Universidad Zaragoza
1542

Trabajo Fin de Grado

Creación de un videojuego de carreras en 3D

Autor/es

Rafael Marcén Altarriba

Director/es

Eduardo Mena Nieto

Escuela de Ingeniería y Arquitectura

2017

Repositorio de la Universidad de Zaragoza – Zaguan http://zaguan.unizar.es

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Rafael Marcén Altarriba,

con nº de DNI 73029102P en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Creación de un videojuego de carreras en 3D,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 03/02/2017

Fdo: _____

Resumen

En la actualidad, el desarrollo de videojuegos ha crecido enormemente gracias a los nuevos motores gráficos y su modelo de negocio donde el desarrollo independiente no necesita realizar un gran desembolso para adquirir una licencia que permita usar el motor gráfico.

En este trabajo de fin de grado se explora la idea de crear un videojuego de carreras futurista denominado "Syncopia" partiendo de cero y utilizando como plataforma de desarrollo principal uno de los motores gráficos más populares entre los desarrolladores independientes como es Unity. Unity proporciona una base muy sólida para trabajar con entornos en 3D permitiendo un desarrollo orientado a múltiples plataformas, pero sin automatizar la creación y desarrollo del conjunto de sistemas que componen un videojuego.

Para lograr esto, se han abordado diversos problemas relacionados con el mundo de los videojuegos en el campo técnico. En concreto, se ha priorizado el desarrollo de los siguientes aspectos:

- Diseño e implementación de una inteligencia artificial adaptativa capaz de reaccionar y adaptarse a los cambios que se producen en el entorno.
- Aplicación de técnicas avanzadas de procesamiento de imágenes (*antialiasing*, oclusión ambiental, etc.).
- Diseño e implementación de la infraestructura de red necesaria para añadir un modo multijugador en red.

Además, como el desarrollo de un videojuego no solo depende de los aspectos técnicos, sino que el aspecto artístico tiene un peso muy importante, se han tenido en cuenta otros factores como la creación y búsqueda de modelos 3D, texturas, música o el diseño e implementación de las interfaces.

Índice

1 Introducción 1

1.1 Objetivo y alcance 1

1.2 Metodología y alcance 2

1.3 Motivación 2

2 Análisis 4

2.1 Requisitos 4

2.2 Casos de uso 6

3 Diseño 9

3.1 Arquitectura del sistema 9

3.2 Física de las naves 13

3.3 Arquitectura de la inteligencia artificial 14

4 Desarrollo 16

4.1 Física de las naves 16

4.2 Inteligencia artificial 18

4.3 Multijugador en red 24

4.4 Efectos gráficos 27

5 Resultados 31

5.1 Versión final del juego 31

5.2 Evaluación de los usuarios 35

6 Conclusiones 39

6.1 Posibles mejoras 40

6.2 Conclusión personal 41

7 Referencias 42

8 Anexos 45

8.1 Game Design Document 45

8.2 Manual de usuario 55

1 Introducción

El objetivo principal de este trabajo de fin de grado o TFG es la creación de un videojuego desde cero abordando todos los aspectos necesarios tanto en el enfoque técnico como en el artístico.

Para explorar esta idea, se ha propuesto desarrollar un videojuego de carreras en 3D denominado "Syncopia". Este videojuego toma gran parte de su inspiración de otras sagas de videojuegos de carreras futuristas como Wipeout[1] o F-Zero[2]. Las principales características de estas sagas son su estilo artístico y musical futurista (ver figura 1) combinado con naves antigravitatorias capaces de alcanzar una gran velocidad y unos controles basados en la física. En este trabajo se han intentado replicar estas características, pero con un estilo visual y musical más retro basado en el movimiento *Synthwave*[3].

Figura 1. Wipeout HD

1.1 Objetivo y alcance

Durante el desarrollo se van a implementar las características comunes a los videojuegos del género de carreras como son la inteligencia artificial capaz de reaccionar y adaptarse a los cambios que se producen en la partida y la inclusión de un modo multijugador en red para poder competir entre varias personas. Además, se van a incluir otras técnicas graficas no tan comunes como el *supersampling*[4]

[1] https://en.wikipedia.org/wiki/Wipeout_(video_game)
[2] https://en.wikipedia.org/wiki/F-Zero
[3] https://en.wikipedia.org/wiki/Synthwave
[4] https://en.wikipedia.org/wiki/Supersampling

donde se pinta cada imagen a una resolución mayor y se escala a la resolución nativa de la pantalla y la resolución dinámica donde se ajusta la resolución en tiempo de ejecución en función del rendimiento.

1.2 Metodología y alcance

El trabajo se enmarca en un contexto técnico enfocado en la resolución de los objetivos previamente planteados. Para ello se propone seguir una metodología ágil para la gestión del trabajo donde se adopte una estrategia de desarrollo incremental.

La herramienta principal a utilizar será el motor gráfico Unity[5]. Unity es uno de los motores gráficos más utilizados en los desarrollos de videojuegos actuales. Dispone de un entorno gráfico con una interfaz muy intuitiva que permite programar tanto en *Javascript* como en *C#*, siendo este último el lenguaje de programación elegido debido a su sencillez y al parecido con otros lenguajes de los que ya se tiene un conocimiento previo como *Java*. Una de sus mayores bazas es el desarrollo multiplataforma que permite disponer de un amplio abanico de posibilidades que van desde los dispositivos móviles hasta la consolas u ordenadores personales. Dispone de una tienda integrada de *assets*[6] que ofrece mejoras, tanto gratuitas como de pago, creadas por la comunidad. Además, ofrece un modelo de precios que va desde una versión gratuita donde se limitan algunos aspectos comerciales, hasta distintas versiones con suscripciones mensuales (Plus, Pro y Enterprise) donde se van reduciendo estas limitaciones y se proporciona un soporte más personalizado.

Además, se usarán otras herramientas como GitHub[7] para el control de versiones o Blender[8] para la creación de modelos en 3D.

1.3 Motivación

La principal motivación para realizar el presente proyecto ha sido la creación de una aplicación que requiere utilizar técnicas muy diferentes de diversos campos combinadas en el mundo de la informática tales como: los gráficos, la inteligencia artificial, la ingeniería del software, la infraestructura de red, etc.

Por todas estas técnicas, este trabajo de fin de grado constituye una gran forma de poner en práctica la mayoría de los conocimientos que se han adquirido durante la carrera.

---

[5] https://unity3d.com/es/
[6] https://www.assetstore.unity3d.com
[7] https://github.com/Arafo/Syncopia
[8] https://www.blender.org/

Además, el interés personal por el género de carreras sumado a la posible dedicación en un futuro a la industria del videojuego, ha motivado a escoger este tema como trabajo de fin de grado.

2 Análisis

2.1 Requisitos

A continuación, se presentan los requisitos funcionales y no funcionales que se plantearon al comienzo del trabajo (ver tablas 1 y 2). En estas tablas, además de los propios requisitos, se detalla la prioridad que se dio inicialmente a cada requisito y si se ha cumplido o no al final del trabajo.

ID	Descripción	Prioridad	Cumplido
RF1	El videojuego permitirá iniciar una partida rápida para un jugador.	Alta	Sí
RF2	El videojuego permitirá iniciar una partida *arcade* para un jugador.	Alta	Sí
RF3	El videojuego permitirá iniciar una partida contrarreloj para un jugador.	Media	Sí
RF4	El videojuego incluirá un sistema de inteligencia artificial capaz de navegar por los circuitos.	Alta	Sí
RF5	El videojuego incluirá un sistema de inteligencia artificial capaz de interactuar con las demás naves.	Alta	Sí
RF6	El videojuego incluirá un sistema de inteligencia artificial capaz de aprender de forma autónoma distintas trazadas en los circuitos.	Media	No
RF7	El videojuego incluirá un sistema de inteligencia artificial capaz de regular la dificultad de forma dinámica.	Media	Sí
RF8	El videojuego permitirá crear una partida multijugador en red para varios jugadores.	Alta	Sí
RF9	El videojuego permitirá unirse a una partida multijugador en red previamente creada.	Alta	Sí
RF10	El videojuego incluirá uno o varios efectos gráficos que mejoren la calidad de la imagen.	Alta	Si
RF11	El videojuego incluirá uno o varios efectos gráficos que mejoren el rendimiento.	Media	Si

RF12	El videojuego permitirá cambiar opciones de configuración (efectos, audio, etc.).	Media	Si
RF13	El videojuego incluirá un sistema de reproducción de música de forma aleatoria dentro de la partida.	Baja	Si
RF14	El videojuego tendrá función de pausa dentro de la partida.	Media	Si

Tabla 1. Requisitos funcionales

ID	Descripción	Prioridad	Cumplido
RNF1	El videojuego podrá ejecutarse en sistemas operativos Windows.	Alta	Si
RNF2	El videojuego podrá ejecutarse en sistemas operativos Mac.	Baja	No
RNF3	El videojuego podrá ejecutarse en sistemas operativos Linux	Baja	No
RNF4	La aplicación tendrá un sistema para cambiar el idioma.	Media	Si
RNF5	La configuración del videojuego (audio, efectos gráficos, etc.) se cargará de un fichero externo.	Media	Si
RNF6	Las animaciones se basarán en el tiempo y no dependerán de las imágenes por segundo.	Media	Si
RNF7	El control del videojuego podrá ser tanto por teclado y ratón como con un mando.	Media	Si
RNF8	El videojuego mostrará los iconos del dispositivo de control que se esté utilizando.	Baja	No

Tabla 2. Requisitos no funcionales

El requisito funcional RF6 se planteó al inicio del trabajo, pero no se incluye en la versión final debido al alto impacto en el rendimiento del videojuego y a la rigidez del sistema que producía unos resultados no muy buenos. En el apartado 3.3 correspondiente al diseño de la inteligencia artificial se explica en detalle el porqué de esta decisión.

2.2 Casos de uso

En la siguiente figura (ver figura 2) se muestran los casos de uso del videojuego y, a continuación, la especificación en detalle de cada uno de ellos.

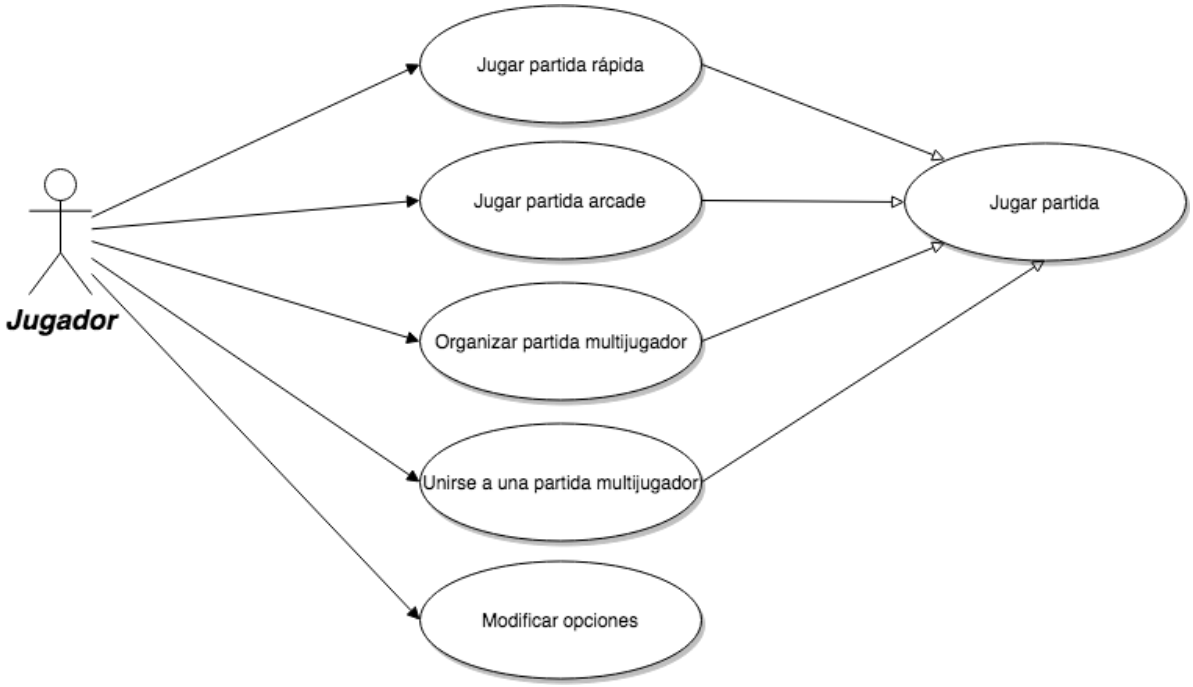

Figura 2. Diagrama de casos de uso

Jugar partida rápida
Descripción: Este caso de uso permite al usuario iniciar una partida en una pista y naves aleatorias, en la dificultad normal con un número de rivales y de vueltas fijo.
Precondiciones: El usuario se encuentra en el menú principal.
Escenario:

1. El usuario pulsa el botón "QUICK RACE" en el menú principal.
2. El sistema inicia la partida.

Jugar partida *arcade*
Descripción: Este caso de uso permite al usuario jugar una partida *arcade* dejando al usuario escoger la pista, la nave, la dificultad, el número de rivales y el número de vueltas.
Precondiciones: El usuario se encuentra en el menú principal.
Escenario:

1. El usuario pulsa el botón "ARCADE" en el menú principal.
2. El sistema muestra la pantalla con las pistas disponibles.
3. El usuario pulsa sobre una de estas pistas.

4. El sistema muestra la pantalla con las naves disponibles.
5. El usuario pulsa sobre una de estas naves.
6. El sistema muestra la pantalla con otras opciones disponibles (dificultad, número de rivales y número de rivales).
7. El usuario selecciona las opciones deseadas y pulsa el botón “READY”.
8. El sistema inicia la partida.

Jugar partida contrarreloj
Descripción: Este caso de uso permite al usuario jugar una partida contrarreloj dejando al usuario escoger la pista, la nave, la dificultad y el número de vueltas.
Precondiciones: El usuario se encuentra en el menú principal.
Escenario:

1. El usuario pulsa el botón “TIME TRIAL” en el menú principal.
2. El sistema muestra la pantalla con las pistas disponibles.
3. El usuario pulsa sobre una de estas pistas.
4. El sistema muestra la pantalla con las naves disponibles.
5. El usuario pulsa sobre una de estas naves.
6. El sistema muestra la pantalla con otras opciones disponibles (dificultad y número de vueltas).
7. El usuario selecciona las opciones deseadas y pulsa el botón “READY”.
8. El sistema inicia la partida.

Organizar partida multijugador
Descripción: Este caso de uso permite al usuario crear una partida multijugador donde el usuario o anfitrión puede elegir la pista, la nave y el color de la nave seleccionada.
Precondiciones: El usuario se encuentra en el menú principal.
Escenario:

1. El usuario pulsa el botón “MULTIPLAYER” en el menú principal.
2. El sistema muestra la pantalla para crear o unirse a una partida multijugador.
3. El usuario pulsa sobre la opción “CREATE”.
4. El sistema muestra la pantalla con las opciones disponibles para un anfitrión en una partida multijugador.
5. El usuario selecciona las opciones deseadas y pulsa el botón “READY”.

6. El sistema espera hasta que la partida tiene los suficientes usuarios para comenzar y, una vez se cumple este requisito, se inicia la partida.

Unirse a una partida multijugador
Descripción: Este caso de uso permite al usuario unirse una partida multijugador donde el usuario o cliente puede elegir la nave y su color.
Precondiciones: El usuario se encuentra en el menú principal.
Escenario:

1. El usuario pulsa el botón "MULTIPLAYER" en el menú principal.
2. El sistema muestra la pantalla para crear o unirse a una partida multijugador.
3. El usuario pulsa sobre la opción "LIST SERVERS".
4. El sistema muestra en pantalla las partidas disponibles. Si no existe ninguna partida, se muestra el mensaje "NO SERVERS FOUND".
5. El usuario selecciona la partida a la que quiere unirse.
6. El sistema muestra la pantalla de las opciones disponibles para un cliente en una partida multijugador.
7. El usuario selecciona las opciones deseadas y pulsa el botón "READY".
8. El sistema espera hasta que la partida tiene los suficientes usuarios para comenzar y, una vez se cumple este requisito, se inicia la partida.

Modificar opciones
Descripción: Este caso de uso permite al usuario cambiar distintas opciones del videojuego.
Precondiciones: El usuario se encuentra en el menú principal.
Escenario:

1. El usuario pulsa el botón "OPTIONS" en el menú principal.
2. El sistema muestra por pantalla las opciones disponibles: "GAMEPLAY", "AUDIO", "GRAPHICS" y "CONTROLS".
3. El usuario elige una de las opciones para las que quiere modificar algún parámetro.
4. El sistema muestra las opciones detalladas de la opción que ha elegido el usuario.
5. El usuario modifica las opciones que desea.
6. El sistema actualiza las opciones modificadas y almacena su nuevo valor en un fichero persistente.

3 Diseño

3.1 Arquitectura del sistema

En esta sección se detalla la estructura del videojuego (ver figura 3), incluyendo todas las partes en las que está dividido el código.

