Universitat Oberta de Catalunya

RaceTrace

Autor: Javier Valls Catalá

Tutor: Bernat Bas Pujols

Profesor: Joan Arnedo Moreno

Grado de Ingeniería Informática

Computación

14/01/2023

Esta obra está sujeta a una licencia de Reconocimiento- NoComercial-SinObraDerivada 3.0 España de CreativeCommons.

Copyright © 2024 Javier Valls Catalá

FICHA DEL TRABAJO FINAL

Título del trabajo:	*RaceTrace*
Nombre del autor:	*Javier Valls Catalá*
Nombre del colaborador/a docente:	*Bernat Bas Pujols*
Nombre del PRA:	Joan Arnedo Moreno
Fecha de entrega (mm/aaaa):	*01/2024*
Titulación o programa:	*Grado en ingeniería informática*
Área del Trabajo Final:	*Videojuegos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Carreras, multijugador, dibujo*

Resumen del Trabajo:

Este trabajo presenta “RaceTrace”, un videojuego multijugador por turnos enfocado a la estrategia, basado en el juego de lápiz y papel propuesto por el divulgador matemático Martin Gardneren en 1973. A diferencia de los juegos predominantes en el mercado, como los juegos como servicio y de disparos, en este destaca el multijugador estratégico por turnos.

El objetivo del jugador es completar una vuelta al circuito manteniendo la dirección del turno anterior y acelerando y frenando una casilla. Se utilizó una metodología de desarrollo iterativa y ágil, permitiendo adaptaciones continúas a partir de pruebas y retroalimentación.

Inicialmente, el prototipo se desarrolló en Python con funcionalidades básicas y posteriormente se trasladó a Unity para obtener una mejor experiencia gráfica y de juego. Durante el desarrollo se requirieron conocimientos matemáticos, de algoritmos, grafos, interfaz de usuario o gestión de proyectos aprendidos durante el grado.

Este trabajo no solo contribuye al sector de los videojuegos con una propuesta innovadora, sino que también aplica de manera práctica los conocimientos adquiridos en el resto de las asignaturas, resultando en un producto tangible y atractivo en el mercado actual de videojuegos.

Ha sido necesario realizar una buena planificación, una gestión de riesgos adecuada y una adaptación continua a los cambios. Al abordar un proyecto completo desde cero se han trabajado en todos los roles del desarrollo de un proyecto software.

Abstract:

This paper presents "RaceTrace," a turn-based multiplayer video game focused on strategy, inspired by the paper and pencil game proposed by mathematical popularizer Martin Gardner in 1973. Unlike the prevalent service games and shooters in the market, this game stands out with its strategic turn-based multiplayer gameplay.

The player's objective is to complete a lap around the circuit, maintaining the direction from the previous turn while accelerating or decelerating by one square at a time. An iterative and agile development methodology was employed, enabling continuous adaptations based on testing and feedback.

Initially, the prototype was developed in Python with basic functionalities and was later ported to Unity to enhance the graphical and gameplay experience. The development necessitated mathematical knowledge, algorithms, graph theory, user interface design, and project management skills learned during the degree program.

This work not only contributes an innovative proposal to the video game sector but also practically applies the knowledge acquired in various subjects, resulting in a tangible and attractive product in the current video game market.

Effective planning, adequate risk management, and continuous adaptation to changes were essential. By undertaking a complete project from scratch, experience was gained in all roles of software project development.

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

1. Introducción

1.1. Introducción/Prefacio

Desde un punto de vista personal los videojuegos siempre han formado parte de mi vida. Mi hermano era poseedor de una Sega Mega Drive y siempre me fascinó este tipo de entretenimiento que me trasladaba a mundos fantásticos, me hacían volar mi imaginación, me retaban a llegar un poco más allá con cada intento y a mejorar constantemente. Y todo ello utilizando unos pocos píxeles puestos en el lugar adecuado.

Pronto comencé a interesarme por el apartado más artístico del sector, fascinándome por los mods de juegos como Counter Strike o creadores de niveles como los del Starcraft y deseaba ser capaz de crear esos mundos y niveles que tanto me gustaban.

La inspiración para este trabajo final proviene cuando estaba en el colegio y jugaba con mis amigos. Solo disponíamos de una libreta cuadriculada y un boli y jugábamos a un juego de carreras por turnos. Dibujábamos un circuito y debíamos recorrerlo antes que el resto de los competidores. Nos pasábamos horas jugando a aquel juego y retándonos a ver quién era el mejor.

Más tarde salieron al mercado de los smartphones videojuegos como ‘Apalabrados’ o ‘Draw It’ que me recordaban a aquel juego y me hacían pensar que quizá a aquel juego de mi infancia podría jugar con mis antiguos amigos desde mi móvil. Con la democratización de herramientas para crear videojuegos gracias a motores como el Unity 3D, podría plasmar esa idea y llevarla a la práctica. Por ello me gustaría realizar aquel pasatiempo al que jugábamos en el colegio en forma de videojuego.

Figura 1: Sonic & Knuckles Sega de Mega Drive

1.2. Descripción/Definición

En el mercado han aparecido otros juegos para móvil multijugador por turnos y algunos de ellos como 'Apalabrados'[1] o 'Draw It'[2] tuvieron un éxito bastante extenso, pero actualmente no hay ningún juego similar al que se esté jugando de forma masiva. Existe en el mercado un hueco para nuevos juegos de este estilo con un multijugador asimétrico.

Se pueden encontrar algunos juegos de mesa como "Formula D"[3] o videojuegos como "T3 - Take the Turn" [4] que mezclan las carreras con los turnos, sin embargo, Steam no nos sugiere ningún juego relacionado que combine ambos géneros.

El trabajo que se pretende realizar es un juego de carreras por turnos en el que en te desplazas por un circuito y gana el primero que realice una vuelta completa sobre el mismo. El objetivo del producto es generar un trazado procedimental para que cada partida sea diferente y rete al jugador. Además, se estudiarán diferentes mecánicas que se puedan agregar para tener una mayor variedad en las partidas.

Se distribuirá el juego a través de itch.io de forma gratuita para que pueda ser ejecutado en cualquier dispositivo con navegador y su ejecutable para PC.

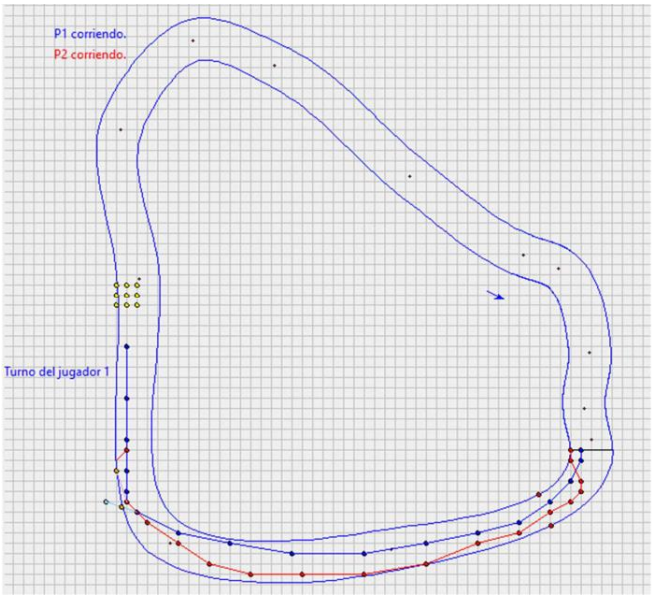


Figura 2: Prototipo del videojuego

En el siguiente video se puede ver un ejemplo de la jugabilidad del prototipo https://youtu.be/g2YR7LqrvBQ

1.3. Objetivos generales

A continuación, se detallan los objetivos de nuestro TF.

