orden significa que se ha seleccionado vehículo con el que correr y si llega la última significa que ha habido un jugador invitado que ha saludo del grupo y lo tiene que eliminar.

En cuanto al propietario, su proceso de red ejecuta *Removenote* sobre una tupla con el mismo formato pero con órdenes distintas. Dichas órdenes pueden ser `'CANCEL_CAR'`, `'VEHICLE_DONE'` y `'LEAVE_GROUP'`. La primera indica que la selección del vehículo se cancela y se debe decir a todos los invitados que retrocedan a la fase anterior. La segunda muestra que el jugador ha seleccionado vehículo con el que correr y la tercera que uno de los jugadores invitados ha decidido abandonar el grupo. En tal caso, el propietario elimina de su lista al jugador que ha abandonado y comunica al resto de jugadores una tupla con el formato `['DELETE_PLAYER', nickNameGroupMultiplayer, namePlayer, codePlayer]`. La primera componente es la orden de borrar a un jugador del grupo, la segunda es el nombre del grupo, la tercera es el apodo del jugador a borrar y la cuarta es el identificador del jugador a borrar.

Tras seleccionar el vehículo, se informa a todos los jugadores invitados de cuál es el vehículo seleccionado, notificando el tipo de bólido y su color. Para ello, el proceso de red de dicho jugador hace *Postnote* con la tupla `['VEHICLE_SELECTED', nickNameGroup, codePlayer, namePlayer, typeOfVehicle, color]`. La primera componente indica al resto de integrantes que el jugador ha seleccionado ya vehículo, la segunda es el nombre del grupo, la tercera es el identificador del jugador que ha seleccionado coche, la cuarta es el apodo del jugador, la quinta es el tipo de vehículo y la última es el color del vehículo. Todos aquellos jugadores que han elegido coche, después de hacerlo permanecen esperando a que el resto de jugadores también lo hagan. De este modo, los procesos de red de dichos miembros ejecutan periodicamente *ReadnoteX* sobre la tupla anterior. Si devuelve la tupla `['NOT_FOUND']` significa que ningún jugador ha seleccionado coche. En caso contrario, se guarda localmente el coche y el color seleccionados. El orden cronológico de incorporación de los jugadores al grupo determina su posición en la parrilla de salida para la primera carrera.

Cabe destacar que una vez seleccionado un vehículo, los jugadores no pueden abandonar hasta que todos los demás hayan registrado su coche. Solamente pueden abandonar si no han seleccionado vehículo todavía. Además, para tener información de cuántos jugadores quedan por escoger y así empezar la carrera, se cuenta con un registro interno que contabiliza que jugadores ya han seleccionado con respecto del total y cuáles no (véase figura J.10). Dicha información se muestra en la pantalla al jugador por medio de dos contadores, los cuáles, pueden cambiar también si algún jugador que todavía no ha escogido bólido abandona el grupo.