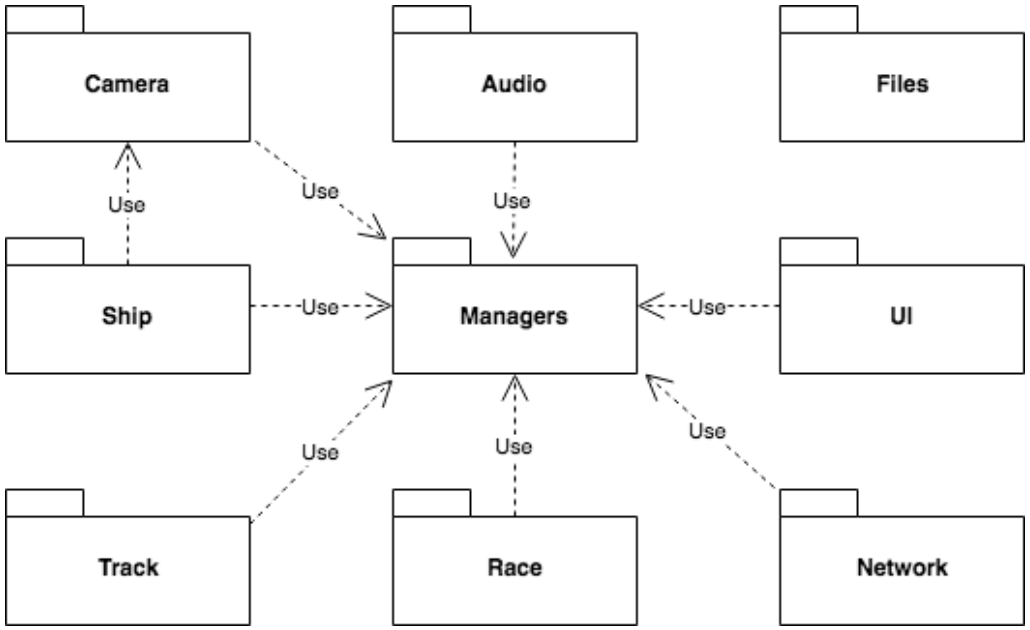

Figura 3. Diagrama de paquetes

Paquete "Audio"

- AudioSettings: Gestiona la lectura y el almacenamiento de la lista de canciones que se reproducen durante las carreras. Además, también proporciona acceso al gestor que se encarga de reproducir la música en las carreras.
- MusicSingleton: Gestiona la reproducción de la música en los menús.

Paquete "Camera"

- DynamicResolution: Implementa la técnica de la resolución dinámica.
- ShipCamera: Gestiona el movimiento de la cámara de una nave.
- SuperSampling: Implementa la técnica del *supersampling* (SSAA).
- TextureRenderer: Gestiona la resolución y el contenido de la textura que utiliza la técnica del *supersampling*.

Paquete "Files"

- BuildNumber: Gestiona el número de compilación cada vez que se ejecuta el proyecto en el editor de Unity.

Paquete "Managers"

- AntiAliasingManager: Gestiona todas las técnicas de *antialiasing* utilizadas de forma que no interfieran unas con otras.
- AudioManager: Gestiona el registro, la reproducción y la destrucción de todos los efectos de sonido que se usan en los menús y durante las carreras.
- BoardManager: Se ocupa de controlar el contenido de los letreros que indican las vueltas que faltan en una carrera.
- ClipManager: Crea todas las fuentes de sonido que se utilizan durante las partidas.
- Enumerations: Almacena todas las enumeraciones utilizadas (dificultad, naves, modos de juego, etc.).
- GameOptions: Gestiona la lectura y escritura de los ajustes de las opciones en el fichero de configuración.
- LanguageManager: Gestiona el cambio de idioma en tiempo de ejecución y almacena las cadenas del lenguaje elegido.
- LanguageSingleton: Asegura que solo existe una instancia de la clase *LanguageManager* e inicia el cambio de idiomas.
- LoadingScreenManager: Gestiona la carga de la partida y la pantalla que se muestra mientras se carga.
- MenuAnimationManager: Gestiona las animaciones de transición entre los distintos menús.
- MusicManager: Gestiona la carga, reproducción y pausa de la música que suena durante las carreras.
- PauseManager: Gestiona la pantalla de resultados que aparece al terminar una partida y el cambio de todas las opciones que se pueden realizar en el menú de pausa.
- RaceManager: Gestiona todas las acciones que ocurren durante una partida. Desde el inicio donde se carga todo lo necesario para empezar una carrera, hasta la ejecución donde se deben actualizar las posiciones de las naves.
- SceneIndexManager: Proporciona una estructura para obtener el índice de las escenas por su nombre.

Paquete "Network"

- EventSystemChecker: Asegura que solo existe una instancia de la clase *EventSystem* de Unity que se encarga de gestionar la entrada de los controladores en el modo multijugador.
- LobbyHook: Proporciona una función para gestionar el paso de datos entre el menú multijugador y la partida.
- LobbyManager: Gestiona todas las conexiones que realizan entre el cliente y el servidor cuando se organiza una partida multijugador.
- LobbyMenu: Gestiona la creación y la unión a partidas multijugador.
- LobbyPlayer: Gestiona todas las opciones que puede cambiar un jugador en una partida multijugador.
- LobbyPlayerList: Gestiona la lista de jugadores que forman parte de una partida multijugador.
- LobbySeverEntry: Gestiona la información que muestra cada partida multijugador en la lista de partidas.
- LobbyServerList: Gestiona la lista de partidas multijugador disponibles que han creado los jugadores.
- LobbyTopPanel: Gestiona la información del servidor al que se conecta un cliente.
- NetworkedPlayer: Se ocupa de configurar las naves en la partida multijugador tanto para el servidor como para el cliente
- NetworkGameManager: Gestiona todas las acciones que se realizan durante una partida multijugador y requieren estar sincronizadas entre todos los integrantes de la partida.
- NetworkLobbyHook: Gestiona el paso de datos entre el menú multijugador y la partida.
- NetworkPlayerController: Implementa la interpolación utilizada en una partida multijugador.
- NetworkShipSetup: Almacena la información de un cliente en una partida multijugador.

Paquete "Race"

- RaceSettings: Almacena todas las referencias que forman parte de una carrera.

Paquete "Ship"

- ShipAI: Gestiona el comportamiento de la inteligencia artificial que utilizan las naves.
- ShipAudio: Gestiona el efecto de sonido que produce el motor de una nave.
- ShipConfig: Estrucutra que contiene todos los parámetros que define la física de una nave.
- ShipCore: Define la estructura que heredan el resto de clases que componen una nave.
- ShipInput: Gestiona la entrada del dispositivo controlador.
- ShipLoader: Gestiona la carga de todos los componentes que definen a una nave.
- ShipPosition: Se encarga de gestionar la posición de la nave en la pista.
- ShipReferer: Clase principal de una nave. Controla el inicio y ejecución de todos los componentes de la nave.
- ShipSimulation: Gestiona la simulación de la física de las naves.
- ShipTrailEffects: Gestiona el efecto de la estela de las naves.

Paquete "Track"

- Data: Estructura para almacenar toda la información que compone una pista.
- GenerateTrackData: Algoritmo que se encarga de leer el modelo de una pista y generar los datos correspondientes.
- Helpers: Clase auxiliar para realizar distintas operaciones con los datos de una pista.
- TrackData: Gestiona los datos generados de una pista y además ofrece una representación visual de estos datos.
- TrackDataEditor: Editor visual de los datos generados sobre una pista.
- TrackDataEditorMono: Clase auxiliar del editor de los datos generados.
- TrackSegment: Clase que representa un segmento de una pista.
- TrackTile: Clase que representa un *tile* de una pista.

Paquete "UI"

- BackListener: Gestiona las referencias que se utilizan en el gestor de animaciones de los menús.
- ConfigButton: Gestiona las animaciones y los colores de los botones.
- HorizontalScrollSlider: Gestiona el comportamiento de las flechas de un botón horizontal.

- HorizontalScrollSliderButton: Gestiona el funcionamiento de la opción seleccionada en un botón horizontal.
- MenuEventManager: Gestiona las distintas opciones que se pueden elegir en los menús.
- RaceUI: Gestiona la información que aparece en pantalla durante una carrera (HUD).
- TextLang: Componente de texto que se actualiza en función del idioma seleccionado.

3.2 Física de las naves

Una de las primeras tareas realizadas, consistió en diseñar la física sobre las que se sustenta el funcionamiento de las naves. Esta física se basa en su mayor parte en la fuerza antigravitatoria, la cual, es una fuerza teórica predicha por las leyes de la física de altas energías que consiste en la repulsión de todos los cuerpos debido a una fuerza que es igual en magnitud a la gravedad, pero en vez de ser atractiva, es repulsiva.

Esta fuerza antigravitatoria se combina con el movimiento de una aeronave capaz de rotar alrededor de tres ejes perpendiculares: eje transversal (Y), eje longitudinal (X) y eje vertical (Z).

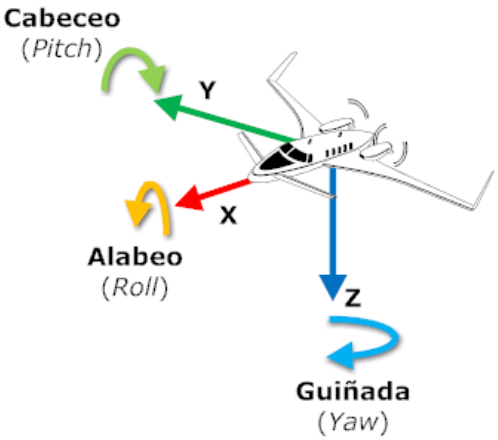

Figura 4. Ejes de movimiento

A partir de estos tres ejes se define los tipos de rotaciones que una nave puede realizar (ver figura 4):

- El cabeceo (*Pitch*) es la rotación respecto del eje transversal de la nave. En este caso, como la nave solo despega del suelo en condiciones de pendiente, este movimiento está limitado.
- El alabeo (*Roll*) es la rotación respecto del eje longitudinal.

- La guiñada (*Yaw*) es una rotación respecto del eje vertical.

Estas dos últimas rotaciones se combinan en un único movimiento para producir una mayor sensación de cambio en la dirección.

3.3 Arquitectura de la inteligencia artificial

La inteligencia artificial en un videojuego de carreras se basa en emular el comportamiento de los jugadores de forma realista. El concepto principal de la inteligencia artificial es la toma de decisiones. Para tomar estas decisiones, el sistema necesita ser capaz de visualizar el entorno y el resto de entidades que le rodean. Para que estas decisiones sean significativas, el sistema de inteligencia artificial necesita alguna forma de poder percibir el entorno y, más en concreto, el resto de naves que forman parte de una carrera. Teniendo en cuenta todo esto, se ha decidido diseñar un sistema de inteligencia artificial adaptativo ya que el videojuego requiere cierta variabilidad y rivales dinámicos capaces de adaptarse por sí mismos.

A continuación, se presenta el diagrama con el funcionamiento de la inteligencia artificial pudiendo observar como interaccionan unos sistemas con otros (ver figura 5):

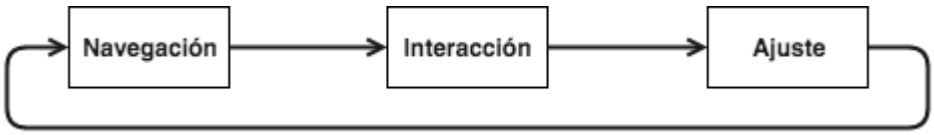

Figura 5. Arquitectura de la inteligencia artificial

- Navegación: la navegación está diseñada de tal forma que utiliza la forma del modelo 3D del circuito para decidir la dirección de la nave.
- Interacción: modela las interacciones que se producen entre las naves y, más en concreto, los adelantamientos mediante un sistema de percepción.
- Ajuste: balancea la velocidad de cada nave en función de su posición en la carrera.

En una primera fase del trabajo se incluyó un sistema capaz de aprender de forma autónoma distintas trazadas en los circuitos (requisito funcional RF6). Este sistema se implementó mediante un algoritmo genético donde la nave era capaz de aprender trazadas subóptimas. Sin embargo, surgieron varios problemas con este sistema.

En primer lugar, la ejecución de este algoritmo no era posible en tiempo de ejecución. El rendimiento del juego caía de forma muy brusca cuando había un

número elevado de naves controladas por la inteligencia artificial. Usa solución fue limitar el sistema a construir trazadas predefinidas y almacenarlas en ficheros para poder utilizarlas más tarde en una partida en tiempo real.

En segundo lugar, el sistema era muy rígido y cuando se utilizaba con el sistema de interacción producía movimientos erráticos. Debido a estos problemas se decidió prescindir de este sistema.

4 Desarrollo

Una de las partes más complejas y largas del proyecto fue el desarrollo. El conocimiento de la plataforma de desarrollo no era muy elevado por lo que al principio se invirtió una gran cantidad de tiempo no productivo consultando tutoriales y documentación oficial antes de comenzar. Una vez se dieron por suficientes los conocimientos adquiridos sobre la plataforma, se comenzó con el desarrollo con un cierto nivel de seguridad.

4.1 Física de las naves

A continuación, se presenta un diagrama con el funcionamiento completo de la física implementada (figura 6):

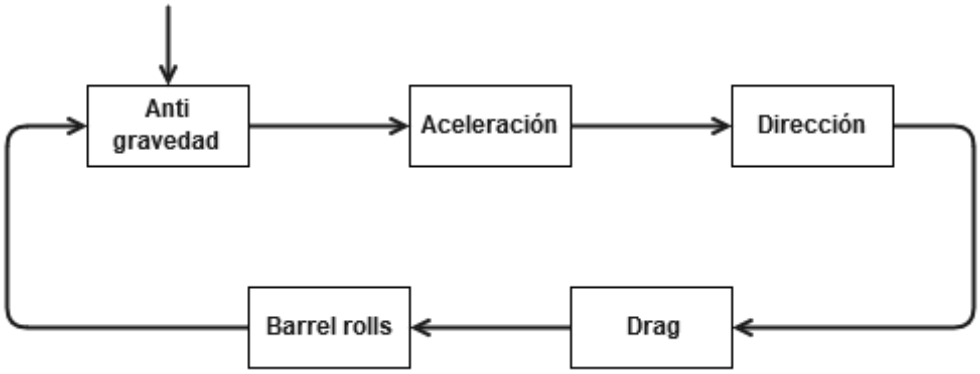

Figura 6. Diagrama de la física

- Antigravedad: Para simular el fenómeno de antigravedad (ver figura 7), se ha utilizado una estructura rectangular utilizando las dimensiones de la nave. En cada uno de los vértices de esta estructura se ha colocado un vector dirigido hacia la superficie sobre la que se quiere flotar. A partir de cada uno de estos vectores, se lanza un rayo en dicha dirección. Si este rayo intersecta con la superficie del circuito significa que la nave está en el "suelo". Cuando la nave está en el suelo, se calcula la fuerza a aplicar en cada vector de la estructura para que esta siga flotando. Por el contrario, si alguno de los rayos lanzados desde los vectores de la estructura hacia la superficie del circuito no produce una intersección en una cierta distancia, se asume que esa parte de la estructura no está en el suelo y se aplica una fuerza mayor

hacia la superficie del circuito. Por último, para dar una sensación de mayor flotación se permite que el origen los rayos lanzados tengan un cierto umbral.

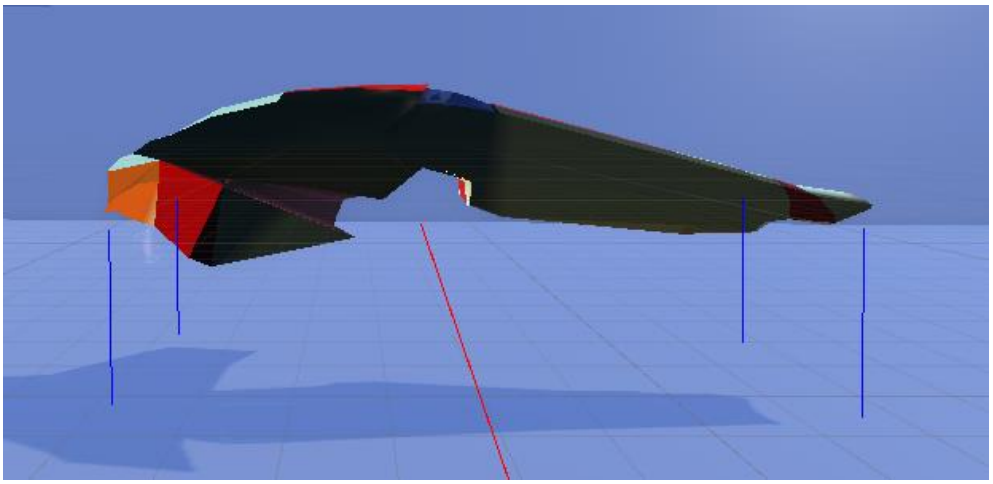

Figura 7. Estructura antigravedad

- Aceleración: La aceleración de la nave se ha implementado utilizando una simulación de un motor de combustión simplificado. Se han utilizado cuatro atributos para similar un motor: la ganancia, la aceleración, el empuje y la caída del motor. Para cada uno de estos atributos solo existen dos estados, la nave acelera o la nave desacelera. En el caso de que la nave acelera, la ganancia, la aceleración y el empuje se interpolan (*Lerp*) entre sus valores actuales y sus valores máximos. La caída en este caso siempre es cero. Estos valores máximos son fijos para cada nave.

$$ganancia = Lerp\ ganacia, gananciaMax, timeFrame$$
$$aceleracion = Lerp(aceleracion, aceleracionMax, timeFrame * ganacia)$$
$$empuje = Lerp(empuje, empujeMax, timeFrame * aceleracion)$$
$$caida = 0$$

  En el caso de que la nave desacelera, la caída se interpola la caída entre su valor actual y el máximo y la aceleración y el empuje se interpolan entre los valores actuales y cero. La ganancia en este caso siempre es cero.

$$caida = Lerp(caida, caidaMax, timeFrame)$$
$$aceleracion = Lerp(aceleracion, 0, timeFrame * caida)$$
$$empuje = Lerp(empuje, 0, timeFrame * caida)$$
$$ganancia = 0$$

- Dirección: La dirección de la nave se calcula a partir de tres atributos: la ganancia, la cantidad y la caída del giro. Dependiendo de si la nave gira hacia la izquierda, hacia la derecha o no se produzca ningún movimiento, el valor de estos atributos se interpola entre unos valores u otros. El atributo

que define la dirección de giro es la cantidad. La ganancia y la cantidad de giro se interpolan entre sus valores actuales y los valores máximos de la nave siempre que se produzca movimiento. La caída es cero cuando hay movimiento.

$$ganancia = Lerp(ganancia, gananciaMax, timeFrame)$$
$$cantidad = Lerp(cantidad, cantidadMax * giro, timeFrame * ganancia)$$
$$caida = 0$$

En el caso de que no se produzca ningún tipo de movimiento, se interpola la caída entre su valor actual y el máximo, la cantidad se interpola desde su valor actual hasta cero y la ganancia siempre es cero.

$$caida = Lerp(caida, caidaMax, timeFrame)$$
$$cantidad = Lerp(cantidad, 0, timeFrame * caida)$$
$$ganancia = 0$$

- Resistencia al aire o *drag*: La resistencia al aire o *drag* se puede producir de dos formas. La primera forma es utilizar los frenos laterales de nave. Estos frenos son independientes y actúan con los alerones de un avión de forma que la propia resistencia del freno lateral provoca que la nave gire. La segunda forma es la propia inercia de la nave cuando se deja de acelerar. En todo momento que la nave está acelerando, actúa sobre ella una fuerza en sentido contrario de forma que simula la resistencia al aire.