1.3.1. Objetivos principales

Objetivos de la aplicación/producto/servicio:

- Desarrollar un videojuego de carreras por turnos que ofrezca una experiencia al usuario entretenida y atractiva.
- Realizar un sistema de juego de turnos eficiente que permita competir contra otros jugadores.
- Confeccionar un sistema de generación de circuitos aleatorio de forma procedimental.
- Implementar un sistema de IA que permita competir contra un jugador controlado por la máquina.

Objetivos para el cliente/usuario:

- Proporcionar un medio de entretenimiento que permita competir en carreras por turnos contra adversarios controlados por IA u otros jugadores.
- Ofrecer una experiencia de usuario sencilla y fácil de comprender, asegurando que es accesible, pero permite profundizar en sus mecánicas a medida que vamos jugando.

Objetivos personales del autor del TF:

- Ganar experiencia en el desarrollo de un proyecto software con metodologías ágiles.
- Adquirir habilidades en el desarrollo de videojuegos utilizando el motor Unity.
- Explorar las mecánicas de juegos competitivos por turnos.

1.3.2. Objetivos secundarios

Objetivos adicionales que enriquecen el TF.

- Definir un estilo visual atractivo y uniforme.
- Implementar un tutorial sencillo para el usuario.
- Crear animaciones que den un aspecto fluido a todo el juego.
- Crear una versión para dispositivos móviles

1.4. Metodología y proceso de trabajo

Una posible opción sería optar por un desarrollo en cascada con una primera fase de diseño, codificación pruebas y con una fecha de entrega del producto a final del semestre. Sin embargo, al ser un producto con unos requisitos no cerrados y con gran incertidumbre, se optará por una metodología ágil basada en entregas continuas e incrementales. Esta metodología permite una mayor flexibilidad a la hora de afrontar cambios en los requisitos. Además, al ser un trabajo con gran componente creativo puede requerir de múltiples revisiones para refinar diferentes aspectos del juego como el gameplay o hacer ajustes gráficos que mejoren la experiencia. Así, sumaremos más funcionalidad o mejorando el producto actual según el desarrollo avance.

Para esta metodología se ha hecho uso de la herramienta Bitbucket[5] en la que se puede tener un repositorio de código y Trello[6] para la descomposición del proyecto en tareas. Ambos productos pertenecen a la misma empresa y disponen de planes gratuitos para equipos de trabajo pequeños.

La gestión del proyecto se ha realizado a través de la herramienta Teamgantt[7] que permite hacer un diagrama de Gantt con las fechas límite de cada entrega y desglosar las tareas que vamos a realizar en cada una de ellas. De esta manera podemos visualizar el avance del proyecto.

Debido al desconocimiento del motor gráfico Unity, se ha optado por realizar un prototipo en lenguaje Python que permita estudiar algoritmos para la generación procedimental de circuitos y testear la jugabilidad básica del videojuego. Después se realizará con el motor Unity actualizado a la última versión estable disponible al comienzo del desarrollo.

En resumen, se ha utilizado una metodología Agile[8] con 4 Sprints de aproximadamente 1 mes cada uno. Se ha descompuesto el problema global en pequeñas tareas abordables unos días o semanas, para realizar una entrega al final de cada uno de estos periodos. Cada entrega ha sido incremental aumentando y corrigiendo las entregas anteriores hasta obtener el producto final.

1.5. Planificación

Para este trabajo se ha dividido en una serie de tareas agrupadas en cinco fases que coinciden con las diferentes entregas de las PEC del itinerario de la asignatura. En las cuatro primeras entregas se ha ido trabajando sobre el prototipo, mientras que la entrega final ha sido la preparación de la defensa final del proyecto, elaborando un video y los entregables. Cada tarea es iterativa y se basa en el trabajo de las tareas anteriores, refinando, en caso necesario, el trabajo previo para ajustarse a las nuevas necesidades.

A continuación, se detallan las distintas entregas con sus fechas y las tareas de cada una de ellas. Posteriormente se presenta un diagrama de Gantt creado en la plataforma Teamgantt.

PEC1 (17/9/23 - 8/10/23):

- Idea del juego y conceptualización.
- Prototipo en Python.

PEC 2 (9/10/23 - 12/11/23):

- Aprendizaje de Unity.
- Creación de esqueleto de proyecto.
- Generación de circuito.
- Lógica movimiento jugador.

PEC 3 (**13/11/23 - 17/12/23):**

- Menú principal básico.
- Lógica partida.

PEC 4 (18/12/23 - 14/01/24):

- Implementación IA.
- Creación de menú principal, menú de pausa y fin de la partida.
- Creación de animaciones.
- Definición de estilo gráfico final.
- Implementación de tutorial.
- Pruebas y corrección de errores.
- Animaciones.
- Sonidos
- Publicación de versión final.

PEC 5 (14/01/24 - 31/01/24):

- Preparación defensa del TFG

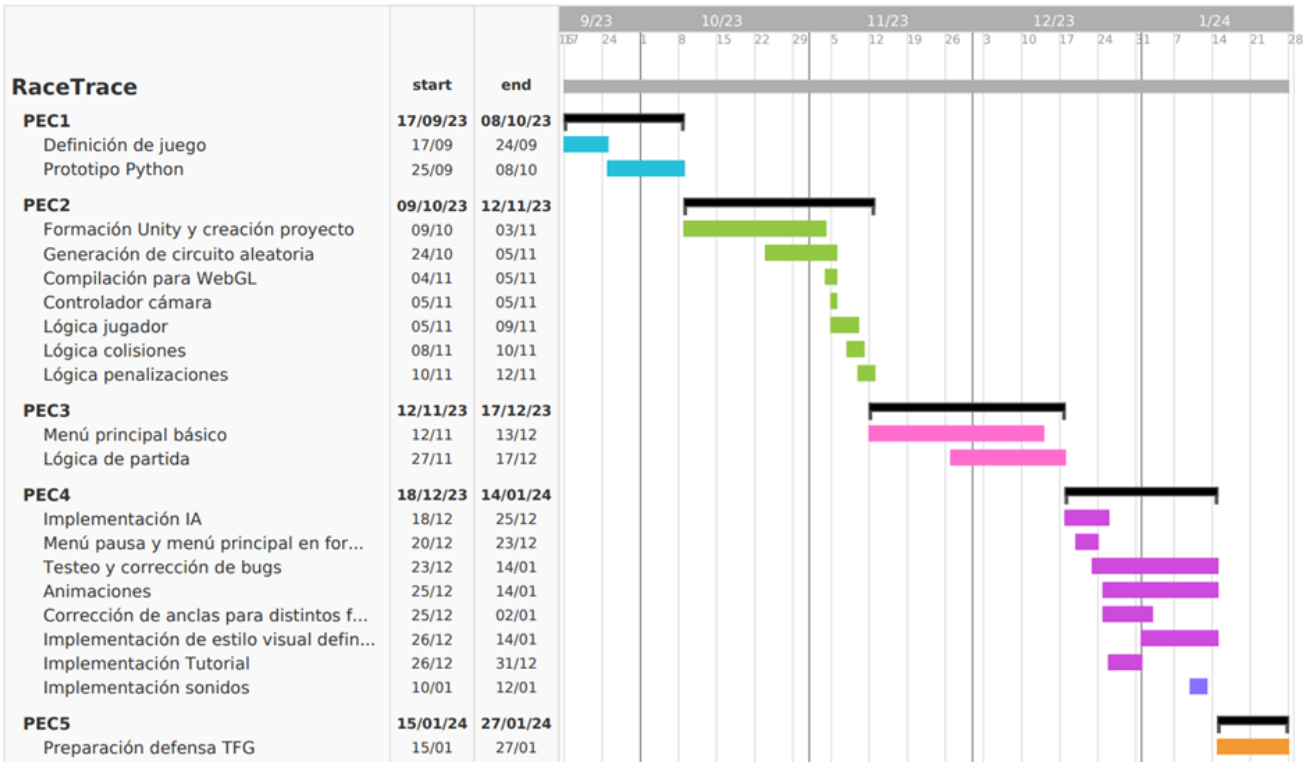


Figura 3: Diagrama de Gantt

El tiempo invertido en cada una de las tareas es el siguiente:

Tarea	Fecha Inicio	Fecha fin	Días	
PEC1	17/09/2023	8/10/2023	22	
PEC2	9/10/2023	12/11/2023	35	
PEC3	12/11/2023	17/12/23	36	
PEC4	18/12/23	14/01/24	28	
PEC5	15/01/24	27/01/24	13	

Tabla 1: Planificación PECs

Tarea	Fecha Inicio	Fecha fin	Días	Estimación Horas
Definición de juego	17/09/23	24/09/23	8	8
Prototipo Python	25/09/23	8/10/2023	14	42
Formación Unity y creación proyecto	9/10/2023	3/11/2023	26	60
Generación de circuito aleatoria	24/10/23	5/11/2023	13	50
Compilación para WebGL	4/11/2023	5/11/2023	2	2
Controlador cámara	5/11/2023	5/11/2023	1	3
Lógica del jugador	5/11/2023	9/11/2023	5	15
Lógica colisiones	8/11/2023	10/11/2023	3	9
Lógica penalizaciones	10/11/2023	12/11/2023	3	10
Menú principal básico	12/11/2023	13/12/23	32	4
Lógica de partida	27/11/23	17/12/23	21	30
Implementación IA	18/12/23	25/12/23	8	20
Menú pausa y menú principal	20/12/23	23/12/23	4	10
Testeo y corrección de bugs	23/12/23	14/12/23	23	20
Animaciones	25/12/23	14/01/24	21	20
Corrección de anclas	25/12/23	2/1/2024	9	3
Implementación de estilo visual def.	26/12/23	14/01/24	20	70 (35 horas diseño)
Implementación Tutorial	26/12/23	31/01/24	18	10
Implementación sonidos	10/01/24	12/01/24	3	8
Preparación defensa TFG	15/01/24	14/01/24	13	30

Tabla 2: Planificación tareas

Como vemos en la tabla anterior, la estimación de horas para el desarrollo del proyecto es de 424 horas, siendo 35 de ellas de diseño y 389 de programación.

1.6. Presupuesto

Para la realización de este trabajo final se dispone de un presupuesto limitado. El objetivo es hacer uso, en la medida de lo posible, de licencias gratuitas de software para equipos pequeños y assets propios o gratuitos siempre que sea posible.

La estimación del presupuesto para el desarrollo de este videojuego es el siguiente:

Equipo humano:

Item	Descripción	Precio
Programación	El coste estimado de diseño y programación del proyecto, suponiendo un salario de 25€ la hora y un total de 389 horas.	9725 €
Diseño interfaz	Supondremos un salario de diseñador también de 25 € y un total de 35 horas	875 €
TOTAL		**10600 €**

Tabla 3: Costes humanos

Suponiendo el coste del programador y diseñador en 25€ la hora tenemos un total de 10600 €.

Hardware

Item	Descripción	Precio
Steam Deck	Ordenador portatil equipado con una APU AMD y 16GB RAM	419 €
NVMe SSD 1 TB	Disco SSD de mayor capacidad para instalación de Windows	100 €
Arzopa G1	Pantalla externa de 15.6’ con resolución 1080p y 144Hz	100 €
Logitech K260	Combo teclado y ratón inalámbricos	20 €
Ipad 9 gen	Tablet de marca Apple	319 €
Lapiz BT	Lápiz para dibujar en Ipad	25 €
TOTAL		**983€**

Tabla 4: Costes hardware

El coste total del equipo informático es de 983 €.

Software y licencias:

Item	Descripción	Precio
Licencia Windows 11	Licencia de Windows 11 para nuestro equipo.	6 €
Bitbucket	Licencia repositorio código para equipos pequeños	0 €
Teamgantt	Herramienta online para hacer diagramas de Gantt del proyecto	0 €
Unity	Motor gráfico. Se ha utilizado una licencia gratuita de equipos pequeños con pocos ingresos.	0 €
Concepts	Aplicación para Ipad	23 €
Figma	Herramienta online para diseñar.	0 €
Gimp	Programa de edición de imágenes gratuito.	0 €
Audacity	Programa de edición de audio gratuito.	0 €
TOTAL		**29 €**

Tabla 5: Costes software y licencias

El coste en software y licencias ha sido de 29 €.

Assets y recursos:

Item	Descripción	Precio
Asset Book - Page Curl Pro	Asset de libro para el menú principal	11,08 €
Asset Cute Cartoon Mobile GUI	Asset para la interfaz gráfica. Usado en el prototipo y como base para la versión final.	0 €
Asset FREE Stylized PBR Textures Pack	Assets de texturas estilo cartoon. Se ha usado una textura de madera de estos assets como base para la mesa de fondo del menú.	0 €

Lilita One	Fuente de google con licencia Open Font License	0 €
Margarine	Fuente de google con licencia Open Font License	0€
Script Dependency Visualizer	Asset para visualizar la arquitectura y dependencia de los scripts	0 €
Sonido	Se han utilizado sonidos con licencia Creative Commons 0	0 €
TOTAL		**11,08 €**

Tabla 6: Costes assets y recursos

El coste total en recursos ha sido de 11,08 €.

Tras agregar todos los costes de desarrollo detallados anteriormente, el presupuesto total ha sido de 11623,08 €.

Item	**Descripción**	**Precio**
Equipo humano	Coste total programación y diseño	10600 €
Hardware	Coste total equipo informático	983€
Software y licencias	Coste total software	29 €
Assets y recursos	Coste total en recursos gráficos	11,08 €
TOTAL		**11623,08 €**

Tabla 7: Costes totales

1.7. Estructura del resto del documento

En los siguientes capítulos de la memoria haremos un análisis del mercado viendo nuestro público objetivo. Definiremos cual es nuestra propuesta, nuestros objetivos, modelo de negocio y marketing.

En el siguiente capítulo se detallará el proceso de diseño realizado, su arquitectura, interfaces, lenguajes de programación utilizados sin entrar en la implementación, que se explicará en el siguiente capítulo junto a los requisitos e instrucciones de instalación. Después se hará una demostración de uso y se mostrarán los prototipos realizados. Por último, detallaremos las conclusiones personales que se han extraído del proyecto realizado, así como posibles líneas de trabajo futuras.

2. Análisis de mercado

2.1. Público objetivo y perfiles de usuario

En el sector de los videojuegos se pueden distinguir varios tipos de perfil de usuarios según su dedicación.

- Jugadores casuales: Son jugadores que ocasionalmente buscan distraerse con un videojuego. Buscan experiencias relajadas que puedan jugar mientras esperan en el transporte público o mientras descansan realizando otras tareas simultáneamente como ver la televisión. Este tipo de jugadores no suele gastar mucho dinero por adquirir un videojuego, incluso habitualmente juega a juegos gratuitos. En ocasiones pueden adquirir algún cosmético o añadido para el juego al que le están dedicando tiempo por pequeñas cantidades de dinero.
- Jugadores competitivos: Son jugadores que disfrutan de desafíos en sus videojuegos, ya sea compitiendo contra otros en partidas multijugador o contra la dificultad del propio juego. Estos jugadores suelen centrarse en un juego y pueden llegar a dedicarle mucho tiempo.
- Jugadores dedicados: Son jugadores cuyo principal o uno de sus principales hobbies son los videojuegos. Estos usuarios suelen invertir mucho tiempo y dinero en ellos buscando experiencias variadas y más avanzadas.