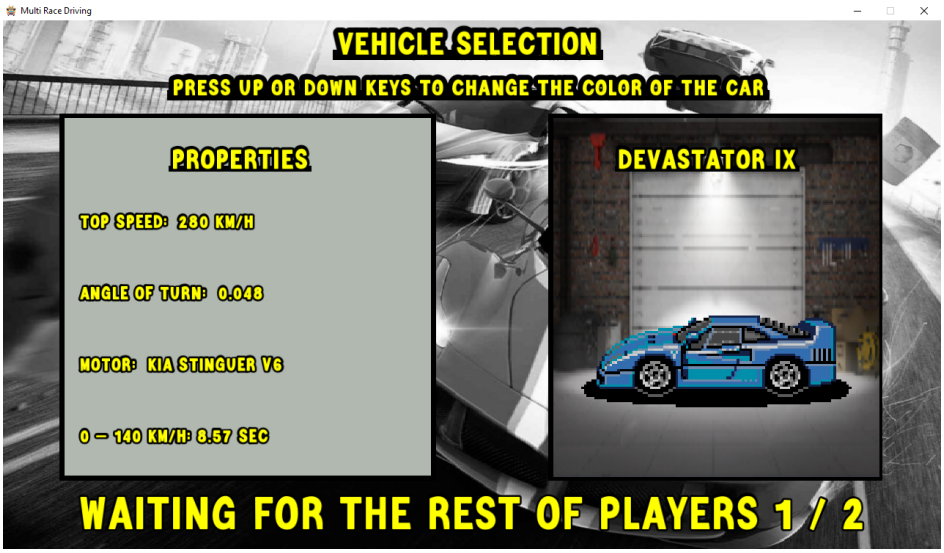


Figura J.10: Jugador en espera tras haber seleccionado vehículo

J.6.5. Fase de comienzo de carrera

En esta etapa se posicionan los vehículos de los jugadores en la parrilla de salida. El orden de posicionamiento es establecido por el orden de incorporación al grupo en caso de que se vaya a correr la primera carrera. Si no es la primera, el posicionamiento depende del orden de llegada de los jugadores en la carrera anterior El proceso de mapeo de posiciones de los coches ha sido un tema muy complejo ya que se debe coordinar la información a ver por parte de los jugadores. A continuación, se muestra un ejemplo de visualización de parrilla de salida para un grupo completo de cuatro jugadores con capturas de pantalla tomadas desde el ordenador de cada jugador (véase figura J.11).

(a) Primer jugador

(b) Segundo jugador

(c) Tercer jugador

(d) Cuarto jugador

Figura J.11: Perspectivas de parrilla de salida

J.6.6. Fase de transcurso de carrera

Esta fase comprende todo el núcleo de acción desde que los coches arrancan hasta que llegan a la meta. Son muchos los aspectos de sincronización que han tenido que tenerse en cuenta para poder brindar a los jugadores una experiencia de juego adecuada y correcta.

Durante la carrera, el modo multijugador ofrece prácticamente la misma funcionalidad que los modos de juego para un solo jugador. Sin embargo, hay algunos aspectos que no se han contemplado dado que no tenían mucho sentido lógico. En primer lugar, para poder salir de la partida los jugadores deben pulsar la X de la ventana del programa, y en segundo lugar, a diferencia de como ocurre en los modos de juego para un solo jugador, el juego no puede ponerse en pausa. De este modo, la carrera comienza y no sufre ninguna parada hasta que se acaba cuando todos los participantes llegan a la meta. La razón por la que se ha suprimido dicho menú es que el modo multijugador necesita ser jugado en tiempo real. Si un jugador pausase el juego fastidiaría a todos

los demás que están jugando con él, sobre todo, si hubiesen muchos jugadores en partida. Por consiguiente, para evitar este posible problema se ha decidido eliminarlo. Además, muchos juegos populares como *Mario Kart Tour*, para *Android* e *iOS* (2019 *Nintendo*)[8], operan de manera similar.

Un aspecto de suma importancia durante las partidas es la gestión de la posible desconexión o caída de los jugadores. Para poder controlar este matiz, el proceso de red de los jugadores está continuamente monitorizando si alguno de los jugadores del grupo, incluido el propietario, se desconecta o se cae. Por simplicidad, cuando un jugador abandona una partida, independientemente del motivo, no puede volverse a conectar a ese grupo, porque ya está cerrado. Si abandona la partida o se desconecta accidentalmente, si desea volver a jugar debe volver a solicitar unirse a otro grupo.

A nivel de sincronización, cuando un jugador abandona la partida, su proceso de red postnotea una tupla con la información `['PLAYER_ABORT', nickNameGroupMultiplayer, odePlayerInGroup , namePlayer]` donde la primera componente significa que un jugador invitado ha abandonado la carrera, la segunda es el nombre del grupo, la tercera es el código de identificación del jugador que ha abandonado y la última es el nombre del jugador. El proceso de red del propietario, por su parte, permanece bloqueado revisando si alguna tupla de este tipo le ha sido enviada. En tal caso, al resto de jugadores (invitados) les manda la tupla `['REMOVE_PLAYER', nickNameGroupMultiplayer, idPlayer, namePlayer]`, donde la primera componente les indica a los jugadores que un jugador ha abandonado el grupo y deben borrarlo de sus listas de jugadores de grupo locales de la misma manera que lo hace el propietario. La segunda es el nombre del grupo y las dos últimas determinan el código de identificación y el nombre del jugador a borrar. Cuando todo esto ocurre se muestra durante tres segundos el nombre del jugador desconectado (véase figura J.12) si aún quedan más de un jugador jugando (el propietario y al menos un jugador invitado).

Si durante la carrera todos los jugadores a excepción del propietario abandonan el grupo, es decir, que el propietario se queda solo, también se cancela el grupo puesto que no hay jugadores con los que seguir compitiendo, y además, estos no pueden volver a conectarse (véase J.13).

[8] `https://mariokarttour.com/es-ES`

Figura J.12: Aviso de jugador desconectado durante partida

Figura J.13: Grupo cancelado al quedarse solo el propietario

El propietario es aquel que tiene el control de la partida. De esta manera, cuando cancela la partida en curso, su proceso de red notifica a todos los demás la tupla `['RACE_ABORT', nickNameGroupMultiplayer, codePlayerGroup, namePlayer]`. El primer elemento indica la orden para cancelar la carrera por parte del propietario,

la segunda es el apodo del grupo, la tercera es el código de identificación del jugador invitado al que se le manda la notificación y la cuarta es su alias.