- *Tonel* o *Barrel roll*: Cuando la nave no está en el suelo, se permite realizar un giro sobre si misma llamado *Tonel* o *Barrel roll*. Esta acrobacia se controla con una máquina de estados donde se va acumulando los grados de giro hasta que se produce un giro completo de 360º.

4.2 Inteligencia artificial

En esta sección se explica el funcionamiento de la inteligencia artificial. Inicialmente se van a cubrir los tres conceptos principales sobre los que se sustenta el sistema: navegación por el entorno, la interacción con los rivales y el ajuste dinámico de la dificultad.

4.2.1. Navegación en el entorno

Para entender cómo funciona la navegación por el entorno, es necesario explicar cómo se representa un circuito (ver figura 8).

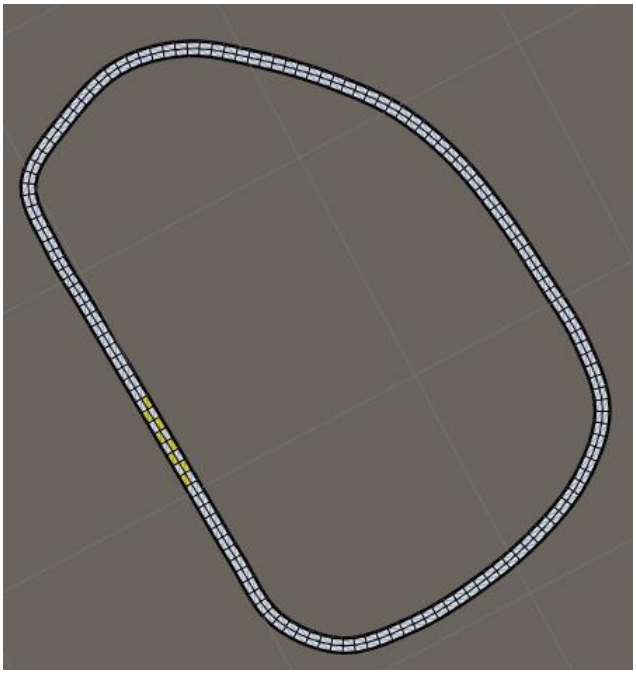

Figura 8. Representación de un circuito

Esta representación sigue una estructura incremental donde se utiliza la propia geometría del modelo 3D del circuito. Cada triángulo de la malla representa un *tile* y a partir los *tiles* adyacentes de forma horizontal se constituye un segmento. El proceso de generar estos *tiles* y segmentos es un proceso offline y está previamente calculado para cada circuito.

Una vez tenemos esta representación, podemos saber en todo momento en que *tile* y en que sección se encuentra cada nave. El proceso de navegación sigue el siguiente esquema:

- Al inicio de la carrera se establecen unos valores aleatorios para evitar que el movimiento de las naves rivales sea siempre el mismo. Estos valores se corresponden a un umbral de movimiento lateral, un umbral de la velocidad de giro y un umbral de la velocidad máxima de la nave.
- En cada *frame* (imagen por segundo) cada nave rival establece su estado como que está acelerando. Por defecto, este es el comportamiento de todas las naves rivales.
- Con la representación de circuito mediante *tiles* y segmentos, se obtiene la posición exacta del segmento en el que se encuentra cada nave rival.

- A continuación, a partir de esta sección se busca en la representación del circuito la posición de otra sección que está situada por delante a un número de secciones determinadas de la inicial (cuatro por defecto). Para que la posición en esta nueva sección no sea siempre la misma se utiliza el umbral de varianza en la trazada del circuito.
- A partir de estas dos secciones se calcula un vector que representa la dirección de rotación que debe de realizar cada nave para poder girar en el sentido que establece la sección situada delante.
- Por último, se aplica en la nave la dirección de dirección de rotación. Para que los giros no se produzcan de manera brusca, se interpola un valor intermedio entre la dirección de rotación actual y la calculada.

Aunque esta rotación ya permite a las naves rivales navegar por el entorno con cierta aleatoriedad, solo se define la rotación en el eje X mientras que el jugador además de rotar en este eje, también rota a la vez en el eje Z. Como esta rotación en el jugador se produce gracias al dispositivo de entrada, es necesario modelar esta rotación en las naves controladas por la inteligencia artificial de forma manual mientras se produce el giro.

4.2.2. Interacciones con los rivales

Para producir una inteligencia artificial realista es necesario simular las interacciones que tiene una nave rival con el jugador y con el resto de naves rivales. En concreto, es necesario simular la interacción principal que se produce en una carrera como son los adelantamientos.

Antes de poder llevar a cabo un adelantamiento con la inteligencia artificial, ha sido necesario crear un sistema de percepción (ver figura 9) alrededor de la nave capaz de identificar en todo momento los objetos que tiene a su alrededor. Este sistema se define de la siguiente forma:

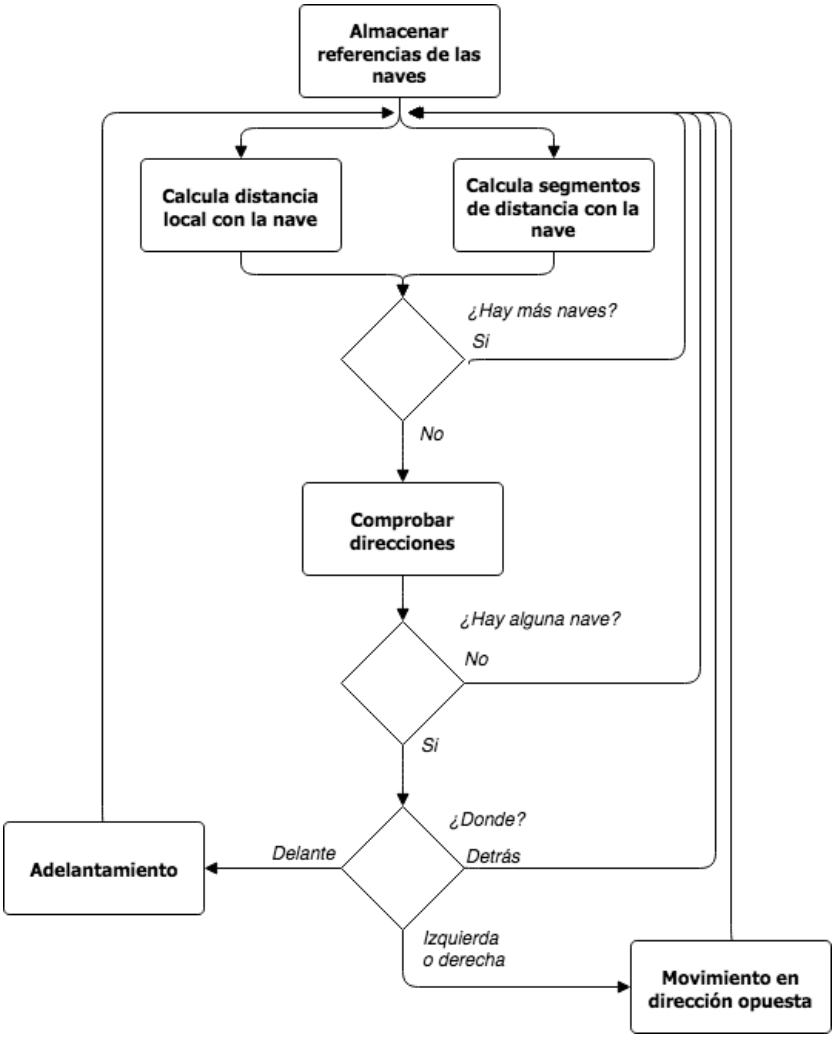

Figura 9. Diagrama de interacción de la inteligencia artificial

1. Al iniciar una carrera cada nave almacena en una estructura de datos las referencias de todas las naves que forma parte de esta carrera exceptuando la misma nave que contiene esta estructura de datos.
2. En cada *frame* (imagen por segundo) cada una de las naves que participa en la carrera calcula para cada una de las naves referenciadas en la estructura de datos previamente construida dos valores: la distancia local entre las naves y la distancia entre los segmentos donde se encuentran las naves.
3. En ese mismo *frame* y con las distancias que hemos calculado, se procede a comprobar si una nave percibe algún objeto en cuatro direcciones: adelante, detrás, izquierda y derecha. Para realizar estas comprobaciones se comparan las distancias entre las naves y los segmentos con unos umbrales que definen como de cerca o de lejos pueden estar los objetos para que puedan ser identificados por la nave.

4. Por último, si una nave percibe un objeto en alguna de las direcciones que se comprueban, esta puede realizar cuatro acciones dependiendo de la dirección:
    - Objeto delante: si una nave percibe otra nave delante significa que está lo suficientemente cerca como para ejecutar una maniobra de adelantamiento. Esta maniobra se explica después.
    - Objeto detrás: si una nave percibe otra nave detrás no se realiza ninguna acción.
    - Objeto a la izquierda: si una nave percibe otra a su izquierda, se aplica sobre la nave una fuerza en dirección opuesta hasta que dicha nave deja de percibir a la otra.
    - Objeto a la derecha: si una nave percibe otra a su derecha, se aplica sobre la nave una fuerza en dirección opuesta hasta que dicha nave deja de percibir a la otra.

La maniobra de adelantamiento comienza cuando una nave detecta, mediante el sistema de percepción, otra nave delante.

1. En primer lugar, se establece la nave objetivo que se quiere adelantar y el lado por el que se va a realizar el adelantamiento. Este lado se calcula mediante la distancia local entre las naves eligiendo el lado cuya distancia es menor, es decir, el más cercano.
2. Se establece un contador de tiempo para limitar el tiempo que tarda la maniobra y que no se prolongue de manera indefinida. Este contador se inicia por defecto a tres segundos y medio.
3. Mientras este contador sea mayor que cero, la nave que está realizando el adelantamiento comprueba que la distancia local entre ella y la nave a adelantar sea menor que un umbral. Si la distancia es mayor que el umbral establecido quiere decir que la nave a adelantar se está alejando y se procede a disminuir el contador en un segundo.
4. Por último y solo si el contador de tiempo es mayor que cero, se calcula un vector de fuerza a partir del lado por el que se quiere realizar el adelantamiento y se aplica a la nave que va a realizar la maniobra.

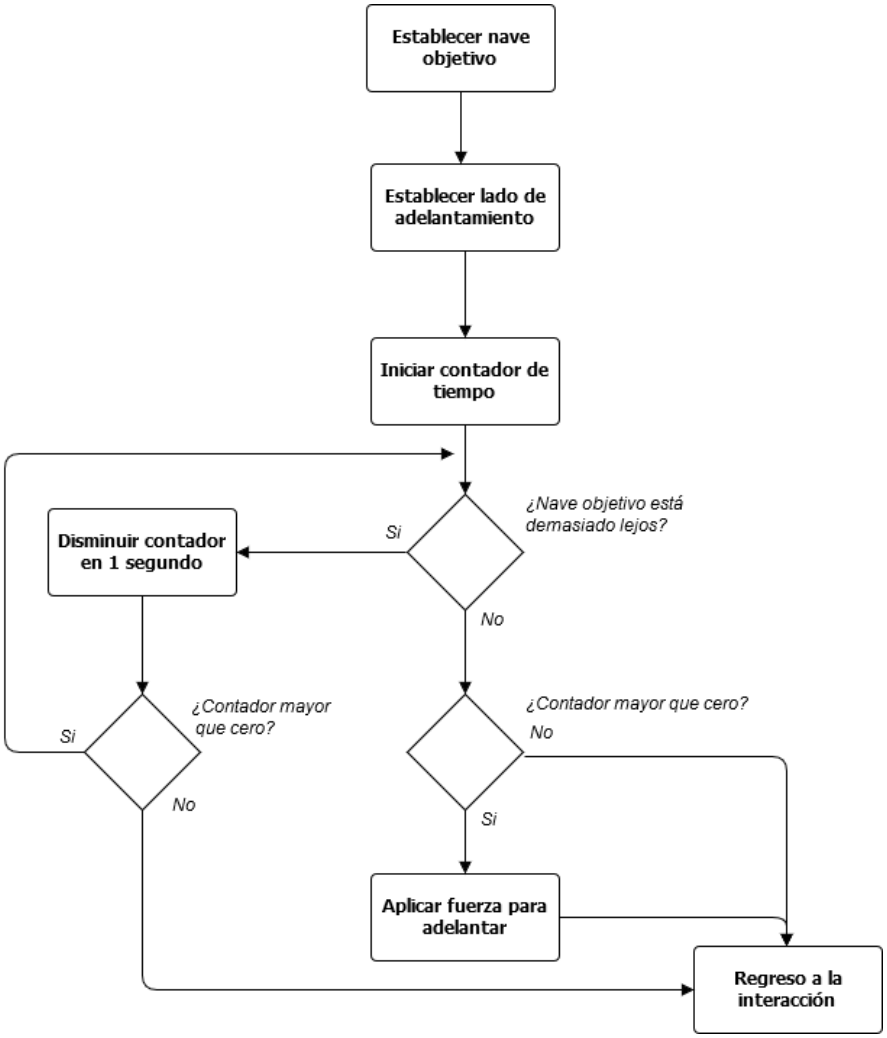

Figura 10. Diagrama del proceso de adelantamiento

Para no prolongar el adelantamiento durante demasiado tiempo, el contador también disminuye en cada *frame* el tiempo que se ha tardado en pintar el *frame* anterior.

4.2.3. Ajuste dinámico de la dificultad (*Rubber Band*)

Para crear un sistema de inteligencia artificial que proporcione carreras interesantes y equitativas es necesario retar al jugador durante todo el transcurso de una carrera, manteniendo a los rivales cerca pero no castigándolos demasiado cuando cometen un error. Las carreras se tienen que gestionar de forma que siempre supongan un desafío para el jugador y que este no siempre se encuentre por delante de los rivales durante todo el transcurso de la carrera o se situé en la última posición y se quede ahí.

Para crear este sistema se ha utilizado la técnica conocida como *Rubber Band*. Esta técnica produce un ajuste dinámico de la dificultad intentando mantener la

tensión y la emoción de la carrera mantenido a los rivales controlados por la inteligencia artificial siempre alrededor del jugador.

En este caso, se ha diseñado un sistema donde las naves rivales son capaces de ajustar de forma dinámica el empuje máximo del motor en función de la posición en la que se encuentran (ver figura 11). Es decir, en una carrera las naves controladas por la inteligencia artificial que se encuentran por detrás de la nave del jugador disponen de un mayor empuje del motor mientras las naves que se encuentran por delante del jugador disminuyen de forma drástica el empuje del motor.

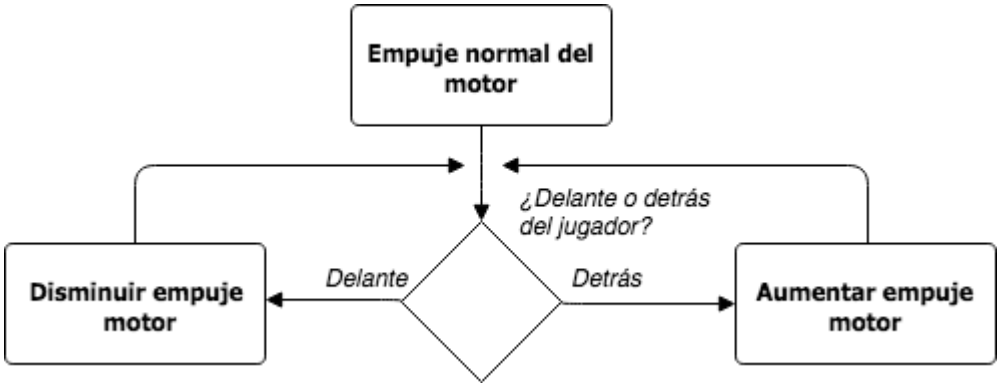

Figura 11. Diagrama del ajuste dinámico de la dificultad

Para minimizar la detección de estos ajustes dinámicos ya que, si se producen de una forma muy brusca son fáciles de detectar y se pueden percibir por el jugador como algo poco justo, los cambios entre los distintos valores para el empuje del motor se suavizan para que apenas se puedan apreciar.

4.3 Multijugador en red

Para crear un sistema que permita la interacción de varios jugadores al mismo tiempo en distintas máquinas, se ha utilizado la API[9] que proporciona Unity para construir un sistema de multijugador en red. Esta API proporciona una capa de abstracción para poder comunicarse entre los jugadores. Esta comunicación hace uso de un servidor intermedio que proporciona Unity y que permite conectar a los distintos clientes con un servidor sin utilizar direcciones IP como identificadores.

El sistema utiliza una arquitectura cliente-servidor, lo que significa que las máquinas no se conectan directamente unas con otras, sino que una de las maquinas actúa como servidor y el resto de máquinas actúan como clientes. El servidor puede actúar también como cliente lo que permite que no sea necesario utilizar un servidor dedicado.

[9] https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

A continuación, se va a presentar como se establecen las conexiones entre los jugadores antes de iniciar una partida (ver figura 12). En primer lugar, un jugador debe de iniciar un *lobby* o partida multijugador. El inicio de un *lobby* envía una petición desde la máquina del jugador al servidor intermedio que proporciona Unity indicando distintos parámetros del *lobby* como el nombre, el máximo número de jugadores que puede albergar o si la partida va a ser pública. El jugador que inicia este *lobby* se considera como el servidor de la partida.

Cuando otro jugador se quiere unir a una partida y por tanto establecer una conexión con el servidor, envía una petición al servidor intermedio y este envía la información sobre las partidas disponibles. El jugador puede elegir unirse a cualquiera de estas partidas. Para ello, el jugador envía otra petición al servidor intermedio indicando la partida a la que quiere unirse y este puede contestar de dos formas: el servidor intermedio acepta la petición y el jugador pasa a ser parte del *lobby* como un cliente o el servidor intermedio deniega la petición y no permite al jugador establecer la conexión con el servidor. La petición puede ser denegada porque se pierde la conexión a internet o porque la partida ha alcanzado el número máximo de jugadores.