También se pueden diferenciar por los perfiles de jugadores en función de la plataforma en la que juegan.

- Dispositivos móviles: Estos juegos son más accesibles y tienen un mercado mucho más masivo, con una mayor audiencia debido a la facilidad de uso y disponibilidad. En estos dispositivos suelen ser frecuentes los videojuegos gratuitos o de coste muy bajo. Su modelo negocio puede basarse en publicidad o compras dentro de la propia aplicación. Los usuarios casuales suelen ser los más frecuentes.
- Videoconsolas: Son dispositivos centrados exclusivamente en videojuegos por lo que tienen mayor potencia y capacidad que los dispositivos móviles. En estas plataformas encontramos tanto jugadores casuales como jugadores dedicados.
- Ordenadores: Este es el que nos ofrece la mayor gama de géneros y estilos de juego. Podemos encontrar desde juegos más amateur hasta los juegos más avanzados del mercado. Debido a la gran heterogeneidad de los ordenadores tenemos desde

jugadores casuales que aprovechan su ordenador de trabajo o estudio para jugar a juegos más casuales hasta jugadores dedicados o competitivos que tienen los mejores componentes para tener la mejor experiencia.

Además, existen otros factores que pueden caracterizar al usuario y sus gustos personales, como pueden ser la edad, su experiencia de juego, el interés por distintos géneros…

El juego de este proyecto está diseñado para ser partidas cortas, de unos cinco minutos, con unas mecánicas sencillas, pero con capacidad de profundizar en ellas. Por lo tanto, el público objetivo serían los jóvenes que juegan de manera casual en dispositivos móviles u ordenadores sin grandes capacidades técnicas.

2.2. Estado del arte

Según un informe de Newzoo's Analytics[9] publicado en mayo del 2020, el mercado de los videojuegos genera 159,3 mil millones de dólares al año con un crecimiento anual del 9.3%. Este mercado se desglosa en un 48% en dispositivos móviles, un 23% en ordenadores y un 28% en videoconsolas.

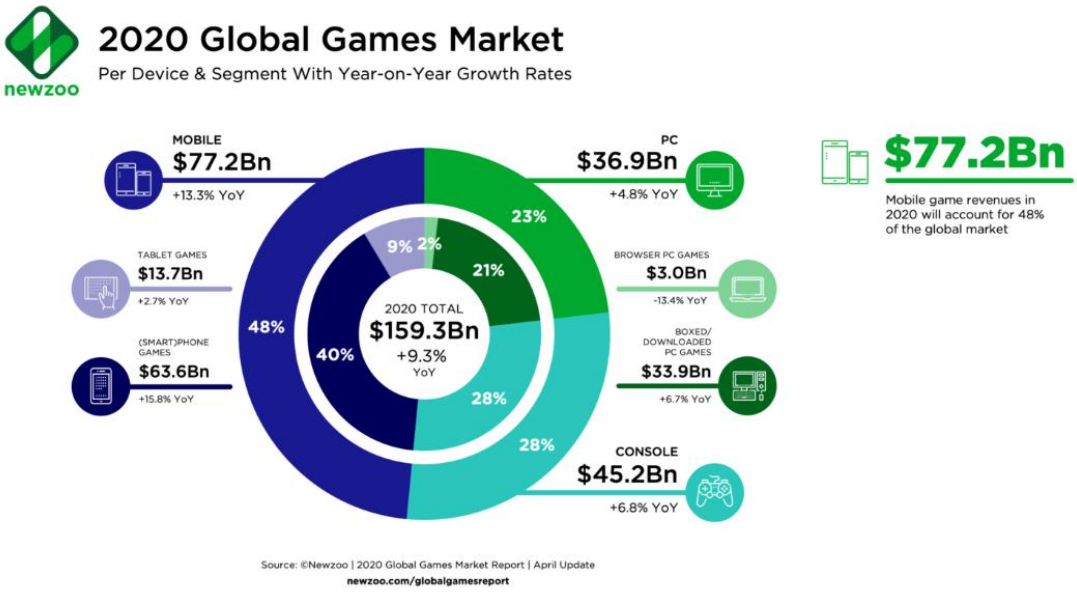


Figura 4: Mercado global videojuegos Newzoo's Analytics

No obstante, a pesar del enorme mercado de los videojuegos, otro punto importante es el gran número de aplicaciones publicadas en todas las tiendas, tanto en dispositivos móviles

como en las distintas tiendas de juegos de ordenador. En la Play store se publican diariamente 1800 nuevas, teniendo un total de más de 3,55 millones. [10][11]

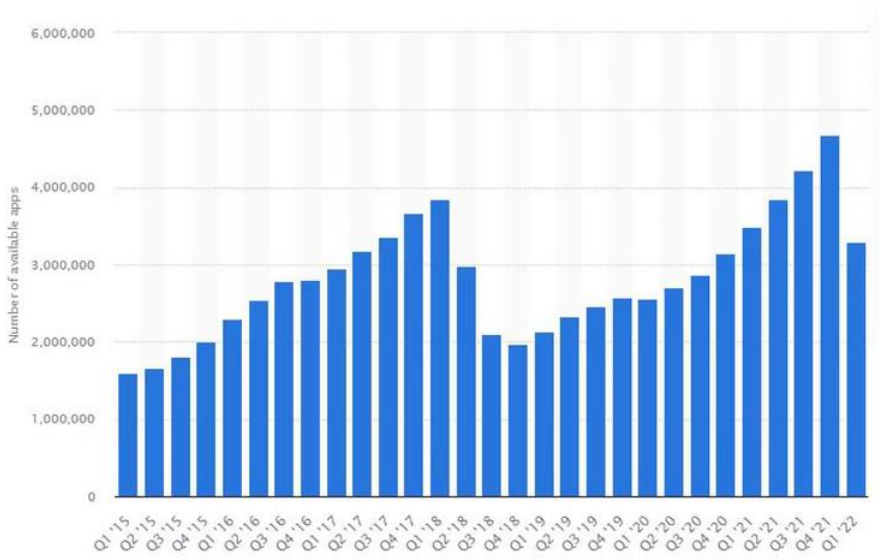


Figura 5: Número de aplicaciones publicadas en Play Store. Fuente Statista.

En Steam en el año 2020 se publicaron 9279 videojuegos[12], el doble que en 2016. De estos la mitad vendieron menos de 640 unidades. El 90% de los ingresos de la plataforma se reparte entre el 5% de los juegos más exitosos.

Otro factor importante es el mercado de la nostalgia[13]. Existen en el mercado muchos relanzamientos de productos que apelan a la nostalgia del jugador por su niñez. Ejemplos de esto los podemos ver en el sector de los videojuegos en productos como las consolas “Nes mini” o relanzamientos de videojuegos para plataformas actuales. En este caso, este juego ha sido jugado por algunos niños en su colegio y se busca atraerlos al videojuego.

2.3. Revisión de la tecnología

El prototipo inicial se ha realizado en Python debido a que este lenguaje de programación permite hacer prototipos de forma muy rápida. En este se han probado los distintos algoritmos matemáticos para la generación de circuitos y la implementación de una inteligencia artificial contra la que poder competir. Una vez realizado este prototipo, ha sido necesario el uso de un motor gráfico facilitara la integración de más funcionalidad y la exportación a distintas plataformas como pueden ser el PC o los dispositivos móviles. Para ello se estudiarán los principales motores gráficos de propósito general de mercado:

- Unity: Es un motor gráfico propietario que permite desarrollar videojuegos de forma gratuita siempre que se cumpla una serie de condiciones. Para estudios pequeños con pocas ventas se puede utilizar libremente sin pagar, y con él, se pueden desarrollar productos tanto 2D como 3D con un acabado profesional, pudiendo tener una calidad visual muy alta. Tiene una gran comunidad de desarrolladores indies y hay una extensa documentación. No obstante, durante el año 2023 la empresa cambió las políticas de monetización para desarrolladores y perdió la confianza de un gran número de ellos. Se han desarrollado productos comerciales como 'Ori and the blind forest' o 'Cuphead' con esta herramienta[14].