Aparte de tener que sincronizar los posibles abandonos que los jugadores pueden experimentar durante una partida multijugador, también se ha tenido que diseñar un mecanismo de coordinación para que los jugadores sepan rápido y en todo momento, no solo donde se localizan todos los jugadores en el escenario, sino también en que estado se encuentran (parados, corriendo, girando, frenando, chocando, etcétera).

La gestión del envío y recibo de las posiciones de los jugadores y de su estado en carrera es un aspecto cuyo tratamiento se ha separado de la gestión de fallos, entre otros motivos, para no sobrecargar con excesivo trabajo a los hilos encargados de esta tarea. De esta manera, aunque hay más hilos en marcha, cada uno tiene una tarea sencilla y concreta que hace más fácil para el programador el poder implementarla. Así las cosas, para llevar a cabo esta tarea se han empleado dos hilos, uno para poder mandar las posiciones del jugador y otro para poder recibir las posiciones de todos los demás miembros del grupo.

En definitiva, cuando un jugador juega al modo multijugador, durante una partida, el cliente tendrá los siguientes procesos ejecutándose:

- Hilo encargado de controlar el abandono o cancelación de la partida en curso. Así como de notificar posibles jugadores caídos.
- Hilo responsable de controlar el tiempo transcurrido de partida (panel de la esquina superior izquierda de la pantalla).
- Hilo para enviar al resto de jugadores del equipo la posición y estado en el que se encuentra el jugador en carrera.
- Hilo para recibir la posición y estado del resto de participantes que compiten con el jugador en la carrera.
- Hilo de ejecución principal que se comunica con todos los procesos anteriores y muestra en pantalla al jugador el estado del juego.

Mientras el jugador está corriendo, el hilo encargado de manejar el envío de las posiciones del jugador está revisando de manera continua las posiciones y el estado del jugador. Dicha lectura se efectúa en exclusión mutua con respecto al hilo de ejecución

principal, que entre otras muchas tareas se encarga de controlar la pulsación de las teclas por parte del jugador que modifican dichas variables (teclas de acelerar, frenar y giro) con el fin de evitar posibles fallos de sincronización. Una vez leída la posición y el estado del jugador, haciendo uso de la operación *Postnote* se envía a cada jugador del grupo una tupla con el formato `['PLAYER_POSITION_' + nickNameMultiplayer, namePlayer, positionX, positionY, codeImage, codeLandScape]`. La primera componente indica el envío de una posición de un jugador. Es una cadena de caracteres a la que se le concatena el nombre del jugador que la manda a fin de cuentas de que el destinatario sepa a que miembro del grupo pertenece esa posición. La segunda componente de la tupla hace referencia al nombre del jugador al que va dirigida esa posición, la tercera y la cuarta son las posiciones donde cada miembro del grupo debe dibujar al jugador remitente en pantalla siempre y cuando sea visible, la quinta hace referencia al estado del jugador, que ha sido representado como un número entero que codifica el la textura del jugador a dibujar en pantalla. La última componente hace referencia al identificador del escenario en el que el jugador se encuentra.

Este identificador es un índice correlativo que se incrementa cuando el jugador completa un escenario y pasa al siguiente. Este identificador es sumamente útil ya que evita por completo que jugadores que, siendo visibles desde otros desde el punto de vista del renderizado de la cámara, sean dibujados porque se encuentran en otro escenario distinto. Cada mapa tiene un sistema de referencia situado en el origen de ese escenario. De esta manera, todos los objetos presentes en el escenario (vehículos y obstáculos) se posicionan tomando como referencia dicho sistema. Dos vehículos pueden tener las coordenadas idénticas pero localizarse en mapas distintos, por ejemplo en el modo *World Tour*, un vehículo está en el mapa de salida en la posición (-0.3, 250.2) y otro vehículo en la posición (0.5, 210.4) del mapa contiguo, es decir, del escenario leído por el fichero *XML*. Dicho escenario se concatena al primero. El primer jugador es visible por el segundo porque sus coordenadas están dentro de la superficie de renderizado de la cámara. Sin embargo, no puede dibujarse porque está en el mapa anterior, es dicho identificador el elemento que evita que dicho jugador sea dibujado.