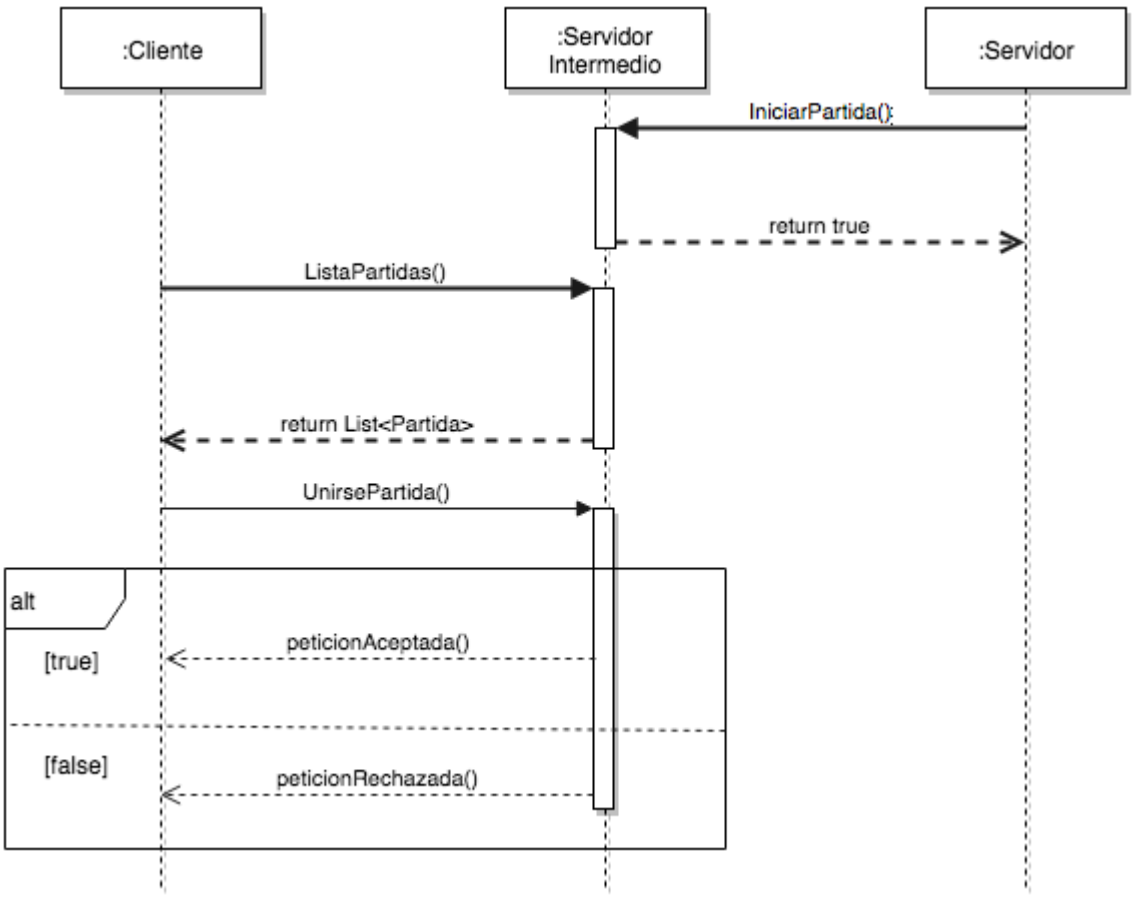

Figura 12. Diagrama de secuencia para establecer la conexión

4.3.1. Interpolación

Una vez se ha establecido la conexión entre los jugadores que forman parte del *lobby* la partida puede comenzar. Durante el trascurso de la partida es necesario sincronizar distintas variables entre los clientes y el servidor. En un principio se utilizó el componente *NetworkTransform* que propone Unity para sincronizar las variables. El problema de utilizar este componente es que, al ser genérico, sincroniza una gran cantidad de variables que no son necesarias y provoca un retardo o *lag* en el paso de mensajes entre los clientes y el servidor lo que se traduce en una sensación de movimiento poco fluido.

Para solucionar este problema se ha implementado un sistema de sincronización propio utilizando interpolación (ver figura 13). En este caso en particular solo es necesario sincronizar dos variables para cada nave: su posición y su rotación.

El comportamiento de este sistema varía en función de si las variables de la nave a sincronizar corresponden con las del jugador local o son las de otro jugador en una maquina distinta.

En el caso en el que el jugador local necesite sincronizar las variables de movimiento y rotación de la nave, se envía una vez cada 11 milisegundos al servidor intermedio el valor de estas variables y el servidor intermedio es el encargado de reenviarlas a todos los clientes que forman parte del lobby.

En el caso en el que las variables a sincronizar pertenezcan a otro jugador, el servidor intermedio envía cada 11 milisegundos el valor de dichas variables y estas se asignan a la posición y rotación de la nave indicada. Para que no se produzcan saltos debidos al retardo o lag en este envío de mensajes, las variables sincronizadas no se asignan de forma directa, sino que se interpola su valor entre el valor que tienen actualmente y el nuevo valor sincronizado. Esta interpolación no sigue el intervalo de sincronización de 11 milisegundos, sino que se realiza en cada *frame* porque el movimiento y la rotación de una nave deben de ser independientes de la conexión entre los jugadores.

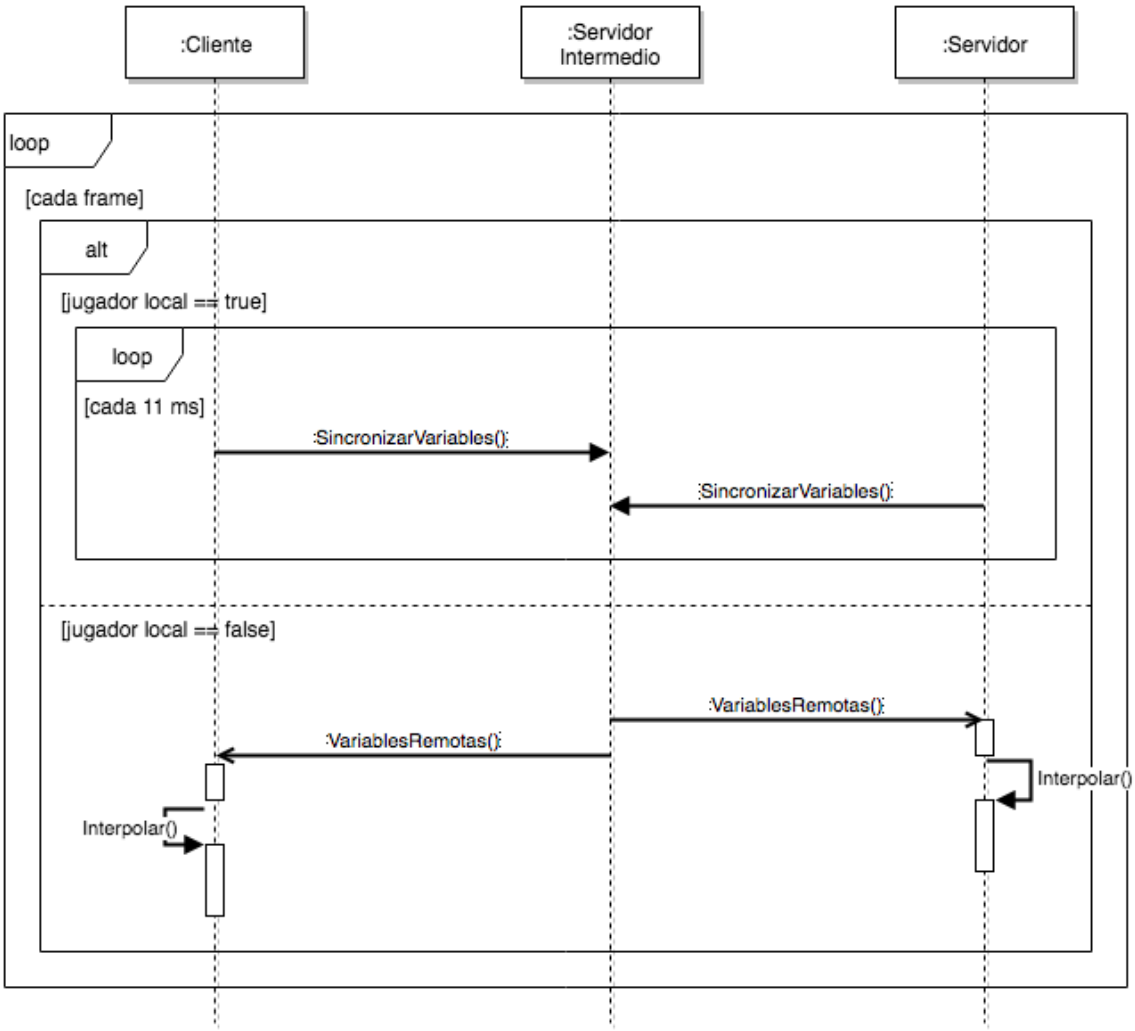

Figura 13. Diagrama de secuencia interpolación

4.4 Efectos gráficos

Para mejorar la calidad de imagen se ha implementado una técnica de *antialisaing* llamada *Supersampling* o SSAA. Para mejorar la estabilidad a nivel de *framerate* también se ha implementado una técnica conocida como resolución dinámica o *Dynamic Resolution Scaling*. A continuación, se explican ambas técnicas de forma detallada.

4.4.1. *Supersampling* (SSAA)

La implementación clásica de esta técnica de *antialiasing* utiliza un *viewport*[10] para restringir la parte que se muestra en pantalla a una porción del tamaño máximo también llamado *render target* y, a continuación, escala la imagen al tamaño del *viewport*.

[10] https://en.wikipedia.org/wiki/Viewport

El problema con esta implementación es que Unity limita el acceso al *viewport* que define una cámara y solo permite la lectura de algunos de sus parámetros. Para solventar esta limitación se ha utilizado un sistema de dos cámaras con una textura donde se va a generar la imagen la final y un *shader*[11] de escalado.

La primera cámara es la que usa cada nave por defecto. Esta cámara está situada detrás de cada nave y enfoca la escena en la dirección de la nave de forma que compone la vista que ve el jugador por defecto.

Unity permite dibujar en una cámara una textura superpuesta. El principal uso de esta textura es superponer imágenes como mapas del entorno o gráficos de la interfaz. En este caso se va a utilizar como un *viewport* ya que la textura sí que se puede escalar en tiempo real.

La segunda cámara se usa como un buffer de la primera cámara. Esta cámara se crea en tiempo real y toma sus propiedades de la primera cámara incluida la textura.

A partir de estos componentes se ha implementado la técnica del *supersampling* de la siguiente forma:

- En el inicio de cada *frame*, se obtiene una referencia de la primera cámara que corresponde a la cámara del jugador, se carga el *shader* de escalado y se define la resolución objetivo a la que se quiere aplicar la técnica.
- A continuación, se crea la textura vacía utilizando esta resolución objetivo y la segunda cámara utilizando los parámetros de la primera. Para que todo funcione correctamente, es necesario establecer un orden en las cámaras de tal forma que la imagen en la primera cámara se pinte antes que en la segunda. Este paso es necesario porque por defecto cuando se crea una cámara esta pasa a ser la primera donde se pinta la imagen y, en este caso, no se desea ver el resultado de la segunda cámara.
- Para pintar la textura vacía a la resolución que se desea, se utiliza la segunda cámara. Esta cámara actúa como un buffer donde se pinta la textura de forma que no sea visible para el jugador.
- El siguiente paso es asignar la textura creada a la primera cámara. Esta asignación se realiza antes de determinar que objetos son visibles a la

[11] https://en.wikipedia.org/wiki/Shader

cámara mediante *culling*[12] para que no afecte al resultado final. Para escalar la imagen a la resolución del *viewport* se duplica la textura utilizando el *shader* de escalado. Al terminar el *frame*, se elimina la textura de la primera cámara y se vuelve a repetir el proceso desde el principio.

4.4.2. Resolución dinámica

Aprovechando que la técnica del *supersampling* está implementada de tal manera que, además de aumentar la resolución, permite utilizar resoluciones inferiores (*Subsampling*), se ha implementado la técnica de la resolución dinámica.

En esta técnica no se busca una alta calidad de imagen, sino que intenta mantener un *framerate* o imágenes por segundo estable durante el trascurso de la partida.

Para mantener el *framerate* siempre estable (a 60 imágenes por segundo) se reduce la resolución en tiempo real cuando la máquina donde está ejecutándose el videojuego no puede mantener un *framerate* constante y se vuelve a aumentar la resolución cuando el *framerate* se estabiliza.

Para seleccionar la resolución en tiempo real se ha diseñado un sistema de niveles donde, en función del coste en tiempo que se tarda pintar en pantalla varios *frames*, se reduce o aumenta el nivel de resolución.

El proceso de selección de nivel en tiempo real sigue los siguientes pasos:

1. Antes de iniciar se construye un sistema de niveles como una lista de valores intermedios entre la resolución más baja que se permite y la resolución más alta. En este caso la resolución más baja permitida es un 70% de la resolución original y la más alta un 200% de la resolución original.
2. A continuación, y ya en tiempo real, se va almacenado en un buffer el coste en tiempo que se tarda pintar en pantalla el *frame* anterior al actual. Para implementar este buffer se ha utilizado una estructura circular de tamaño fijo.
3. Como el sistema se basa en el coste en tiempo de los *frames*, es necesario establecer unos umbrales que permitan establecer cuando un *frame* se considera que tiene un coste alto y cuando un coste bajo. En este caso se

[12] Back-face culling: https://en.wikipedia.org/wiki/Back-face_culling

ha definido que un *frame* es caro cuando tarda un 20% más que el tiempo máximo y barato cuando tarda un 20% menos que el tiempo máximo de *frame*.

4. A partir de estos umbrales y del buffer de costes de los *frames*, se procede a elegir un nivel nuevo en el sistema de niveles. Hay cuatro casos:
    - El ultimo *frame* medido supera el umbral máximo de coste. En este caso se reduce la resolución en dos niveles.
    - Los últimos tres *frames* superan el umbral máximo de coste. En este caso también se reduce la resolución en dos niveles.
    - Predecir el coste del siguiente *frame* usando interpolación lineal. En este caso se intenta predecir cuanto tiempo va a costa el siguiente *frame* utilizando una interpolación lineal entre los dos últimos *frames* medidos o entre el último y el antepenúltimo *frame* si están disponibles. Si el resultado de esta interpolación supera el umbral máximo, se reduce la resolución en un nivel.
    - Los últimos tres *frames* no superan el umbral máximo de coste. En este caso se aumenta el nivel de resolución en un nivel.
5. Por último, es necesario aplicar el cambio de resolución dependiendo del nivel resultante. Para los cambios de resolución se utiliza el *supersampling* como una caja negra donde se introduce la resolución deseada y devuelve una imagen a dicha resolución.

5 Resultados

5.1 Versión final del juego

A continuación, se muestran una serie de capturas de la versión final del videojuego (ver figuras 14, 15 y 16) explicando en cada una de ellas a que parte corresponden.

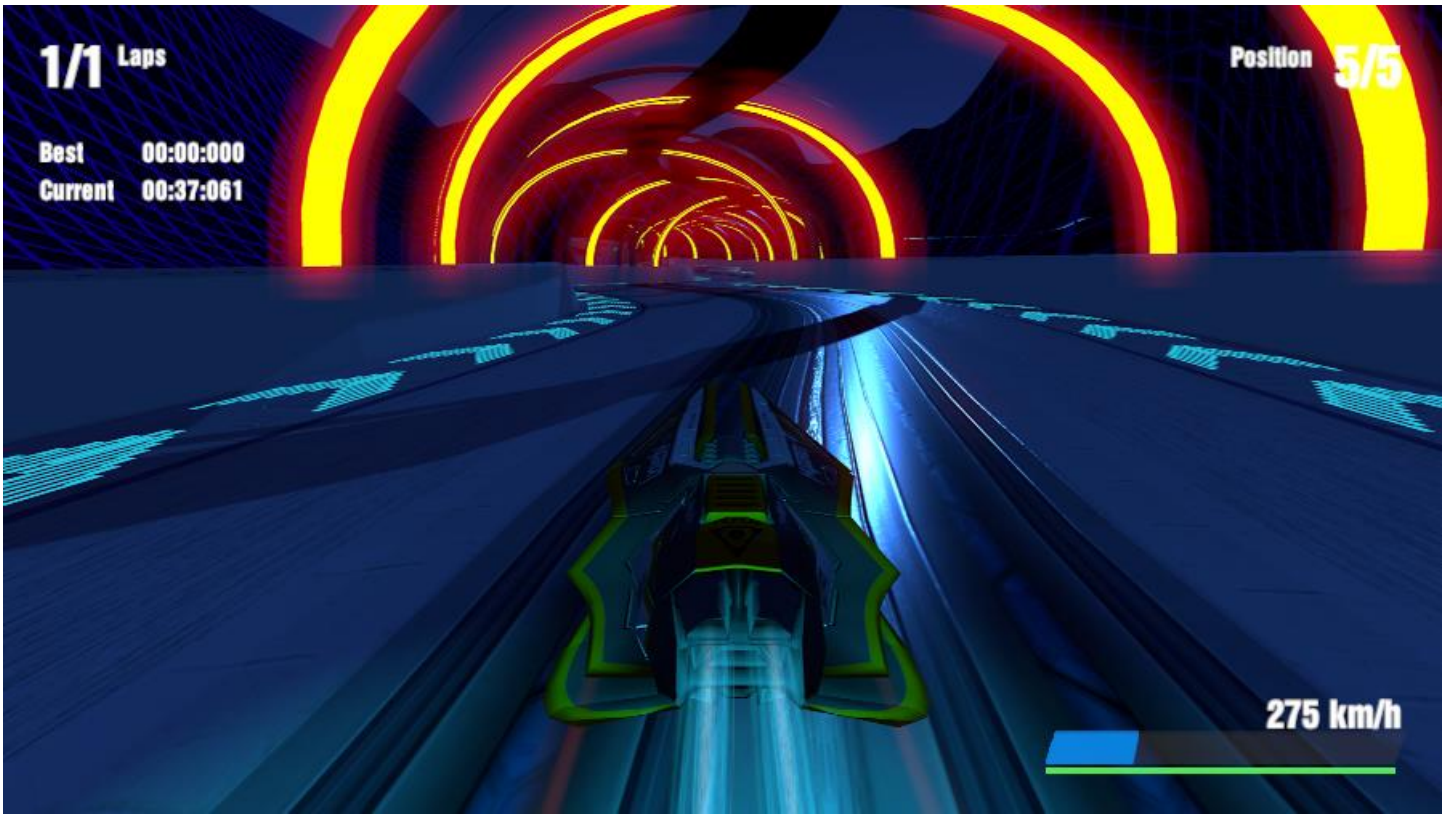

Figura 14. Pista Blood Dragon con nave Hyper

Figura 15. Pista Test con nave TeshShip

Figura 16. Pista Volcano con nave Flyer

5.1.1. Supersampling

Para comprobar los resultados del *supersampling*, se presentan dos capturas (ver figura 17) tomadas en el mismo lugar, la primera sin *antialiasing* y la segunda con *supersampling* x4.