Figura 6: 'Cuphead' y 'Ori and the Will of the Wisps, videojuegos desarrollados con Unity

- Unreal Engine: Es el producto más avanzado del mercado, permitiendo aplicar efectos visuales de calidad fotorealista. Los proyectos con mayor presupuesto de la industria suelen licenciar este motor gráfico para su desarrollo y se ha convertido en un estándar entre los grandes estudios. Es un motor pensado para videojuegos en 3D y tiene una curva de dificultad alta. Es gratuito para estudios pequeños y no se empieza a pagar hasta alcanzar una gran cantidad de beneficios. Este motor tiene el apoyo de Epic Games y es utilizado para el desarrollo de su exitoso videojuego 'Fortnite', por lo que está en constante evolución. Ha sido utilizado como base para el desarrollo de otros juegos como 'Hellblade' o 'Yoshi's crafted wold' [15].

Figura 7: 'Hellblade 2' y 'Yoshi's crafted world', Unreal engine.

- Godot[16]: Es un motor open source que ha ganado gran popularidad entre los desarrolladores indie a raíz de la polémica de Unity por el cambio de políticas de monetización[17]. Es un motor enfocado en desarrollos 2D, aunque tiene capacidad para desarrollar proyectos en 3D. Está en constante evolución por la comunidad, sin embargo, todavía no hay mucha documentación para desarrolladores y no está del todo pulido. Algunos videojuegos comerciales desarrollados en este motor son el 'Deponia' para PS4 y el 'Sonic Colors Ultimate'.

Figura 8: 'Deponia' y 'Sonic Colors Ultimate, Godot

- Existen otros motores alternativos en el mercado como puede ser GameMaker Studio[18]. Este motor está más centrado en gráficos 2D y con algunos componentes sencillos 3D. Este motor es también utilizado por estudios pequeños, pero tiene menos documentación disponible al no estar tan extendido. Algunos juegos exitosos de estos motores pueden ser 'Chicory: A Colorful Tale' o 'Nuclear Throne'

Figura 9: 'Chicory: A Colorful Tale' y 'Nuclear Throne', GameMaker Studio

En este proyecto se va a escoger Unity para desarrollar el videojuego 2D al ser una herramienta muy sólida y no necesitar recursos 3D muy avanzados. Además, tiene de una curva de aprendizaje moderada y se dispone de una amplia documentación y recursos que pueden facilitar el desarrollo y publicación del juego. Conjuntamente, hay una comunidad grande y activa que puede facilitar la colaboración y consejos por parte de otros desarrolladores para este o para futuros proyectos.

3. Propuesta

3.1. Descripción del juego

RaceTrace es un videojuego de carreras por turnos en el que varios jugadores compiten por dar una vuelta completa al circuito. Estas pistas están generadas de manera procedural mediante algoritmos matemáticos. El jugador debe evitar chocar con los bordes del circuito mientras completa el trazado del circuito.

El juego original fue propuesto por el divulgador matemático y filósofo Martin Gardner, en enero del 1973, en su columna “Mathematical Games” de la revista “Scientific American”. El nombre del juego original fue RaceTrack[19][20].

El juego se basa en el concepto de movimiento estratégico. En cada turno el desplazamiento mantiene la dirección del turno anterior, pero el jugador puede acelerar o frenar una casilla. De esta manera si el jugador quiere ir más rápido seleccionará los puntos más alejados a su posición actual, si quiere reducir su velocidad los puntos más cercanos y si desea ir girando seleccionará alguno de los puntos laterales en la dirección en la que se oriente la curva. De esta manera el usuario debe moderar la aceleración o no será capaz de tomar las curvas y colisionará contra los bordes del circuito, siendo sancionado sin jugar un número de turnos[21].

En la siguiente figura podemos ver un esquema de cómo sería el turno de un jugador. El punto central amarillo se corresponde con el movimiento del turno anterior y puede seleccionar además cualquier de los puntos contiguos. Los puntos más cercanos los podría utilizar para frenar en caso de curva y los puntos más alejados para acelerar en las rectas.

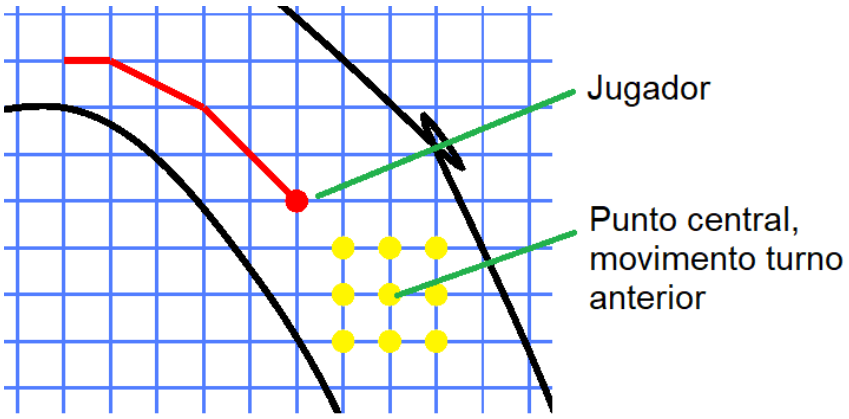


Figura 10: Descripción jugabilidad en prototipo.

3.2. Definición de objetivos

El objetivo del jugador es completar una vuelta al circuito antes que el resto de los jugadores. Si choca con el borde del trazado será sancionado un número de turnos en función de su velocidad. De esta manera se evitará que el jugador de manera intencionada colisione para optimizar una trazada. La penalización siempre debería ser mayor que el tiempo que le hubiera costado detenerse. Posteriormente reaparecerá en el punto más cercano.
El jugador que complete el circuito antes ganará. Si los jugadores completan el circuito en el mismo número de turnos se podría contemplar que jugador ha sido el que menos colisiones ha realizado para desempatar.

Un jugador no podrá colocarse en la misma casilla que otro, de esta manera se evita que los jugadores copien los movimientos y se puede obligar a otro jugador a colisionar contra el lateral si se coloca en un punto estratégico.

3.3. Concept arts

Se busca trasladar al jugador la sensación nostálgica de jugar en una libreta con un bolígrafo. El tablero de juego simulará ser una libreta cuadriculada como las utilizadas en las escuelas y el circuito aparentará estar dibujado a mano con un bolígrafo. Cada jugador utilizará un bolígrafo o un lápiz para ir dibujando la trayectoria que realiza a través del circuito.