De manera paralela, está en funcionamiento el proceso de recibo de posiciones, que está continuamente consultado mediante la operación *ReadnoteX* la última posición enviada por los jugadores. Para consultar las posiciones se ha utilizado esta operación en vez de las operaciones de consulta *Readnote* o *Removenote* debido a que dicha operación no es, a diferencia de las otras, bloqueante si la tupla no está. Si hay una nueva tupla, se ejecuta sobre ella la operación *Removenote* para eliminarla. De este

modo, aunque se ralentiza el proceso de consulta, se mantiene el espacio de tuplas solamente con la información relevante ya que toda la demás se elimina conforme se procesa. Así, se reduce el espacio de búsqueda, y por consiguiente el tiempo de ejecución de las operaciones.

J.6.7. Terminación de carrera

Esta parte se corresponde con la última etapa de ejecución de la carrera. Cuando el jugador llega a la meta, se le muestra la misma animación de llegada, y acto seguido, se espera a que todos los demás competidores lleguen también a la meta.

El mecanismo usado para diseñar esta fase de sincronización es muy parecido al utilizado en la selección de vehículos. Se ha desarrollado una barrera, en la que tras llegar a la meta, los jugadores permanecen bloqueados esperando a que todos los jugadores que compiten lleguen.

Cuando todos han conseguido llegar se baja la barrera y los jugadores pasan al siguiente circuito si es el caso del *World Tour* y quedan más niveles por recorrer. Si es el caso del *Pole Position* los jugadores invitados regresan a la animación de espera de selección de circuito y el propietario al menú de selección de circuito.

Cuando un jugador llega a la meta, su proceso de red, haciendo uso de la operación *Postnote* inserta en el espacio de tuplas la tupla `['ARRIVED_GOAL, nickNameGroup, codePlayer, namePlayer, posArrival]`. Esta tupla permite comunicar al resto de jugadores de que ya ha llegado a la meta y de que va a proseguir a esperar en la barrera hasta que todos los demás lleguen.

La primera componente de la tupla informa de la llegada del jugador a la meta, la segunda es el nombre del grupo al que pertenece el jugador, la tercera y la cuarta sirven para identificar al jugador en cuestión (identificador y nombre de jugador), y por último, la quinta marca la posición en la que ha llegado a la meta.

Tras mandar esta tupla, permanece esperando en la barrera hasta que todos los jugadores llegan. Regularmente, lo consulta haciendo uso de la operación *ReadnoteX* sobre una tupla con el mismo formato que la anterior, pero sustituyendo el identificador y nombre del jugador propios por los del resto de jugadores. Para poder controlar los jugadores ya chequeados se dispone de un registro que almacena los jugadores de

la partida que ya han llegado y en el orden en el que lo han hecho. De este modo, para construir las parrillas de salida de los carreras futuras se usa esta estructura a diferencia de si es la primera carrera, que se usa como criterio de posicionamiento el orden cronológico de incorporación a grupo.

Durante la espera de la barrera, al igual que en la selección del vehículo, el jugador puede ver en todo momento en qué estado se encuentra ya que se le muestra por pantalla cuántos jugadores quedan por llegar con respecto del total de jugadores activos, y además, puede ver el orden en que van llegando dichos jugadores (véanse figuras J.14 y J.15).

Por motivos de simplicidad, tras llegar a la meta el jugador no puede salirse del grupo ni cancelarlo ya que sino se dificultaría enormemente todo el proceso de sincronización. No obstante, si que es capaz de recibir las peticiones de abandono del resto de jugadores, siempre y cuando todavía estén en carrera. De este modo, se evita esperar en la barrera a aquellos jugadores que han abandonado. Si esto no se hiciese así, se produciría un bloqueo total de la aplicación.