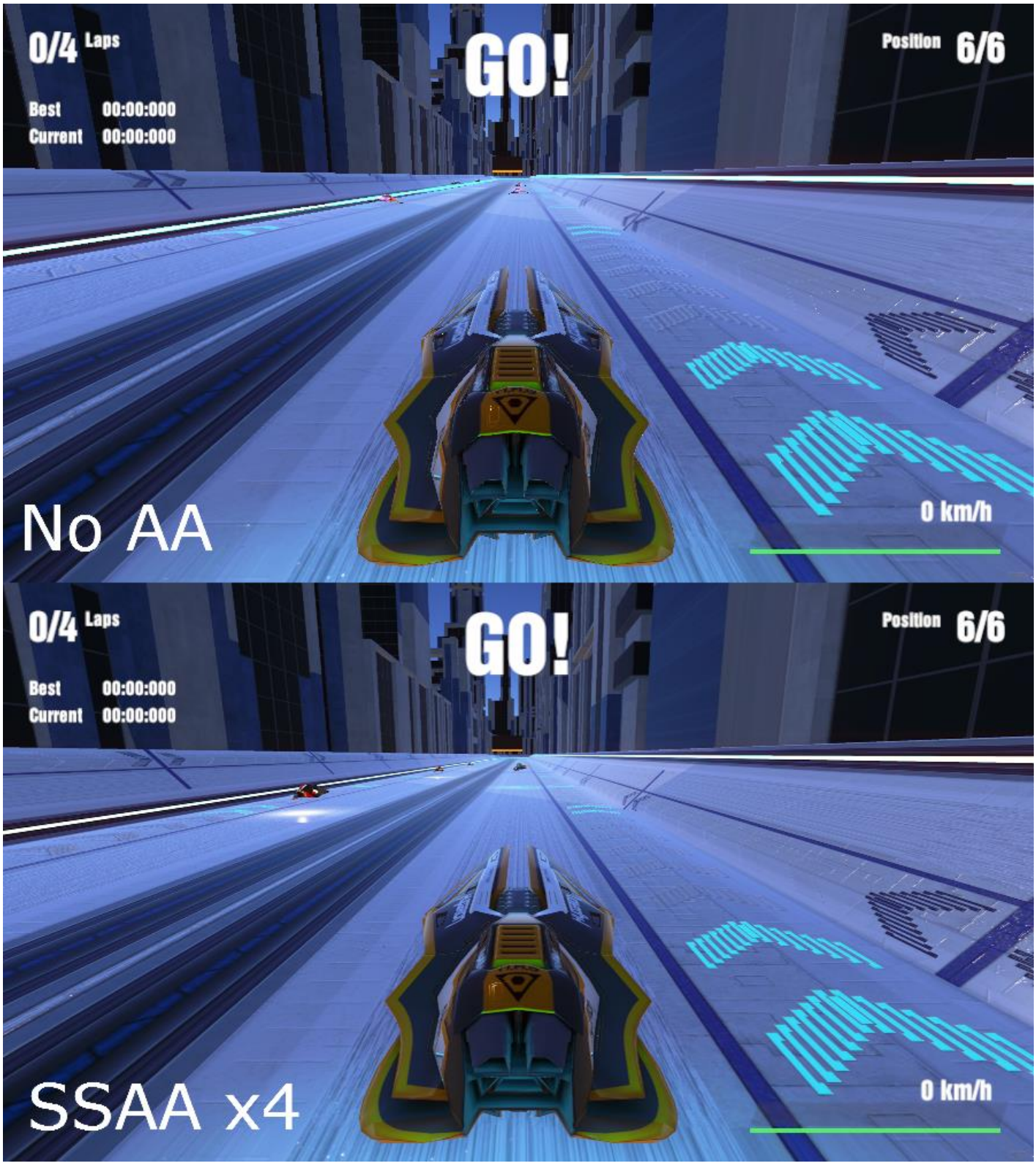

Figura 17. Comparativa entre No AA y SSAA x4 (*supersampling*)

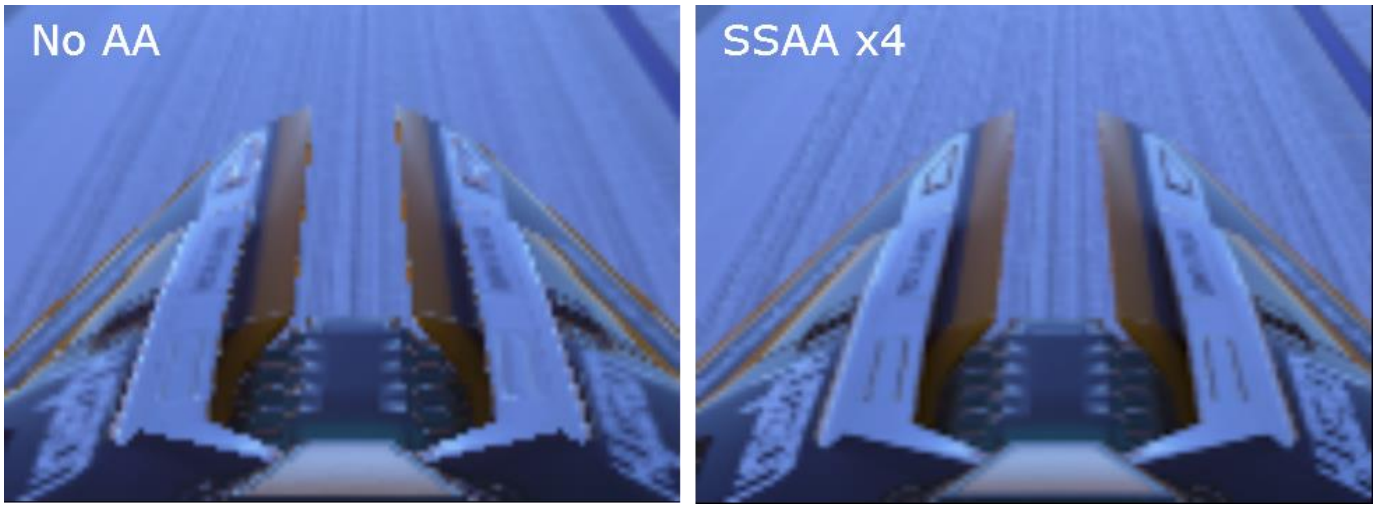

Figura 18. Misma comparativa que la figura anterior, pero con zoom

Se puede observar claramente (ver figuras 18) como el *supersampling* reduce de forma significativa los artefactos y mejora la calidad de la imagen.

5.1.2. Resolución dinámica

Para comprobar el rendimiento de la resolución dinámica se ha realizado la misma prueba en distintos ordenadores (ver figura 19). En esta prueba en concreto se han empleado tres ordenadores en el mismo circuito, con la misma nave y a la misma resolución (640x480).

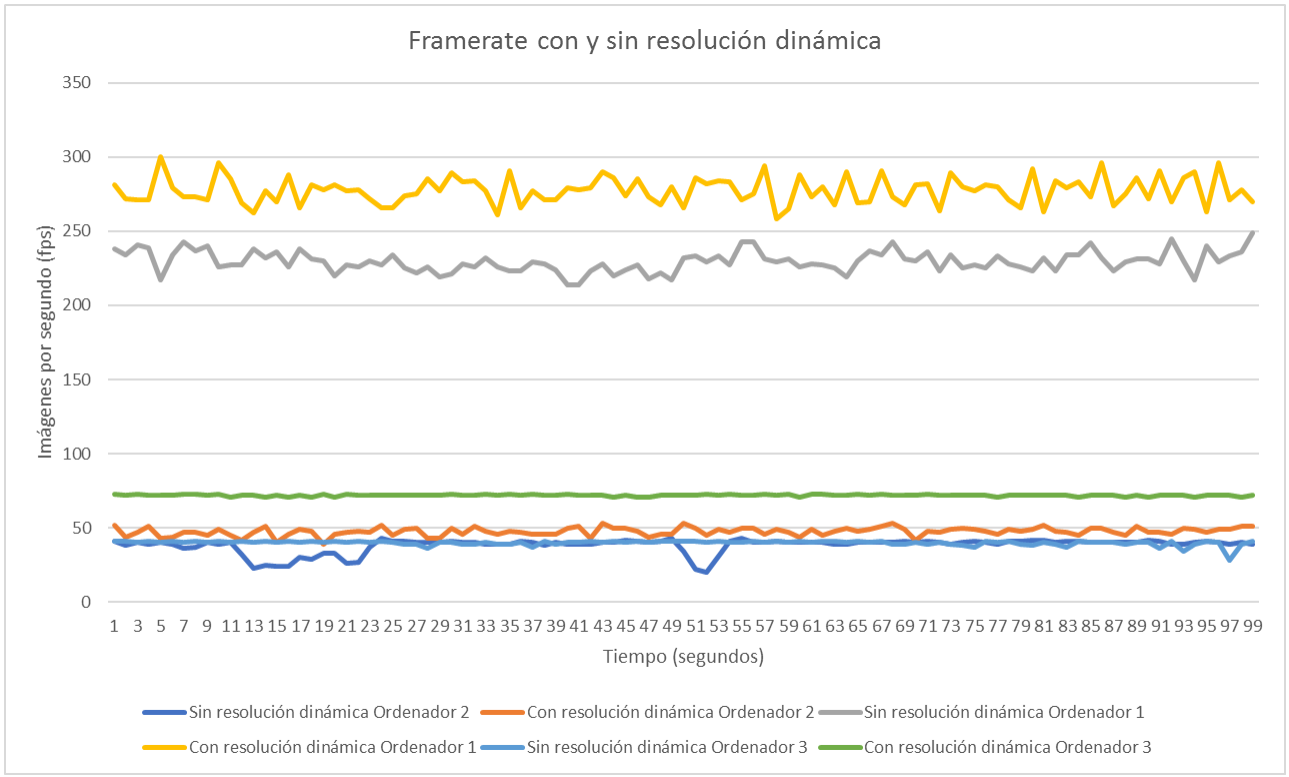

Figura 19. Comparativa de *framerate* entre una partida sin resolución dinámica y otra con resolución dinámica.

A continuación, se detalla los componentes de cada uno de estos ordenadores:

Ordenador 1:

- Procesador: Intel Core i5-4670K a 4.0 GHz
- Memoria RAM: 8GB
- Tarjeta gráfica: Nvidia GeForce GTX 770
- Sistema operativo: Windows 10 de 64 bits

Ordenador 2:

- Procesador: Intel Core i5-4200U a 2.30 GHz
- Memoria RAM: 4GB
- Tarjeta gráfica: Intel HD 4000
- Sistema operativo: Windows 10 de 64 bits

Ordenador 3:

- Procesador: Intel Core 2 Quad a 2.40 GHz
- Memoria RAM: 3GB
- Tarjeta gráfica: ATi Radeon HD 5550
- Sistema operativo: Windows 7 de 64 bits

En todas las máquinas se ha podido observar que el número de imágenes por segundo aumenta de forma notable cuando se utiliza la técnica de la resolución dinámica.

5.2 Evaluación de los usuarios

Tras acabar la implementación se realizaron algunas pruebas con usuarios para comprobar la jugabilidad del videojuego desarrollado. Se han realizado pruebas con un total de cinco usuarios con distinto conocimiento sobre videojuegos (ver figura 20) que evaluaron cuatro aspectos: la navegación en los menús, el control de las naves, la jugabilidad y la inteligencia artificial.

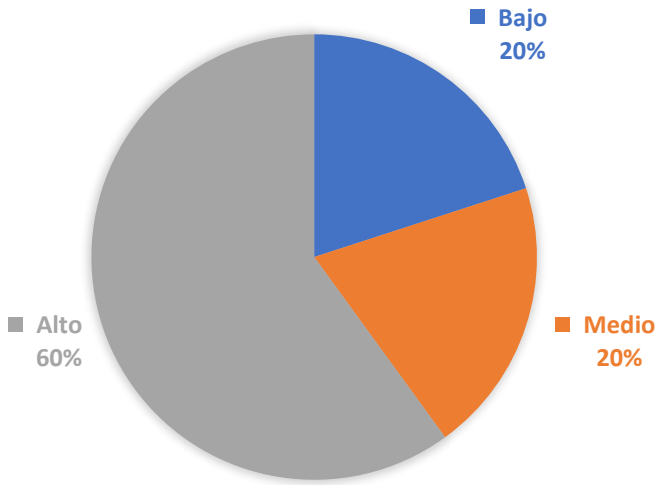

Figura 20. Conocimiento de los usuarios

5.2.1. Navegación en los menús

En primer lugar, se hicieron pruebas para comprobar la navegación entre los menús (ver figura 21). Casi todos los usuarios entendieron a la primera la navegación entre los menús, como empezar una partida y como cambiar las opciones del videojuego. Por otra parte, algunos usuarios tuvieron dificultades a la hora de entender cómo funcionaba las pantallas en el modo multijugador.

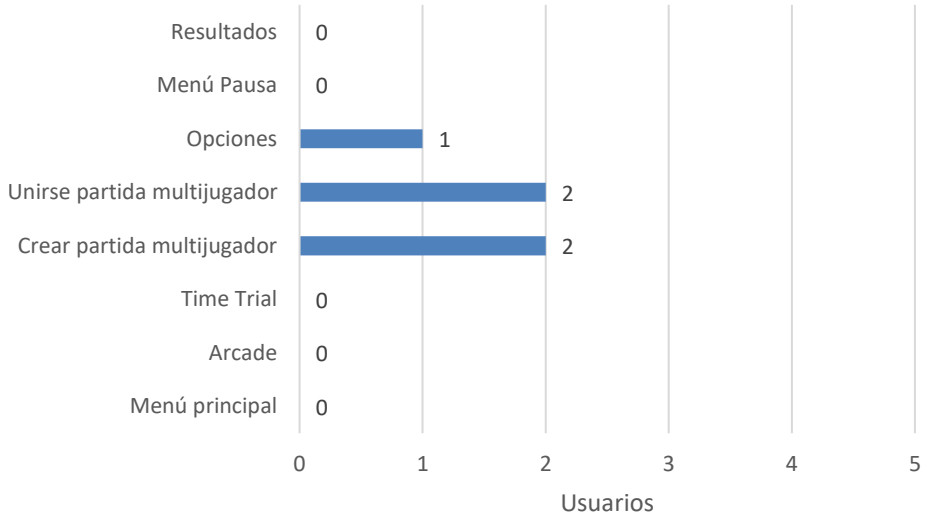

Figura 21. Problemas encontrados por los usuarios en la navegación por los menús

5.2.2. Control del juego

A continuación, se probó el nivel de adecuación de los controles (ver figura 22). La mayoría de los usuarios encontraron la disposición de los controles correcta, sin embargo, algunos de los usuarios tuvieron problemas a la hora de advertir como se controlaba la nave. Para solucionar este problema se plantea como una

posible mejora la inclusión de un tutorial al comienzo de la partida que explique los controles.

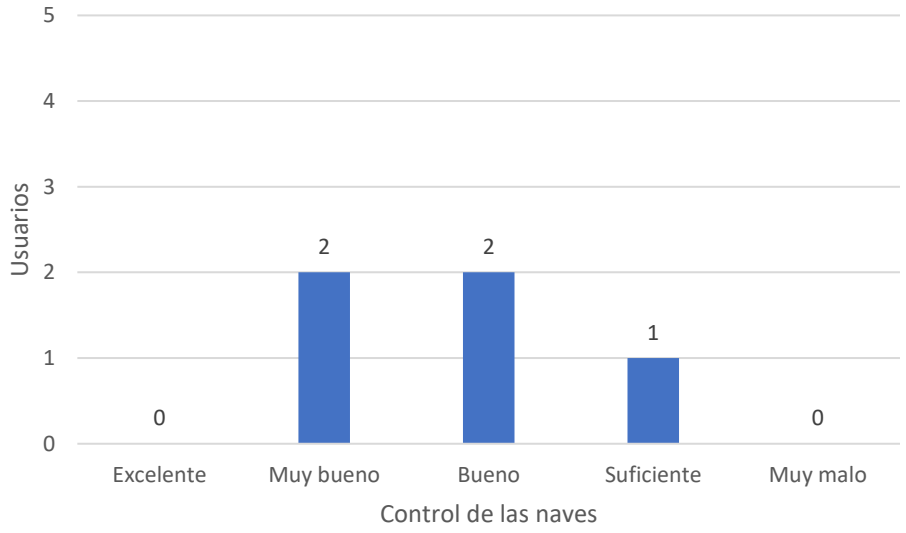

Figura 22. Nivel de adecuación de los controles

5.2.3. Jugabilidad

También se preguntó a los usuarios por el nivel de jugabilidad del videojuego (ver figura 23). Muchos usuarios encontraron dificultades a la hora de controlar la nave lo que producía cierta frustración. Para solucionar este problema se plantea como una posible mejora la realización de más pruebas enfocadas únicamente en mejorar como se controlan las naves.