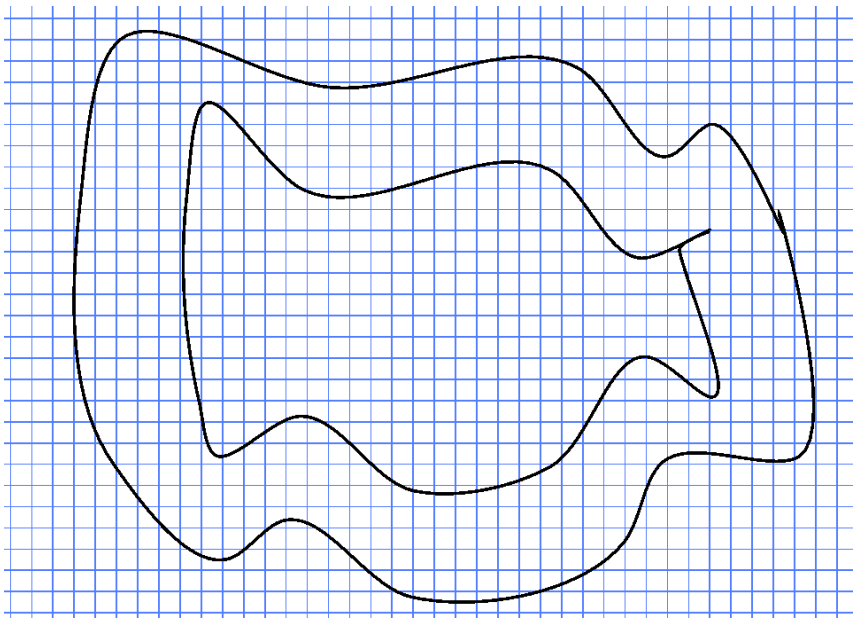


Figura 11: Ejemplo de circuito sobre libreta cuadriculada.

Cada jugador elegirá un personaje con el que quiera competir. Estos personajes serán caricaturas de elementos que se utilizan habitualmente en los colegios como pueden ser bolígrafos o lápices.

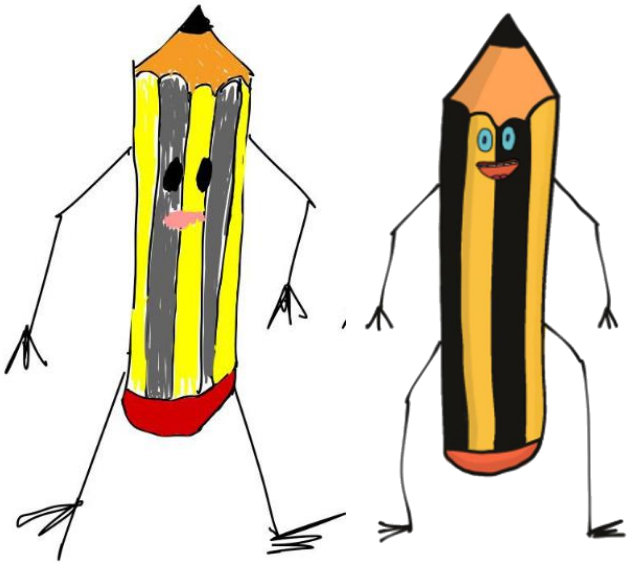


Figura 12: Bocetos personajes

El objetivo es crear un producto multiplataforma, con posibilidad de ser jugado tanto en ordenadores, navegadores o dispositivos móviles. Sin embargo, la optimización de controles para dispositivos móviles se ha dejado como línea de trabajo futura.

3.4. Modelo de negocio

Los gastos producidos por el desarrollo de este videojuego se encuentran en el apartado 1.6 Presupuesto. Aunque el objetivo del desarrollo no es buscar un beneficio o ser viable económicamente, se podrían estudiar modelos para monetizarlo.

Un posible modelo de negocio sería vender el producto por una pequeña cantidad de dinero. Este modelo limitaría la distribución al ser el usuario de dispositivos móviles más reacio a pagar por las aplicaciones. Sin embargo, los usuarios de ordenadores o videoconsolas si están más dispuestos a pagar por los videojuegos.

Otro posible modelo de negocio sería la introducción de anuncios. Esta publicidad no debería ser demasiado intrusiva ya que puede desincentivar al usuario a seguir jugando, por lo que podría introducir al finalizar una partida o en el menú de pausa.

Por último, se podría optar por el uso de cosméticos para los personajes. Estos cosméticos se podrían comercializar mediante micropagos o la introducción de un pase de temporada. Este modelo se utiliza habitualmente en juegos gratuitos como 'Fortnite'. La desventaja de estos modelos es que requieren de un trabajo continuo durante toda la vida útil del producto.

En este proyecto se podría optar por un modelo híbrido como el videojuego 'Vampire Survivors'[22], que es gratuito con publicidad en dispositivos móviles, y de pago con un coste muy bajo en ordenadores o videoconsolas.

Se va a publicar en la plataforma itch.io[23] de forma gratuita y posteriormente al lanzamiento se estudiará la viabilidad de introducir cosméticos para los personajes. Se debe diseñar para no descartar ninguno de los modelos de monetización.

3.5. Estrategia de marketing

Como se ha comentado anteriormente, no está previsto a corto plazo lanzar el videojuego para dispositivos móviles. Para ordenadores será publicado de forma gratuita en itch.io. Esta página permite a los desarrolladores independientes publicar sus juegos y elegir la forma de monetización. Más adelante se estudiará la publicación en otras plataformas como Steam o Play Store por una pequeña cantidad de dinero.

El nombre "RaceTrace" ha sido seleccionado como parte fundamental de la estrategia de branding. Éste refleja la identidad y los conceptos principales del juego. Por un lado, indica que es un videojuego centrado en las carreras y la competitividad y por otro sugiere que el personaje deja un rastro. Este nombre tiene una gran resonancia, al ser dos palabras cortas que riman entre ellas y son fáciles de recordar. Además, no hay ninguna aplicación con ese nombre en las principales tiendas de videojuegos.

Debido al limitado presupuesto de este proyecto, no será posible ninguna campaña de promoción del producto.

4. Diseño

Tal y como se ha indicado en el apartado anterior, se ha optado por utilizar el motor gráfico Unity para desarrollar el videojuego. Uno de los principales motivos del uso de este motor es la relativamente baja dificultad de aprendizaje y a la disponibilidad múltiples recursos y assets.

En este capítulo se detallará como se ha realizado el diseño a alto nivel del proyecto y la arquitectura de este. En el siguiente capítulo se detallará la implementación de cada componente y algoritmos utilizados.

4.1. Arquitectura general del juego

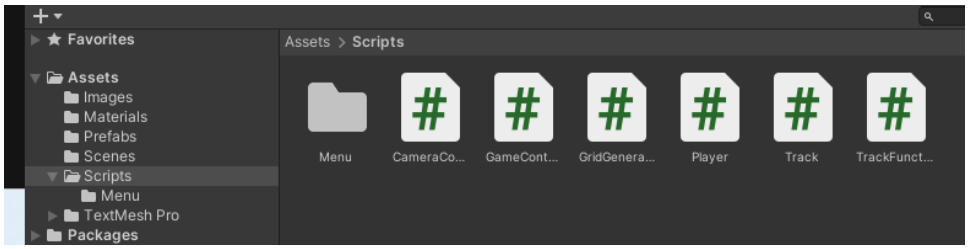


Figura 13: Estructura assets proyecto Unity

Los proyectos en Unity tienen una arquitectura general basada en la utilización de diferentes recursos (Assets), que componen el juego o aplicación A continuación se definirán los recursos y la relación entre ellos para formar la estructura del proyecto.