Figura J.14: Propietario esperando a que termine la carrera el invitado

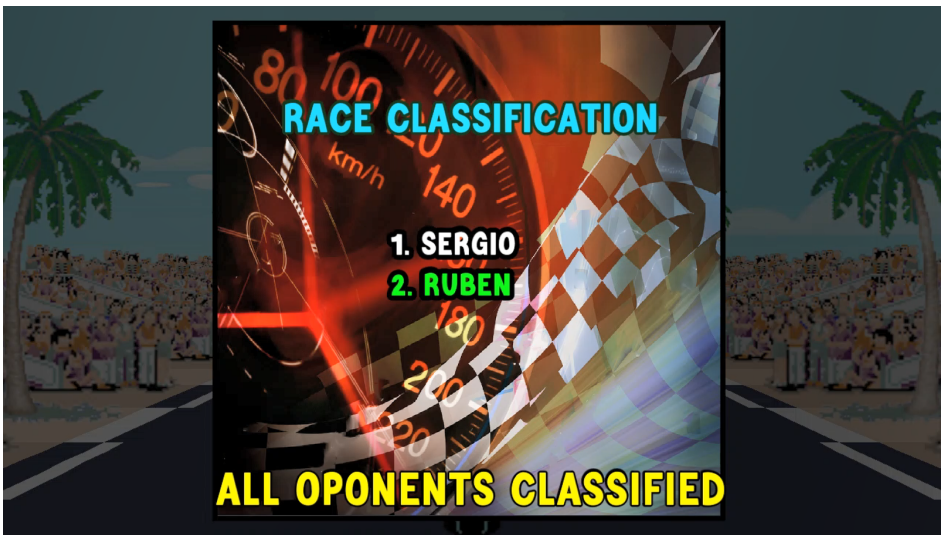


Figura J.15: Invitado tras acabar la carrera habiendo llegando el propietario

Esta interfaz es visualizada localmente por cada jugador de manera que a la hora de mostrar la clasificación de los jugadores, todos los rivales aparecen en blanco y el propio jugador en verde. De esta manera, el jugador localiza más rápido su posición en el *ranking*.

Anexo K

Valoraciones de *Multi Race Driving*

En este anexo se procede a comentar las valoraciones que los distintos *game testers* realizaron sobre la versión *beta*. Para garantizar el anonimato, las valoraciones han sido citadas sin especificar los nombres de las personas en cuestión. Para cada una de estas valoraciones se han especificado la edad de la persona, la profesión que ocupa actualmente, y los aspectos positivos y negativos de la arquitectura. A continuación, se exponen las valoraciones.

K.1. Primera valoración

El primer *game tester* tiene 21 años y estudia Ingeniería Informática. De acuerdo a su valoración, el programa presenta los siguientes puntos positivos:

- Buena recreación de los distintos modos originales.
- Gráficos muy bien conseguidos, muy visuales, y muy estéticos.
- Banda sonora variada y muy pegadiza al igual que el *SFX* utilizado.
- Colisiones bastantes realistas.
- Mucha configurabilidad y libertad para que el jugador pueda configurar el *software* a su gusto.
- Fácil manejo de los vehículos.
- Elementos decorativos de mapas muy visuales y distintos.
- Sensación de velocidad bastante lograda.
- *IA* bastante variada y notable diferencia de jugar de unas dificultades a otras.

– El modo multijugador es bastante completo y divertido de jugar.

Los aspectos negativos son los siguientes:

– En las colisiones, los vehículos alguna vez atraviesan los obstáculos.

– En modo de máxima dificultad, algunas veces es demasiado costoso poder ganar porque muchos rivales te obstaculizan.

– Algunos *sprites* no son acordes con la temática del escenario ni con el modo de juego.

– El modo multijugador a veces sufre *lag*.

– En las bifurcaciones, el cambio de color es abrupto y los coches no deberían desaparecer.

– Alguna que otra vez se bloquea la terminal durante un instante en la pantalla de carga.

– En el modo *World Tour* y *Pole Position* al empezar la carrera hay un pequeño *glitch*[1].

K.2. Segunda valoración

El segundo *game tester* tiene 22 años y es estudiante de Ingeniería Informática. Los puntos positivos son los siguientes:

– Personalización de la aplicación muy completa porque le proporciona al jugador el control en casi todos los aspectos.

– Interfaz y *GUI* muy cuidadas. Buena gama de color, fuentes legibles, concisas y con mensajes significativos.

– Navegabilidad muy dinámica.

– Buena respuesta de las teclas.

– Modos de juego muy variados y bastante conseguidos.

[1] `https://www.geekno.com/glosario/glitch`

- Gráficos muy estéticos y visuales.
- La sensaciones de velocidad y profundidad son muy realistas.
- Interfaz de juego simple y clara.
- Paleta de colores muy variada.
- Muy buen aporte poder jugar con más vehículos que en los juegos originales.
- *IA* variada y dinámica. Muy notable el cambio de unos modos de dificultad a otros.
- Animaciones muy cuidadas.
- Música y *SFX* muy acordes a la temática de la aplicación.
- Multijugador bastante bien logrado.