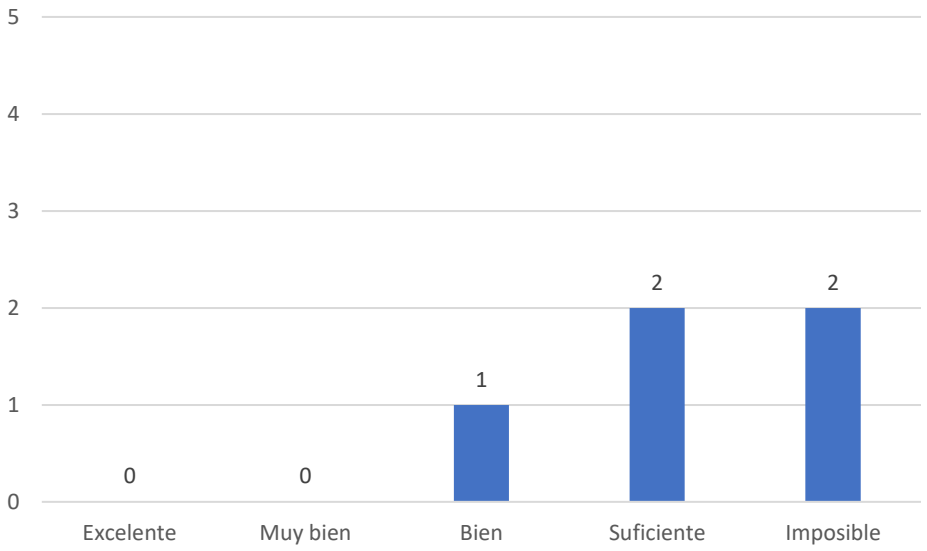

Figura 23. Nivel de jugabilidad

5.2.4. Inteligencia artificial

Otro de los aspectos sobre el que se preguntó a los usuarios fue sobre su experiencia con la inteligencia artificial (ver figura 24). En este caso, más de la mitad de los usuarios encontraron que la inteligencia artificial era demasiado rápida en la dificultad media y prácticamente imposible en la dificultad más alta. Por otro lado, el resto de usuarios afirmaron que el nivel de dificultad de la inteligencia artificial era correcto en los niveles medio y alto. Algún usuario también afirmo que la inteligencia artificial era muy agresiva en los adelantamientos.

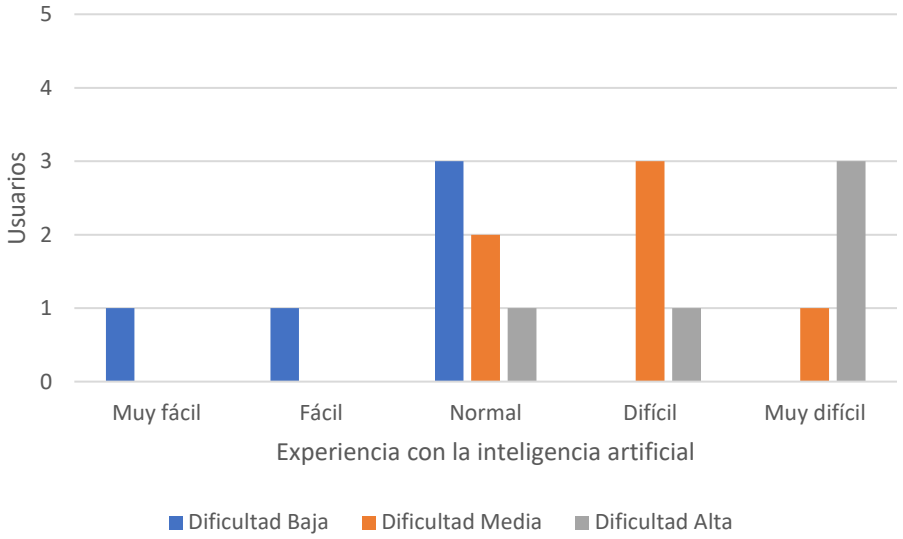

Figura 24. Experiencia con los distintos niveles de la inteligencia artificial

6 Conclusiones

En este proyecto se ha implementado desde cero un videojuego de carreras en 3D utilizando el proceso de desarrollo del software que se ha aprendido durante toda la carrera. Este proyecto ha supuesto una gran experiencia de aprendizaje en el desarrollo de un software que aúna tantas disciplinas como lo es un videojuego.

Se ha aprendido sobre el funcionamiento de las técnicas de inteligencia artificial en videojuegos de carreras que se desconocían por completo. Gracias a ello, se ha implementado con éxito una inteligencia artificial capaz ofrecer un reto al usuario.

Por otro lado, se ha obtenido una gran experiencia sobre la arquitectura de red necesaria para hacer funcionar un juego en línea y sobre las técnicas que se utilizan para mejorar la experiencia del usuario.

Además, se ha conseguido implementar dos técnicas gráficas que abordan tanto el aspecto de calidad de la imagen como el aspecto del rendimiento del videojuego. Destacar la técnica de la resolución dinámica ya que es una técnica muy novedosa que se ha empezado a utilizar en los últimos años en proyectos comerciales.

La realización de este proyecto no ha sido un camino de rosas y durante el desarrollo también han surgido varias dificultades.

La primera es el nivel de complejidad que conlleva realizar un videojuego una sola persona. Esta persona tiene que cubrir todos los campos que forman parte del proceso y, además, cualquier problema que surge tiene que ser arreglado por esta persona lo que a veces resulta una tarea muy dura.

Otra gran dificultad ha sido la gestión del tiempo que se ha invertido en cada tarea. En numerosas ocasiones se ha estimado un tiempo concreto para un requisito y ha habido que ampliar el tiempo de forma significativa, en algunas ocasiones el doble o el triple de tiempo.

Otro problema más concreto ha sido el desarrollo de la inteligencia artificial. Para dotar de cierta inteligencia a las naves se invirtió una gran cantidad de tiempo y, como queda constancia en el requisito no cumplido, surgieron dificultades a la hora de poder aprovechar las ideas que inicialmente estaban planteadas.

Otra dificultad que acompaño al proyecto desde sus inicios fue la creación de contenido propio y más en concreto todo el contenido que tiene que ver con la parte más artística del desarrollo de un videojuego.

En la siguiente dirección se puede descargar la última versión del videojuego: https://github.com/Arafo/Syncopia/releases

La estimación inicial de tiempo para acabar este trabajo fue cuatro meses, y finalmente ha costado doce meses (ver tabla 3). En total, la realización de este trabajo ha llevado un total de 380 horas. A continuación, se presenta el cronograma con el desarrollo de las tareas realizadas a lo largo del tiempo.

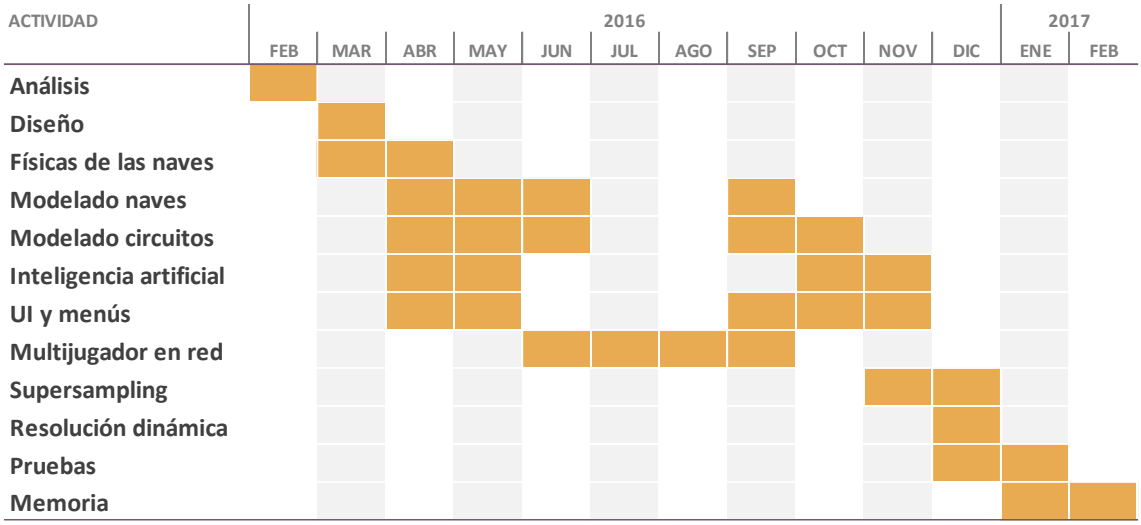

Tabla 3. Cronograma

6.1 Posibles mejoras

Un proyecto de este tipo es muy susceptible de poder mejorarse con nuevas características. A continuación, se exponen algunas de las posibles mejoras que se podrían realizar:

- Mayor número de circuitos y naves. El contenido del videojuego puede ser algo limitado sobre todo si se piensa en lanzar el videojuego comercialmente en alguna plataforma.
- Incluir un modo de juego tipo trayectoria o campeonato donde se enlacen todos los circuitos disponibles.
- Incluir una búsqueda automática de partidas multijugador como se hace en los videojuegos actuales.
- Incluir un sistema para guardar de forma persistente los mejores tiempos que se consiguen en el modo contrarreloj.
- Incluir los iconos del controlador o *glyph*[13] en los menús para facilitar la navegación.

[13] https://en.wikipedia.org/wiki/Glyph

- Mejoras en la interfaz y mayor variedad en la música que suena en la partida.
- Mejoras en los controles de las naves. Como se ha observado en las pruebas con usuarios, uno de los mayores problemas encontrados ha sido la dificultad de controlar las naves de forma precisa.
- Incluir un tutorial la primera vez que se juega para guiar al usuario explicando los controles y la interfaz.

6.2 Conclusión personal

Desarrollar un videojuego desde cero es una tarea compleja y que consume mucho tiempo si la lleva a cabo una sola persona. Durante este desarrollo ha habido momentos con cierto encanto, pero también mucha frustración.

Este desarrollo me ha permitido profundizar mucho en el trabajo que requiere gestionar un proyecto de software con tantas y tan diversas ramificaciones. Me ha permitido profundizar en el campo de la inteligencia artificial enfocada a videojuegos de conducción. También he aprendido mucho del funcionamiento de la estructura de red que requiere un videojuego para poder funcionar en línea.

También me ha enseñado a valorar más todo el trabajo artístico y creativo que hay detrás de un videojuego. Un videojuego entra por los ojos y si no gusta no importa lo bien desarrollado que este.

Por último, también me ha enseñado a apreciar la buena documentación y la gran comunidad de desarrolladores que ponen a disposición de todo el mundo sus conocimientos.

7 Referencias

Inteligencia artificial

1. Physics-based Racing AI Controllers and Track Analysis in SUPERBIKE CHAMPIONSHIP, *Paolo Maninetti*, http://aigamedev.com/ultimate/presentation/physics-racing-superbike/ Se consultó el 26 ene. 2017.
2. Video Games and Artificial Intelligence, *Microsoft Research*, https://www.microsoft.com/en-us/research/project/video-games-and-artificial-intelligence/ Se consultó el 26 ene. 2017.
3. How better AI can make racing games more fun, *Julian Togelius*, http://togelius.blogspot.com/2007/08/how-better-ai-can-make-racing-games.html Se consultó el 26 ene. 2017.

Multijugador

1. Network System Concepts, *Unity Documentation*, https://docs.unity3d.com/Manual/UNetConcepts.html Se consultó el 25 ene. 2017.
2. Fast-Paced Multiplayer, *Gabriel Gambetta*, http://www.gabrielgambetta.com/fast_paced_multiplayer.html Se consultó el 25 ene. 2017.
3. UNET HLAPI cheatsheet, *Unity Asset Store comparisons, guides and tips*, http://www.invisiblerock.com/unet-hlapi-cheatsheet/ Se consultó el 25 ene. 2017.
4. UNET Multiplayer: Interpolation and latency compensation, *Unity forums*, https://forum.unity3d.com/threads/unet-multiplayer-interpolation-and-latency-compensation.398102/ Se consultó el 25 ene. 2017.
5. TANKS! Networking Demo, *Unity Technologies*, https://www.assetstore.unity3d.com/en/#!/content/46213 Se consultó el 26 ene. 2017.
6. Unity Networking Sample Project, *SUPER88*, http://molx.us/2016/03/28/1/ Se consultó el 25 ene. 2017.

Resolución dinámica

1. Dynamic Resolution Rendering Article, *Intel Developer Zone*, https://software.intel.com/en-us/articles/dynamic-resolution-rendering-article Se consultó el 25 ene. 2017.
2. Any performance advantages to making a dynamic resolution scaler using Render Texture?, *Reddit*, https://www.reddit.com/r/Unity3D/comments/4m9zk9/any_performance_advantages_to_making_a_dynamic/ Se consultó el 25 ene. 2017.
3. The Lab Renderer, *Valve Corporation*, https://www.assetstore.unity3d.com/en/#!/content/63141 Se consultó el 26 ene. 2017.

Música

1. C418 - Cat (Caution & Crisis Remix), *Harvey Thompson*, https://soundcloud.com/harveydecember/c418-cat-caution-crisis-remix Se consultó el 29 ene. 2017.
2. Reaktion, *Canton*, https://soundcloud.com/canton/reaktion Se consultó el 29 ene. 2017.
3. 303 vs 909 (remix), *Canton*, https://soundcloud.com/canton/303-vs-909-remix Se consultó el 29 ene. 2017.
4. Tounge, *Itoshii*, http://sampleswap.org/artist/Katzenjammer Se consultó el 30 ene. 2017.
5. Virtual Memories, *RBnG*, https://soundcloud.com/rbng-1/virtual-memories Se consultó el 30 ene. 2017.

General

6. Performance tips for Unity 2d mobile, *Damian Connolly*, https://divillysausages.com/2016/01/21/performance-tips-for-unity-2d-mobile/ Se consultó el 26 ene. 2017.
7. VR Samples, *Unity Technologies*, https://www.assetstore.unity3d.com/en/#!/content/51519 Se consultó el 26 ene. 2017.
8. Reflections, Catlike Coding, http://catlikecoding.com/unity/tutorials/rendering/part-8/ Se consultó el 26 ene. 2017.

9. Rewired Documentation, *Rewired*, http://guavaman.com/projects/rewired/docs/Documentation.html Se consultó el 26 ene. 2017.
10. KinoBloom, KinoVignette y KvantWall, *Keijiro Takahashi*, https://github.com/keijiro Se consultó el 26 ene. 2017.
11. BallisticNG, *GitHub*, https://github.com/bigsnake09/BallisticNG Se consultó el 26 ene. 2017.
12. Volumetric Lights for Unity 5, *Michal Skalsky*, https://github.com/SlightlyMad/VolumetricLights Se consultó el 26 ene. 2017.
13. Temporal Anti-Aliasing, *Unity forums*, https://forum.unity3d.com/threads/temporal-anti-aliasing.412482/ Se consultó el 26 ene. 2017.
14. 4copter: Conceptos (I) - Ejes giros y fuerzas, *Ranganok Schahzaman*, http://skiras.blogspot.com/2012/10/4copter-conceptos-i-ejes-giros-y-fuerzas.html Se consultó el 26 ene. 2017.
15. Hyperion, *MEG.A.BYTE*, http://megabyte-art.tumblr.com/post/93401754842 Se consultó el 29 ene. 2017.
16. Low-poly Futuristic Towers, *SolCommand*, http://www.solcommand.com/2013/03/futuristic-towers.html Se consultó el 30 ene. 2017.

8 Anexos

8.1 Game Design Document

En este anexo se expone el documento correspondiente al diseño del videojuego o *Game Design Document* (GDD). En él se describe, en un lenguaje de alto nivel, todas las decisiones de diseño de forma que sirva de guía para el desarrollo tanto a nivel técnico como artístico.

8.1.1. Introducción

Syncopia es un juego de carreras futurista basado en el fenómeno de la antigravedad. Está inspirado en juegos como Wipeout o F-Zero (ver figura 25).

Figura 25. *Wipeout* a la izquierda y *F-Zero* a la derecha

<u>Resumen</u>

En un futuro distante, la tecnología ha avanzado a unos niveles inimaginables. Los antiguos coches de carreras propulsados por motores de combustión son cosa del pasado. Los nuevos bólidos están propulsados por una combinación de hidrógeno y electricidad que permite alcanzar velocidades de hasta 700 km/h. Además, en estas nuevas naves se ha eliminado el elemento menos aerodinámico, las ruedas, y se han sustituido por unos propulsores antigravitatorios que permiten que las naves "floten".

A raíz de esta evolución ha surgido una nueva competición, Syncopia. Una competición donde las naves más rápidas del planeta compiten entre ellas por ser las mejores

A pesar de los avances, solo tres empresas del sector privado han conseguido crear con éxito un bólido capaz de hacer frente a las exigencias de la competición.

Quien conseguirá ganar y alzarse con la corona de campeón en la competición más extrema jamás vista.

Características principales

- Sistema único de inteligencia artificial
- Completamente en 3D
- Compite con 4 personas en el modo online
- Compite con hasta 5 rivales en el modo arcade
- Compite contra el cronómetro en el modo contrarreloj
- Estilo artístico retro futurista

Plataforma de desarrollo

El juego se va a desarrollar para el sistema operativo Windows sin descartar versiones para OS X y Linux. Estas son las características del sistema en el que se va a desarrollar:

- SO: Windows 10
- Procesador: Intel Core i5-4670K 4.0Ghz
- Memoria: 8 GB de RAM
- Gráficos: NVIDIA GTX 770
- DirectX: Versión 11

8.1.2. Gameplay

Descripción

Syncopia es un juego de carreras para uno o varios jugadores. Su modo de juego principal es el modo *Arcade*. A los jugadores se les presenta el reto de intentar ser los más rápidos en cada carrera mientras compiten contra unos rivales que no le pondrán las cosas fáciles. El objetivo de este modo de juego es finalizar la carrera en la mejor posición posible.

Cuando se empieza una nueva carrera en el modo *Arcade*, el jugador puede elegir el número de rivales (hasta 5), el número de vueltas (hasta 6) y la dificultad que puede ser baja, media o alta. El grado de dificultad no solo afecta a los rivales, sino que también aumenta o disminuye de forma general la velocidad máxima de todas las naves.

También se cuenta con un modo contrarreloj. En este modo de juego solo toma parte un jugador y consiste en intentar batir una y otra vez los mejores tiempos de vuelta. Cuando se empieza una nueva carrera en el modo contrarreloj, el jugador puede escoger el número de vueltas que desea dar.