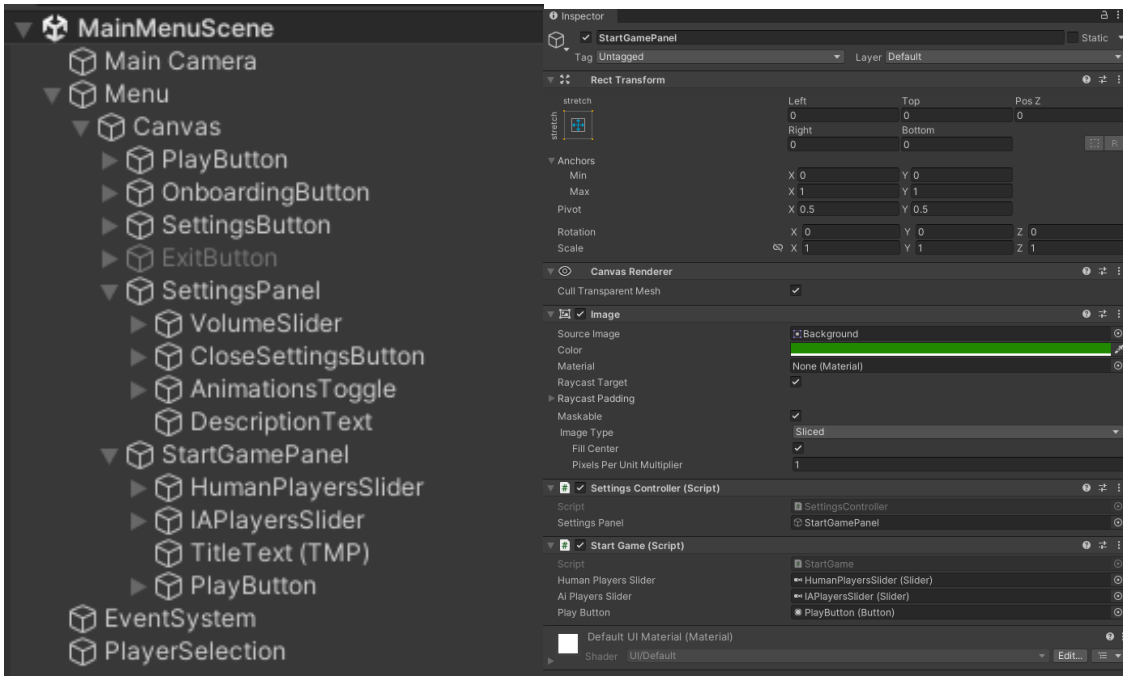


Figura 14: GameObjects y componentes

- Escenas (Scenes): Los proyectos están compuestos por una o más escenas que forman un nivel. Cada una de ellas contiene elementos de juego (GameObjects). En memoria únicamente se carga una escena y si queremos cambiar a otro menú u otro nivel debemos borrar la actual e instanciar la nueva.

- Objetos de juego (GameObjects): Estos son los elementos fundamentales que forman la escena. Cada objeto del juego, como puede ser el circuito, un jugador o un controlador de lógica, equivale a uno o varios objetos. A estos GameObjects se le pueden asociar componentes que definen su comportamiento, sus gráficos o su sonido.

- Componentes y Scripts: Los componentes son módulos que se asocian a los GameObjects para dotarlos de funcionalidad, físicas, sonidos, etc. Un tipo de componente son los scripts, programas escritos en lenguaje C# para definir comportamientos y lógica personalizados a dichos GameObjects. En este proyecto se ha utilizado Visual Studio Code para editar estos Scripts.

- Materiales y gráficos: Estos definen la apariencia visual de los GameObjects. Se utilizan Shaders, texturas, modelos 3D o Sprites. Estos se pueden aplicar para cambiar el aspecto.

- Sistema de audio: Son componentes que permiten reproducir sonidos o música a través de clips de audio. Se utilizan ficheros en formato mp3, wav, ogg, etc. Existen sistemas emisores de sonido, Audio Sources y sistemas que reciben sonido, Audio Listeners. En cada audio fuente de sonido se puede configurar el clip a reproducir, el volumen, el pitch, la reverberación…

- Sistema de animación: Estos componentes permiten definir secuencias de movimientos para los objetos. Se puede concretar el movimiento o los cambios que debe realizar un gameobject a partir de imágenes clave. El sistema de animación las interpolará para calcular como debe animarse en los frames intermedios. Además, los objetos se pueden comportar como máquinas de estados, configurando el

comportamiento en cada estado y definiendo como se deben realizar las transiciones entre ellos.

- Sistema de partículas: Se utilizan para crear efectos visuales dinámicos como puede ser el fuego, el humo, explosiones, etc. En este proyecto se ha decidido no utilizar sistema de partículas ya que todo se hará mediante animaciones o scripts.

- Interfaz de usuario: Estos elementos son botones, barras, menús u otros objetos interactivos o informativos que se colocan en un elemento Canvas de Unity. Estos GameObjects utilizan la posición relativa de la cámara y no la posición en el mundo real.

En la siguiente figura se puede ver un ejemplo de GameObjects y componentes del juego. Se observa que los elementos de juego se estructuran en un árbol y agrupados por bloques. A cada objeto se les añade componentes para dotarlos de funcionalidad.

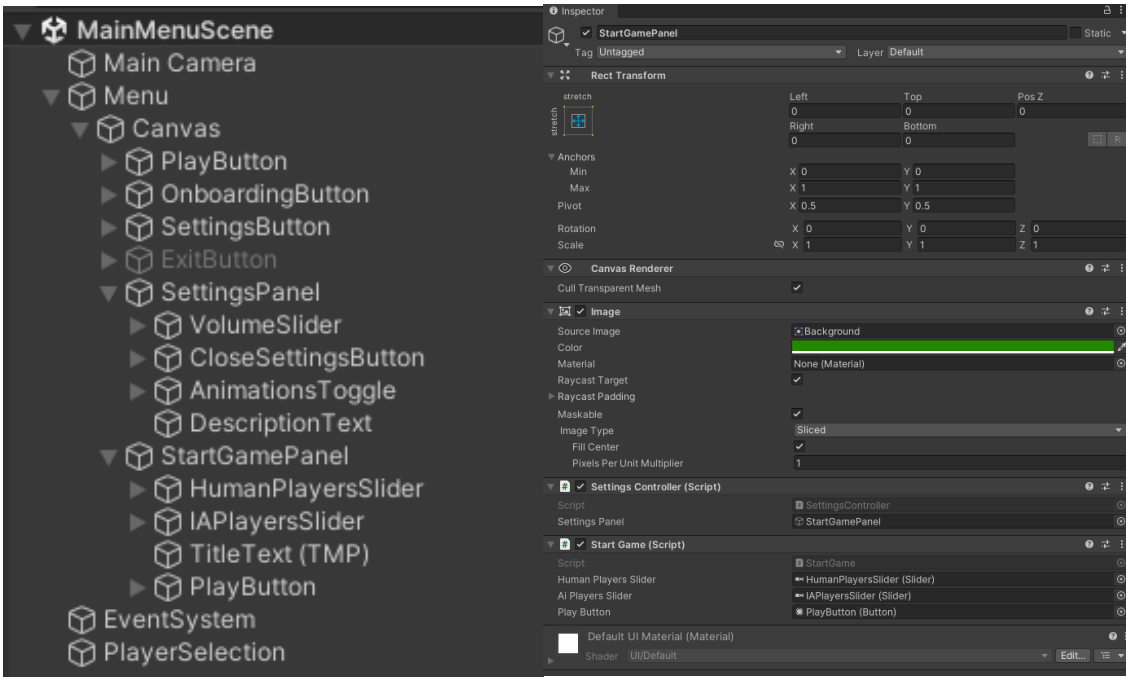


Figura 15: GameObjects y componentes

4.2. Arquitectura de la información y diagramas de navegación

En esta sección se detalla el flujo de información a alto nivel sin entrar en la implementación ni en las decisiones que se hayan tomado, que serán detalladas en el siguiente capítulo de la memoria.

4.2.1. Escenas

El proyecto está compuesto por tres escenas “MainMenuScene”, “TutorialScene” y “GameScene”.