Los aspectos a mejorar por este segundo *game tester* son los siguientes:

- Algunos *sprites* son muy poco acordes con el modo de juego (palmeras en un circuito de *F1*).
- El multijugador sufre un *lag* a la hora de actualizar los coches contrarios (dan saltos).
- El programa registra el pulso de teclas aún estando en segundo plano. solo debería capturarlas si está seleccionado ya que dificulta otras tareas.
- Los *sprites* de los vehículos presentan una calidad algo más reducida que los de los escenarios. Altamente notable si se juega sin efecto *pixel art* y se conduce el coche de *F1*.
- El modo *World Tour* se hace corto, debería haber más niveles.
- Fuente de las interfaces de juego poco pegadizas. Muy curvadas, deberían ser más pixeladas para dar efecto *retro*.
- Menús con calidad alta e interfaz de juego *retro*. Hay cierta pequeña incompatibilidad.
- La bifurcación es algo brusca y los coches desaparecen.

K.3. Tercera valoración

El tercer *game tester* tiene 60 años, ocupa como profesión el cargo de funcionario. Los rasgos atractivos para él son los siguientes:

- Notable diferencia a la hora de correr con los distintos vehículos.
- Dinámica del juego excelente.
- Muy buena jugabilidad y control del vehículo.
- Necesario frenar para poder tomar las curvas porque sino te chocas.
- Música y sonidos muy pegadizos y adictivos.
- Gráficos muy conseguidos para ser un juego de temática *retro*.
- Muy útil conocer el *test* de conexión previo para comprobar el estado del cliente.
- Fácil de modificar los menús, vehículos y escenarios.
- Gratificante poder jugar en múltiples resoluciones, incluida pantalla completa.
- Multijugador bastante logrado.
- Material con *copyright* bien reflejado.
- Muy buen control de los vehículos del jugador.

Las críticas son las siguientes:

- En los escenarios en los que aparecen señales de tráfico, a veces aparecen muy seguidas.
- Hay un pequeño parpadeo al arrancar en algunos modos de juego, no en todos.
- Hubiese sido mejor diseñar los *sprites* a mano y no emplear música con *copyright*, sino música *free*.
- Las líneas de fondo de los paisajes hacen daño a la vista, sobre todo, si vas a bastante velocidad y juegas con alta resolución en la *TV*.
- Detecta las teclas cuando el programa está en segundo plano.
- El vehículo de policía al igual que el resto de coches debería tener más colores.

– El multijugador se ralentiza durante la partida.

– Podría ser buena la opción de poder jugar con mando.

K.4. Cuarta valoración

El cuarto *game tester* tiene 16 años y estudia educación secundaria obligatoria. Los aspectos buenos son los citados a continuación:

– Vehículos y colores muy variados.

– Paisajes y gráficos muy visuales.

– Gestión del audio muy lograda.

– Buen control de los coches, aceleran y frenan respondiendo perfectamente al pulso de teclas.

– Modalidades de juego muy variadas y muy bien explicadas. Son buenas recreaciones personalizadas en comparación a los juegos originales.

– Sensación de velocidad, perspectiva y físicas muy conseguidas.

– Muy buena idea incorporar el multijugador.

– Las interfaces de juego son sencillas, claras, y no entorpecen la visibilidad del juego.

– La *IA* es variada y muy dinámica.

– Los aspectos de configuración según el modo de dificultad están muy cuidados.

– Menús muy estéticos.

– Mucha configurabilidad y muy intuitiva.

– Animaciones muy visuales.

Los aspectos negativos son los siguientes:

– Al igual que el *Out Run*, debería haber contador de puntos y *ranking* en las clasificaciones para dar mayor rivalidad en el resto de modos.

– La pantalla de carga dura mucho tiempo.

– En la salida de carrera, si se empieza muy lejos y se juega con *pixel art* no se aprecian ni al banderillero ni al semáforo cambiar. Se debería poner otro en la parte superior central para que se viese mejor.

– El *pop-up* del fondo de los paisajes debería eliminarse.

– Algunas veces, cuando vas a mucha velocidad y tomas una bajada parece como que el vehículo flota porque el escenario baja más de la cuenta.

– El modo multijugador debería permitir más jugadores por grupo y que para los apodos se pudiesen usar caracteres especiales.

– Las distancias en la interfaz son irregulares, los vehículos miden su velocidad en km/h. Las distancias a los puntos de control y a las metas se miden en km, y descienden muy rápido. Es físicamente imposible. Deberían ser m.

– En modo difícil es muy complicado adelantar a los rivales, sobre todo, en *Pole Position* si usas vehículos pesados como camiones o todo terrenos.

– En los choques, los vehículos pueden atravesar los objetos.

K.5. Quinta valoración

Este último *game tester* tiene 30 años, es ingeniero mecánico. Las características positivas son:

– Muy buena jugabilidad.

– Gráficos muy variados y muy visuales.

– Modos de juego muy diversos y bien logrados.