Por último, se cuenta con el modo multijugador en el que los jugadores pueden competir con hasta tres jugadores a través de internet. El objetivo principal de este modo de juego es acabar la carrera en la primera posición. Cuando se empieza una partida en el modo multijugador, el jugador puede ser el anfitrión si ha creado la partida o un cliente si se ha unido a la partida. En el caso del cliente, el jugador puede escoger el tipo de la nave y su color. El anfitrión puede escoger lo mismo y, además, puede elegir el circuito donde se va a disputar la carrera.

Game Flow

En el menú principal, los jugadores tienen las siguientes opciones:

- Quick Race
- Arcade
- Time Trial
- Multiplayer
- Options
- Exit

"Quick Race" inicia una carrera en un circuito y con una nave aleatorios. El número de rivales, la dificultad y el número de vueltas es siempre el mismo.

"Arcade" permite a los jugadores iniciar una carrera con las opciones que desean. A los jugadores se les presentan varias pantallas donde, además del número de rivales, la dificultad y el número de vueltas, pueden elegir el circuito donde quieren competir y la nave con la que quieren correr. Al terminar la carrera, los jugadores pueden visualizar una pantalla con los resultados y reiniciar la carrera o volver al menú principal.

"Time Trial" inicia un modo donde los jugadores compiten por batir su propio tiempo. A los jugadores se les presentan varias pantallas donde pueden elegir el circuito donde quieren competir, la nave con la que quieren correr, la dificultad y el número de vueltas. Al terminar este modo, los jugadores pueden visualizar sus tiempos en cada vuelta y reiniciar el modo o volver al menú principal.

"Multiplayer" permite a los jugadores competir entre sí a través de Internet. A los jugadores se les presenta una pantalla donde pueden elegir crear una nueva partida o unirse a una partida ya existente. Al terminar una carrera en este modo, los jugadores pueden visualizar los resultados y volver al menú multijugador.

"Options" permite al jugador cambiar los efectos gráficos, el volumen, los controles y el idioma.

"Exit" termina el juego.

Controles

Por defecto, las flechas izquierda y derecha del teclado se usan para girar a izquierda y derecha. La barra espaciadora se utiliza para acelerar. Las teclas A y D se pueden utilizar para activar los frenos laterales. La tecla *Shift* se utiliza para activar el turbo.

Estos controles también están disponibles para la mayoría de mandos genéricos. Utilizando un mando de Xbox, estos son los controles por defecto: el joystick izquierdo se usa para controlar el giro. El botón A se utiliza para acelerar. Los gatillos LT y RT se utilizan para activar los frenos laterales. El botón X se utiliza para activar el turbo.

Acción	**Teclado**	**Mando**
Acelerar	Espacio	Botón A
Girar izquierda	Flecha izquierda	Joystick izquierdo
Girar derecha	Flecha derecha	Joystick izquierdo
Freno lateral izquierdo	A	LT
Freno lateral derecho	D	RT
Turbo	Shift	Botón X

Tabla 4. Controles por defecto

En cualquier momento durante la partida, si se presiona el botón escape del teclado o *Start* en el mando, se presentará un menú de pausa que permitirá al jugador cambiar las opciones del juego, volver al menú principal o volver a la partida.

Naves

Las naves en Syncopia son vehículos antigravitatorios, similares al diseño del juego de carreras Wipeout. Estos vehículos no son coches ni tampoco naves espaciales, sino que son una mezcla donde los vehículos todavía son terrestres, pero se asemejan al aspecto de una nave espacial.

Hyper*:* Diseñada para los jugadores que desean tener un control total sobre la nave. Es la más moderna y presenta el mejor diseño aerodinámico de todas las naves antigravitatorias construidas hasta el momento. Por el contrario, su complejo diseño hace que tenga la peor velocidad máxima

Flyer: El equilibrio perfecto entre movimiento y velocidad. Su diseño estaba basado en una nave espacial convencional lo que limita su movimiento, pero permite una velocidad bastante decente.

TestShip: Diseñada para los jugadores que buscan la nave con la velocidad máxima más alta. Todavía en fase experimental, es la nave más ligera de todas, pero en su contra, el control no es demasiado preciso.

Las características de cada nave se muestran en la siguiente tabla:

	Hyper	**Flyer**	**TestShip**
Velocidad máxima	500km/h	507km/h	516km/h
Agarre	Medio	Alto	Medio
Cantidad de giro	Alto	Medio	Bajo

Tabla 5. Características de las naves

Cada nave tiene que tener definidas unas características que la diferencien lo suficiente de otra nave para no tener la misma sensación de control.

Circuitos

El campeonato Syncopia incluye tres circuitos, cada uno con unas características diferentes.

Blood Dragon: Circuito futurista situado a una gran altura de la superficie terrestre. Sus curvas en forma de S recuerdan la forma de un dragón. Las carreras en este circuito se realizan de noche.

Volcano: Circuito situado en un ambiente desértico con temperaturas muy elevadas. Este circuito destaca por sus cambios de nivel.

Test: Circuito tipo ovalo para realizar pruebas.

Powerups

Los jugadores y los rivales pueden coger *powerups* por todo el circuito para mejorar sus habilidades. Los *powerups* se identifican por su superficie brillante.

El único *powerup* que se va a incluir es el turbo. Este *powerup* permite aumentar la velocidad de la nave durante unos instantes en el momento que se pasa por encima de él.

8.1.3. Interfaz

El diseño de la interfaz se puede separar en la interfaz del menú principal (ver figura 26) y la interfaz de usuario dentro del juego. La interfaz del menú principal será la primera que vea el usuario cuando el juego comienza después de que se muestre la pantalla de bienvenida. La pantalla de bienvenida mostrara el logo de Unity.

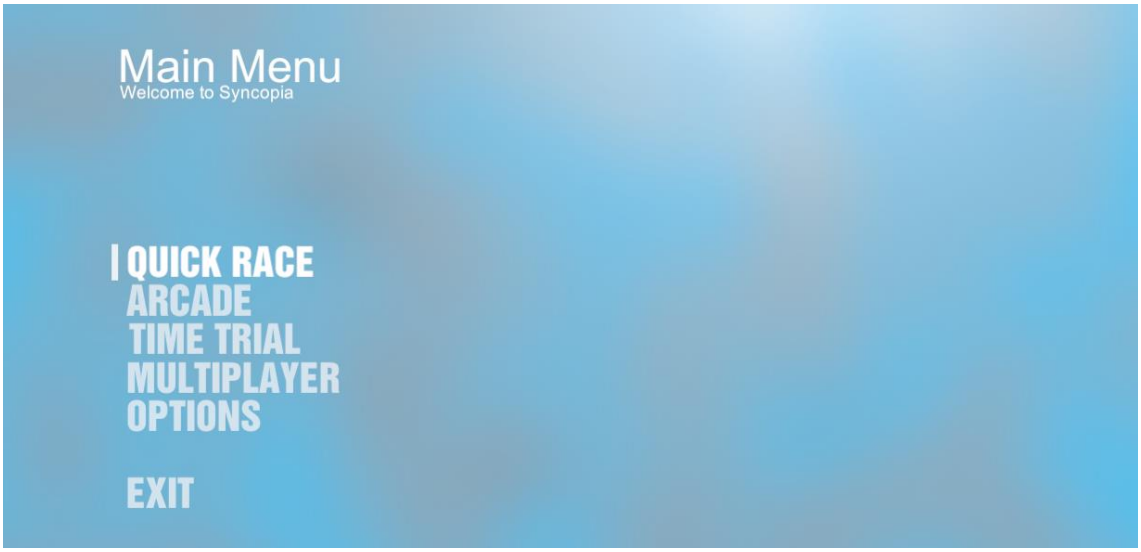

Figura 26. Menú principal

La interfaz de usuario estará compuesta de menús interactivos y el *heads up display* (HUD). Los menús interactivos se podrán controlar con el ratón, el teclado o un mando. Cada menú menos el principal incluirá un botón que permita ir al menú anterior.

Menú principal

El menú principal tendrá las siguientes opciones: Quick Race, Arcade, Time Trial, Multiplayer y Opciones. El siguiente diagrama de flujo (ver figura 27) muestra las interacciones del menú principal con otros menús de la interfaz.

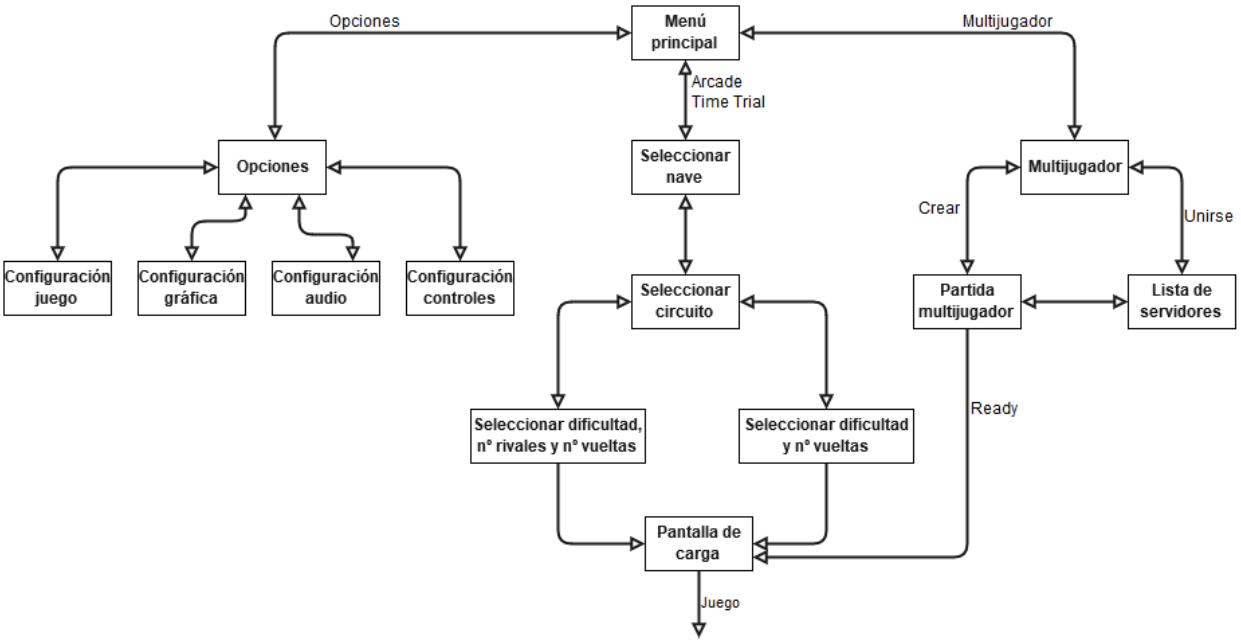

Figura 27. Diagrama flujo del menú principal

Menú de elección de nave

Este menú permitirá al jugador elegir la nave que quiere usar. Una vez el jugador a escogido una nave, se le muestra el menú de selección de circuito.

Menú de elección de circuito

Este menú permitirá al jugador elegir el circuito en el que quiere competir. Una vez el jugador a escogido un circuito, se le muestra el menú para seleccionar las opciones de la carrera.

Menú de opciones de carrera

Este menú permitirá elegir al jugador diferentes opciones de la carrera como el número de vueltas a dar, la dificultad de los rivales y el número de rivales. Una vez el jugador ha elegido las opciones, se le mostrara la pantalla de carga seleccionando "Ready".

Menú multijugador

Este menú permitirá elegir al jugador entre crear una partida multijugador o unirse a una partida multijugador ya existente.

Menú de partida multijugador

Este menú permitirá al jugador elegir diferentes opciones de la carrera multijugador dependiendo de si es el anfitrión o un cliente. En el caso de que el jugador sea el anfitrión, es decir, el que ha creado la partida, podrá elegir la nave,

el color de la nave y el circuito. En el caso de que el jugador sea un cliente solo podrá elegir la nave y su color.

Menú de lista de servidores

Este menú permitirá al jugador visualizar los servidores o partidas disponibles. El jugador podrá unirse a cualquiera de estas partidas. Cuando el jugador se una a una partida se le mostrara el menú de partida multijugador.

Menú de opciones

Este menú permitirá al jugador elegir el tipo de configuración que quiere cambiar. El jugador podrá cambiar las siguientes opciones:

- Opciones del juego (idioma)
- Opciones gráficas (resolución, *antialiasing*, pantalla completa, etc.)
- Opciones de sonido
- Opciones del control

Pantalla de carga

La pantalla de carga mostrara un mensaje de texto indicando la carga y una barra de progreso para indicar cuanto tiempo cuanto tiempo falta para que empiece la partida.

Heads Up Display (HUD)

El HUD dentro del juego mostrará la siguiente información:

- Posición del jugador (Position 1/4)
- La vuelta en la que el jugador se encuentra (*Lap* 2/3)
- El tiempo de la vuelta actual
- El tiempo de la mejor vuelta
- La velocidad en km/h del jugador
- Una barra de progreso para mostrar la velocidad de forma gráfica
- Una barra de progreso que indique la cantidad de turbo que queda en la nave

Menú de pausa

El menú de pausa dentro del juego tendrá las siguientes opciones:

- Continue: El jugador puede continuar la partida en el punto en el que la había pausado.
- Options: Muestra el menú de opciones en el mismo formato que el que se muestra en el menú principal.

- Exit: Vuelve al menú principal.

Pantalla de resultados

La pantalla de resultados mostrara diferentes clasificaciones dependiendo del modo de juego.

Si el jugador ha terminado una partida *arcade* o multijugador, se le mostrara la posición final, el nombre, la nave, el tiempo total y el mejor tiempo de vuelta de todos los jugadores.

Si el jugador ha terminado una partida contrarreloj, se le mostrara una lista con las vueltas realizadas. En cada vuelta se podrá ver el tiempo total y el tiempo de los parciales intermedios.

Independientemente del modo de juego, el jugador podrá elegir en esta pantalla las siguientes opciones:

- Restart: Reinicia la prueba que acaba de terminar el jugador.
- Exit: Vuelve al menú principal.

8.1.4. Tecnología

Plataforma destino

Estas son las características mínimas para las que se va a desarrollar:

- SO: Windows 7, 8, 8.1, 10
- Procesador: i3 2.6Ghz
- Memoria: 2 GB de RAM
- Gráficos: NVIDIA GT 340 / ATi Radeon HD 5550
- DirectX: Versión 11

Estas son las características recomendadas:

- SO: Windows 10
- Procesador: i5 2.6Ghz
- Memoria: 4 GB de RAM
- Gráficos: NVIDIA GTX 770 / AMD R9 280X
- DirectX: Versión 11

Objetivos técnicos

- IA adaptativa capaz de reaccionar y adaptarse a los cambios que se producen en el entorno
- Motor físico adaptado a las mecánicas del juego
- Modo multijugador
- Técnicas avanzadas de procesamiento de imágenes (*antialiasing*, oclusión ambiental, etc.).

APIs / SDKs

Las siguientes tecnologías se van a utilizar en el desarrollo del juego:

- Unity 5.4.0f3
- DirectX 11.0
- Visual Studio 2015
- Blender 2.75a
- Microsoft Word 2016
- GitHub
- Draw.io

Documentación y control de versiones

Para gestionar el control de versiones se utilizará la plataforma GitHub.

8.2 Manual de usuario

Para iniciar la aplicación solo hacer falta hacer doble clic sobre el fichero ejecutable del juego "syncopia.exe".

Requisitos mínimos:

- SO: Windows 7, 8, 8.1, 10
- Procesador: Intel Core i3 2.6Ghz
- Memoria: 2 GB de RAM
- Gráficos: NVIDIA GT 340 / ATi Radeon HD 5550
- DirectX: Versión 11

Requisitos recomendados:

- SO: Windows 10
- Procesador: Intel Core i5 2.6Ghz
- Memoria: 4 GB de RAM
- Gráficos: NVIDIA GTX 770 / AMD R9 280X
- DirectX: Versión 11

Menú principal

Una vez se ha iniciado el videojuego, esta es la primera pantalla que se muestra (ver figura 28):

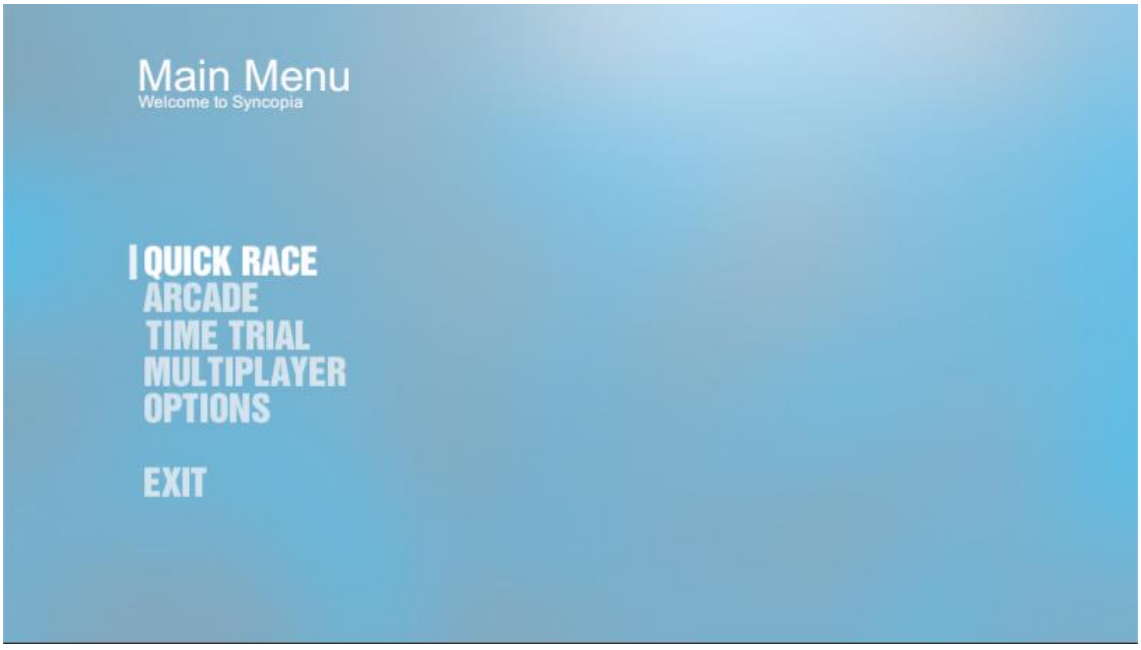

Figura 28. Menú principal

Desde la pantalla "Main Menu" se puede acceder a todas las posibilidades de Syncopia. La pantalla está compuesta por seis opciones dispuestas en vertical.