– Configurabilidad muy sencilla.

– Interfaz y animaciones muy adecuadas.

– Sonidos y bandas musicales de alta calidad.

– Detección de teclas muy bueno.

– Sensación de velocidad y perspectiva muy conseguidas.

– Gran variedad de vehículos entre los que elegir.

- Bastante bien equilibrados los modos de dificultad.
- Interfaces de juego muy bien cuidadas y adaptadas de manera que no afecta a la visibilidad de la acción en partida.
- Configuración de controles y volúmenes de sonido correcta.
- Navegabilidad muy dinámica.
- Sección de créditos muy trabajada.

Los aspectos negativos son los siguientes:

- La interfaz de los menús debería ser un poco más *retro*.
- Los vehículos controlados por la *IA* giran en dirección contraria a veces en las curvas.
- Detecta la pulsación de teclas en segundo plano.
- No hay mucho margen de tiempo de juego si te chocas, sobre todo, en el modo difícil.
- Existe cierta descompensación entre los vehículos. Los camiones deberían por ejemplo poder apartar de un golpe al resto de coches.
- Los vehículos desaparecen en las bifurcaciones.

Anexo L

Posibles ampliaciones

En este anexo se explican cuáles son las posibles alternativas que se habían pensado para extender la arquitectura de *Multi Race Driving*, y además, también se detallan cuáles han sido los principales problemas que se han tenido que abordar durante la realización del proyecto.

L.1. Extensiones de *Multi Race Driving*

En esta sección se van a describir todas aquellas extensiones o mejoras que inicialmente se habían pensado introducir a *Multi Race Driving* pero que finalmente no se han incorporado.

L.1.1. Modo de juego *Delivery*

Este modo iba a estar disponible al igual que *World Tour* y *Pole Position*, tanto para un solo jugador como para multijugador. La temática de este juego iba a consistir en hacer misiones como entregas de paquetes, llevar personas a ciertos sitios, etcétera, yendo de un punto del mapa a otro antes de que alguno de los otros rivales (personas físicas o rivales controlados por *IA*) cumpliesen con el objetivo.

Era un modo de juego inspirado en otro videojuego muy popular de los 90, *Crazy Taxi*[1] (véase figura L.1). La razón por la que no se ha metido es que los escenarios de los otros modos de juego no eran acordes a esta temática. Dichos paisajes son dirigidos, es decir, solo hay una única ruta que seguir, la que viene marcada por la carretera.

[1] `https://as.com/meristation/2020/02/01/reportajes/1580543004_618146.html`

Figura L.1: *Crazy Taxi* lanzada por *Sega* en 1999

Este modo de juego necesitaba de la creación de o bien mundos en 3D, lo que rompía por completo con el resto de modos de juego, cuyos gráficos son 2.5D, o crear planos de ciudades en 2D vistos con perspectiva aérea (véase figura L.2), técnica muy empleada en videojuegos antiguos.

Figura L.2: Perspectiva aérea de *Grand Theft Auto 2* lanzado en 1999

En cualquiera de los casos, es más que obvia esta incompatibilidad, y aún mucho más influyente es que la técnica para generar los escenarios con ficheros *XML* tampoco era compatible. Hubiese sido necesario crear nuevos escenarios y desarrollar una técnica nueva, sencilla e intuitiva que permitiese poder generarlos.

L.1.2. Modo multijugador global

Otra posible ampliación era poder extender el modo multijugador al resto de modos de juego, es decir, a *Out Run*, a *Driving Fury* y a *Demarrage*. Sin embargo, al final no se decidió hacerlo, no por tema de complejidad ya que el modo multijugador es un módulo completamente aparte, sino que tras un análisis concienzudo se llegó a la conclusión de que la temática de estos modos de juego no concordaba con la idea de hacerlo multijugador porque la gracia del juego se perdía.

L.2. Cuestiones de viabilidad

En este apartado se detallan todos los aspectos de la arquitectura que no han podido mejorarse tanto por las limitaciones que tiene el diseño de *SFML*, como por la falta de conocimiento y tiempo para poder diseñarlas.

L.2.1. Colisiones

A pesar de ser bastante realistas, tienen el problema de que en algunas ocasiones el vehículo del jugador puede atravesar los objetos (véase figura L.3). Este problema se debe a que la manera con la que *SFML* dibuja los distintos objetos en el mapa está basada en el principio de superposición.

Para poder visualizar el mapa, primero se dibuja el fondo. Seguidamente, el terreno del paisaje, luego la carretera y los carriles. Finalmente, los elementos del mapa, el jugador y la *HUD*. Si se dibujase primero la carretera y luego el terreno, la calzada sería machacada por el terreno. Si los carriles de la calzada se dibujasen antes que la propia calzada, serían superpuestos obteniendo como resultado carretera lisa sin carriles.

En las colisiones es lo mismo. Si el vehículo fuese dibujado antes que los objetos del mapa, al acercase el vehículo sería cortado puesto que se superpondría el obstáculo. Con el enfoque actual ocurre que al chocar con un objeto del mapa, el vehículo puede superponerse sobre él. Para evitarlo, se han configurado los escenarios para que los elementos estén muy separados. No obstante, podría volver a ocurrir si se hacen cambios inadecuados en la frecuencia de aparición de los objetos.

Figura L.3: Moto atravesando cartel

L.2.2. Bifurcaciones

SFML, es una librería que permite cargar imágenes en distintos formatos. Sin embargo, no se pueden cargar con porcentaje de opacidad. La idea que se tenía en mente, era cargar la imagen de fondo del nuevo escenario al que se había decidido acceder mientras se cruza la bifurcación, y superpuesto delante, el fondo del mapa actual con un grado de opacidad variante entre el 0 % y el 100 %. De este modo, al principio de la bifurcación el fondo sería el mismo que el del escenario que se acaba de concluir, y conforme se va recorriendo la bifurcación por el carril escogido, ese fondo iría dibujándose con una mayor transparencia dejándose ver el fondo del nuevo escenario, hasta que desaparecería por completo al llegar al final de la bifurcación. Sin embargo, no pudo hacerse porque las imágenes no pueden ser cargadas con opacidad. De ahí, que el cambio haya quedado tan abrupto.

Los colores, por su parte, si que se pueden cargar con un cierto grado de opacidad, un claro ejemplo se puede observar en el menú de pausa. A pesar de ello, no se ha podido lograr un cambio de color mucho más progresivo, que fuese cambiando lentamente el color del terreno del mapa antes de la bifurcación al color del terreno del mapa resultante tras salir debido a que no se conocía ningún algoritmo o técnica que permitiese hacerlo. La diferencia de este matiz con respecto al anterior es que éste si que para versiones futuras puede ser susceptible a mejora.

L.2.3. *Glitch* de arranque

Otro matiz es que al arrancar el programa, se produce una especie de *glitch* en el que por un pequeño instante de tiempo aparece el *shell* de *Windows*. El *software* ha sido compilado como una aplicación de usuario (*GUI application*), es decir, no es una aplicación de consola (*console application*). Sin embargo, para poder realizar el *test* de conexión al servidor, es necesario invocar al *shell* para que internamente ejecute el comando *ping* y evalúe la calidad de conexión con el servidor remoto. Para poder hacer eso, es necesario asignar una consola al proceso porque de lo contrario no puede ejecutarse dicho *test*. La razón estriba en que las aplicaciones de usuario no tienen consola por defecto. La asignación de una consola *shell* para poder ejecutar dicho *test* por defecto siempre es visible, y posteriormente, se configura para que no lo sea, es decir, no se puede configurar el lanzamiento de la consola para que siempre esté oculta, sino que por un momento se ve, y luego desaparece ocasionando un pequeño *glitch*.

L.2.4. Modo multijugador

El modo multijugador se ha diseñado con el fin de poder tratar la máxima cantidad de fallos posibles que puedan ocurrir. Sin embargo, al ser el primer multijugador diseñado, algún aspecto es susceptible a mejora. Por ejemplo, que el transcurso del modo multijugador funcione correctamente depende enormemente del propietario. De tal manera que si éste experimenta alguna anomalía repentina como un bloqueo o que simplemente el jugador mate al programa desde el panel de administración de tareas del computador, el grupo se bloquearía y colapsaría debido a que no hay propietario que lo controle. Dicho aspecto podría resolverse implementando un algoritmo que fuese capaz de detectar la caída del propietario, y acto seguido, elegir uno nuevo de entre los jugadores invitados para tomar el mando del grupo. Aparte, habría que diseñar un sistema que permitiese capturar y tratar las señales que manda el *SO*. La dificultad en este sentido estriba en que el tratamiento de señales típico de la arquitectura de *Windows* es mucho más enrevesado que por ejemplo el que tiene la arquitectura de *Unix*. Además, el soporte de *C++* para el tratamiento de señales con *Windows* es bastante escaso, cosa que no ocurre con el de *Unix*. Con dedicación e investigación, este matiz podría resolverse, por lo que se tiene en cuenta como característica de diseño a mejorar en caso de continuar con el proyecto.