Quick Race
Disfruta de carreras rápidas y sencillas.

Arcade
Disfruta de carreras personalizadas.

Time Trial
Disfruta de una lucha contra el cronometro.

Multiplayer
Disfruta de las carreras online.

Options
Personaliza la experiencia a tu gusto.

Exit
Sal de juego.

Modo Arcade y Modo Time Trial

El Modo Arcade y el modo Time Trial ofrecen la experiencia más completa compitiendo en una carrera contra otros rivales o luchando contra el cronómetro. Una vez que se ha seleccionado uno de estos modos de juego, se presentan varias pantallas para poder configurar los siguientes ajustes (ver figuras 29, 30 y 31):

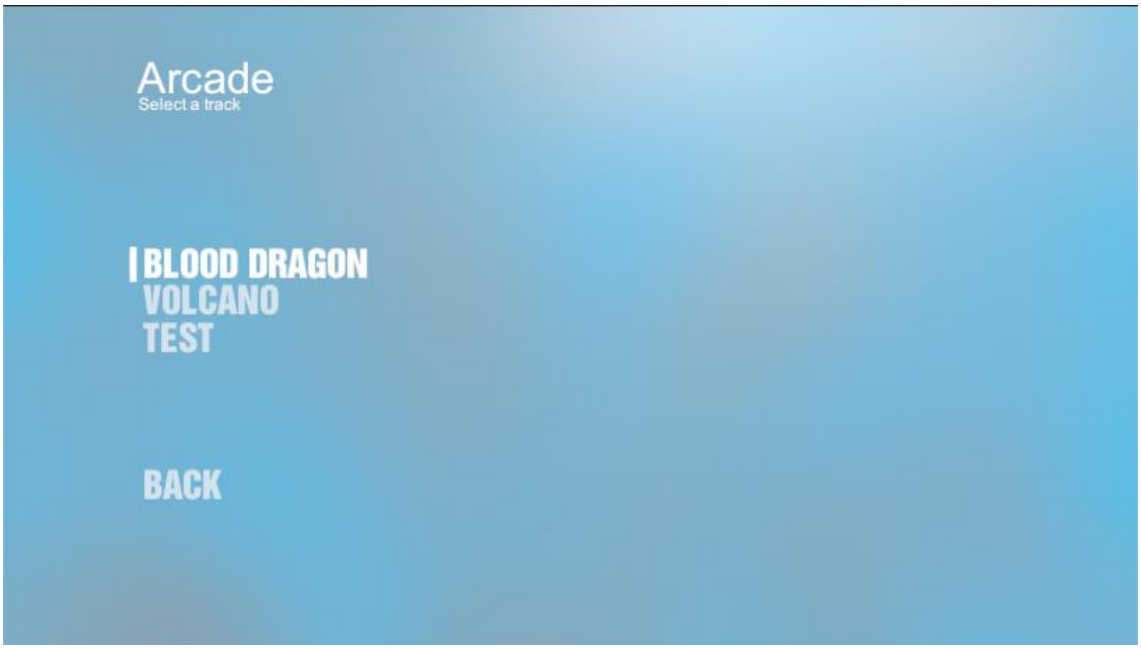

Figura 29. Menú para seleccionar una pista

1. Elige una pista
    a. Para elegir la pista donde se quiere competir: Blood Dragon, Volcano o Test.

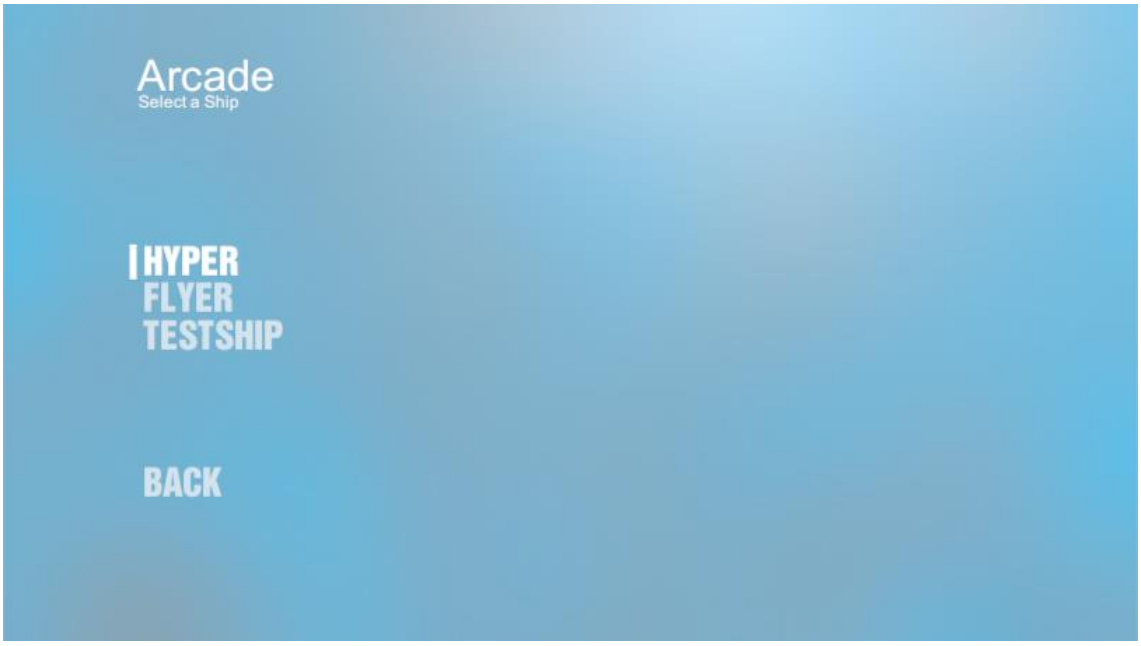

Figura 30. Menú para seleccionar una nave

2. Elige una nave
    a. Para elegir la nave con la que se quiere competir: Hyper, Flyer o TestShip.

Figura 31. Menú para configurar las opciones de carrera

3. Configura las opciones de carrera

a. Para configurar las opciones de carrera para los modos Arcade y Time Trial. En el modo Arcade se puede elegir el nivel de dificultad, el número de vueltas y el número de rivales, mientras que en Time Trial se puede configurar el número de vueltas y la dificultad.

Modo multijugador

El modo multijugador (ver figura 32) permite disputar carreras online con otras personas.

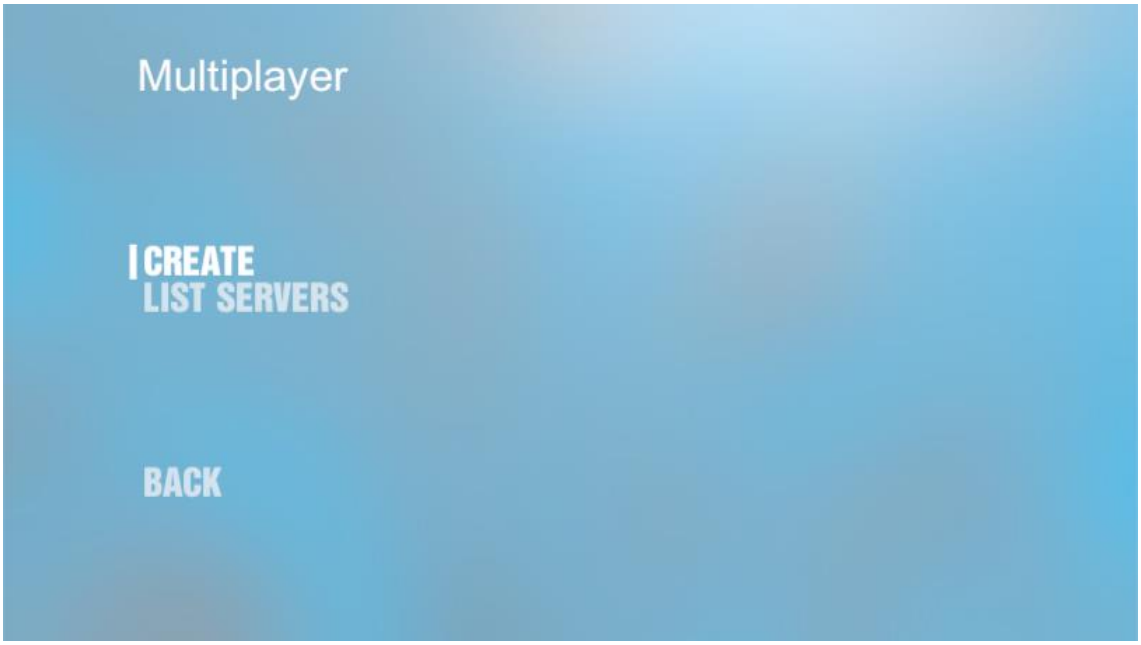

Figura 32. Modo multijugador

Crear una partida multijugador

Crear una partida multijugador (ver figura 33) permite ser el anfitrión de una partida pudiendo personalizar distintas opciones.

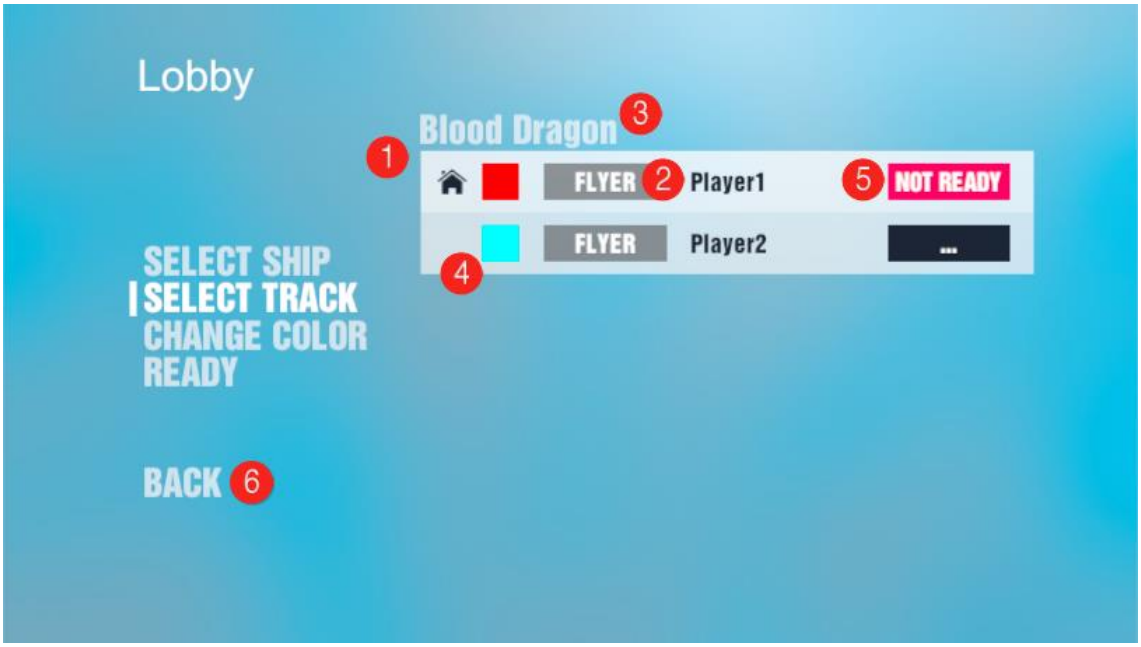

Figura 33. Lobby multijugador

A continuación, se muestra la información que aparece en el *lobby* multijugador.

1. Lista de jugadores en la partida
2. Nave seleccionada
3. Pista seleccionada
4. Color de la nave
5. Estado para iniciar la partida, preparado o no preparado
6. Volver al menú anterior

Unirse a una partida multijugador

Unirse a una partida (ver figura 34) permite unirse a una partida creada por otro jugador.

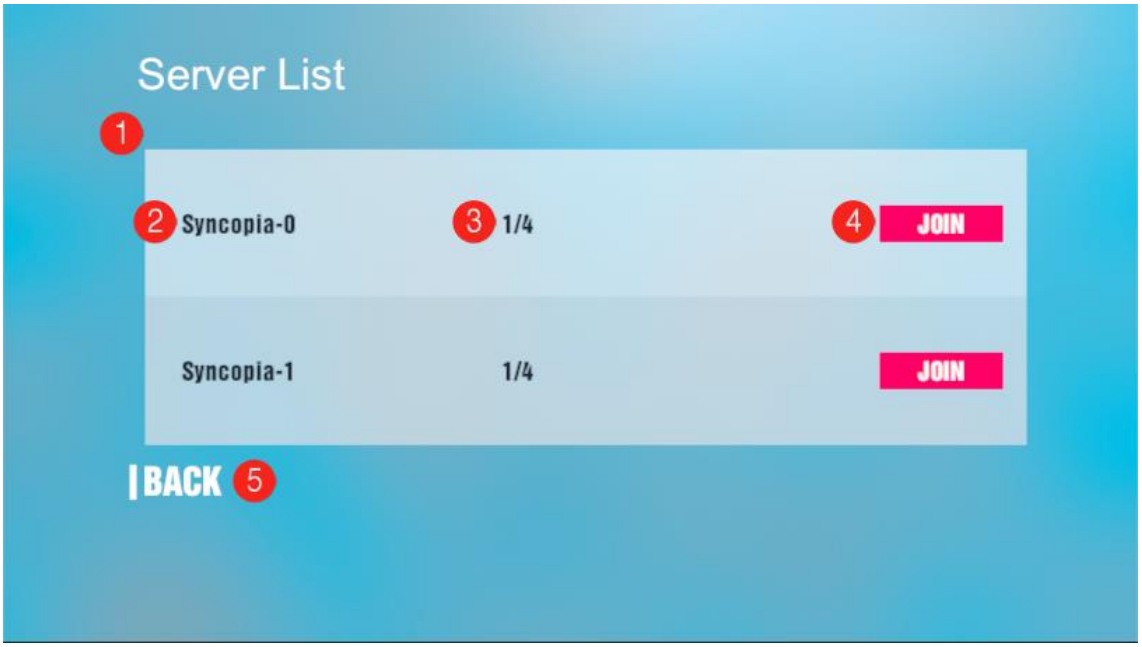

Figura 34. Lista de servidores

1. Lista de partidas
2. Nombre de la partida
3. Número de jugadores en la partida
4. Unirse a la partida
5. Volver al menú anterior

Opciones

El menú de opciones (ver figura 35) permite configurar los siguientes ajustes:

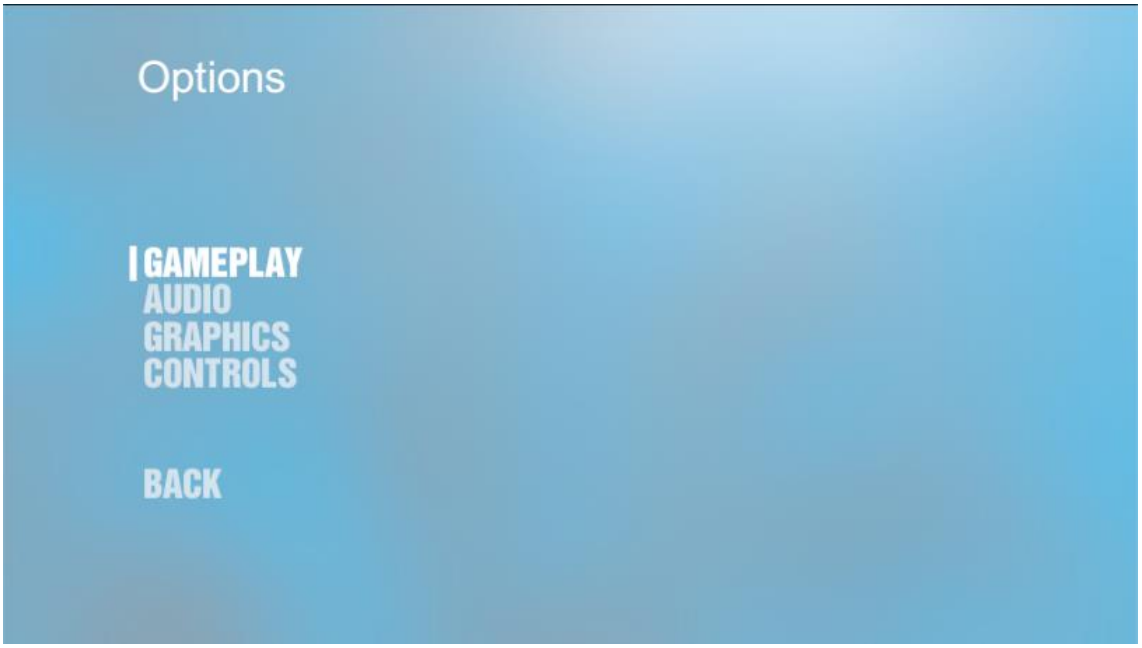

Figura 35. Menú de opciones

Gameplay
Establece el idioma del juego.

Audio
Establece el volumen de sonido de la música, los efectos de sonido, las voces o todas en general.

Graphics
Establece opciones graficas como la resolución de la pantalla, pantalla completa, el tipo de *antialiasing*, *bloom*, límite de *framerate*, distancia de dibujado y resolución dinámica.

Controls
Edita los controles por defecto.

HUD

A continuación, se muestra la información que aparece en pantalla durante las carreras (ver figura 36).

Figura 36. HUD de la nave

1. Vuelta actual/Vueltas totales
2. Tiempo total
3. Mejor vuelta de la carrera
4. Posición actual
5. Velocímetro
6. Turbo

Menú de pausa

El menú de pausa (ver figura 37) permite parar el juego en mitad de una partida. En este menú se puede reanudar la partida donde se había parado, reiniciar la partida desde el principio o cambiar las opciones del juego de forma similar al menú de opciones.

Figura 37. Menú de pausa

Pantalla de resultados

Una vez se haya terminado una carrera en cualquiera de los modos, se podrá visualizar una pantalla de resultados (ver figura 38). La información exacta variará en función del modo de juego.

Figura 38. Pantalla de resultados