- **MainMenuScene**: Es la escena inicial al abrir el videojuego y se utiliza como nexo central desde el que podemos acceder al resto de escenas. Además, permite configurar las opciones para las partidas, desde el número de jugadores hasta opciones de visualización y de sonido. Está formado por varias subsecciones en forma de páginas de libro. En la primera página se encuentran los botones de “Jugar”, “Tutorial”, “Opciones” y “Salir”. Este último botón está deshabilitado en navegadores ya que no es necesario.

  Al hacer clic sobre jugar se accede a la sección de opciones de partida, donde se puede configurar el número de jugadores y número de jugadores controlados por la inteligencia artificial que se quieren en el juego.

  El siguiente enlace lleva a la escena de tutorial que detallaremos a continuación.

  El tercer botón abre la subsección de opciones, donde se puede configurar el nivel de dificultad del juego, habilitar o deshabilitar diferentes opciones de visualización o modificar el volumen.

  Por último, está el botón de Salir, que como se ha comentado, se deshabilitará cuando se compile el juego para ser ejecutado desde un navegador.

  Desde cada una de las subsecciones se puede volver a la sección principal y desde cada escena salir a la escena de menú principal.

La transición desde menú principal a las otras escenas se ha decidido hacer fluida, desarrollando una animación asíncrona antes de cargar cada escena. De esta manera se ha evitado darle al jugador la sensación de corte abrupto nada más pulsar el botón.

- **TutorialScene**: Esta escena hace de onboarding para el jugador. Se hace un breve resumen del funcionamiento del juego a través de una partida guiada donde se le explican los diferentes conceptos del juego y un repaso rápido de los controles básicos. Esta escena es una extensión de la escena de juego, haciendo uso de la mayor parte de lógica de una partida normal, pero con un comportamiento más guiado. Se detallarán el funcionamiento en una sección posterior.

- **GameScene**: Es la escena principal de nuestro juego y la que alberga el mayor peso a nivel de lógica y código. En ella se genera un circuito aleatorio que el jugador puede cambiar y posteriormente se realiza la partida. En ella cargamos las opciones que el jugador ha seleccionado desde el menú principal.

De forma gráfica, la arquitectura de escenas es la siguiente:

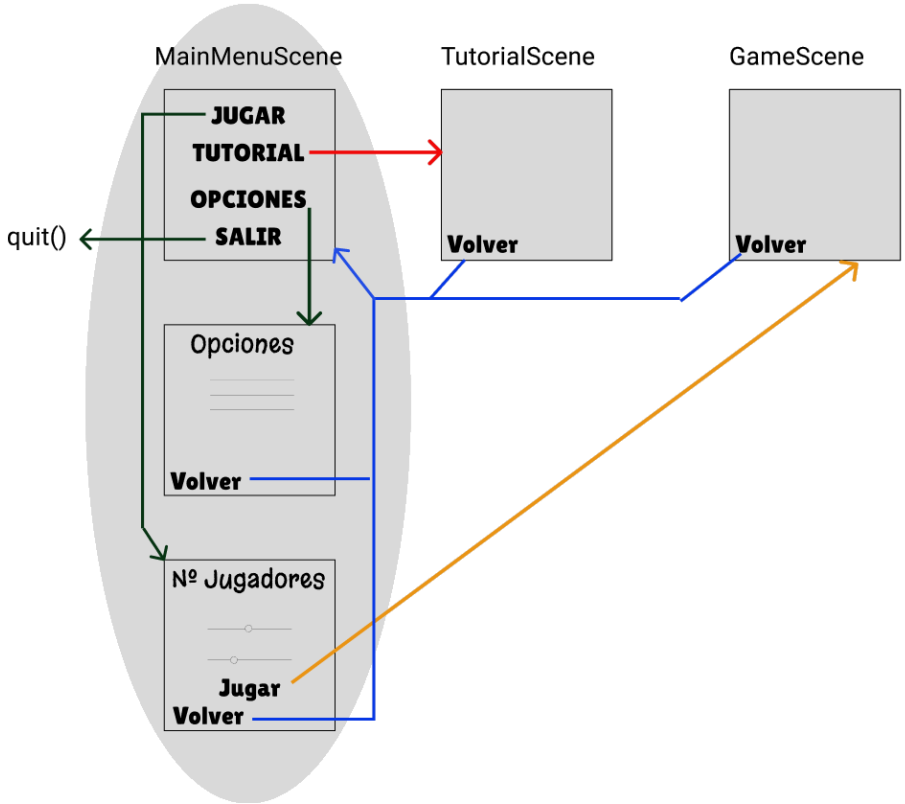


Figura 16: Esquema arquitectura escenas

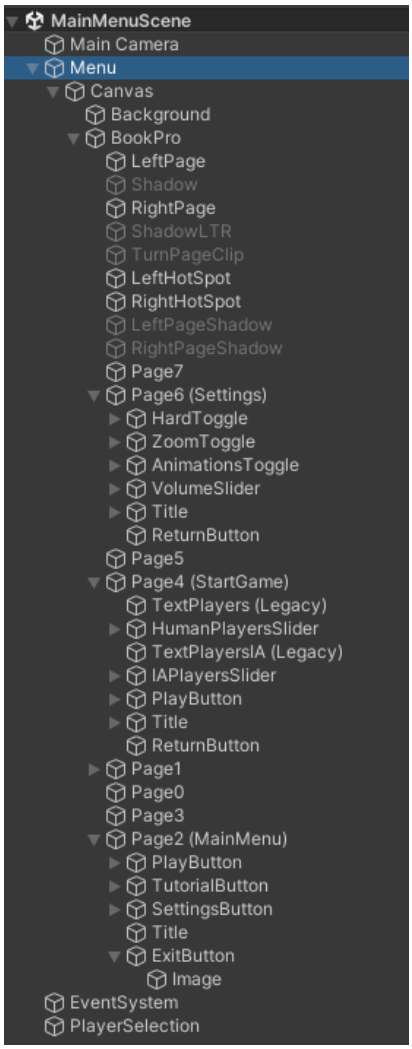
Como se observa en la figura anterior, el proyecto consta de tres escenas. Desde la escena de menú principal se puede acceder a las otras dos escenas, flechas roja y naranja en el esquema. Además, desde cualquier punto es posible volver al menú principal a la sección inicial, haciendo una carga de escena si estamos en otra o desplazándonos hacia la sección si ya estamos en esta escena.

A nivel de clases se ha optado por utilizar una arquitectura Fachada (Facade) [24] que permite tener en cada escena una clase principal que actúa como controlador e interfaz simple con el resto de subsistemas. En cada escena hay una clase SceneController que permite desacoplar la lógica del resto de elementos. De esta manera es posible abstraerse de los algoritmos complejos como la inteligencia artificial, algoritmos complicados de generación y gestión del circuito, etc. Esta clase fachada asume que hay otra clase de servicio que resolverá estos problemas y hace uso de ellas.

A continuación, veremos a alto nivel la arquitectura de cada una de las escenas.

4.2.2. Arquitectura menú principal

Como se ha comentado en el apartado anterior, el menú principal está formado por secciones. En cada una de esas secciones se dispone de diferentes ajustes o enlaces.



En la jerarquía de GameObjects de la escena hay una cámara principal a la que se le han asignado los scripts para controlarla y un objeto Menú que es el que implementa todos los elementos referentes al menú principal.

El objeto Canvas[25] implementa la interfaz gráfica que se superpone delante de la cámara. De fondo de este HUD se ha utilizado una imagen estática y encima se ha colocado el objeto BookPro[26] importado de la librería de assets. Este objeto implementa toda la lógica relativa al menú en forma de libro e integra todos los elementos, botones, textos o elementos de interfaz.

Se ha decidido concentrar todos los elementos de interfaz en las páginas derechas, las páginas pares, y las impares se han dejado vacías o con algún elemento de decoración.

Por último, el objeto PlayerSelection que viajará entre escenas con las opciones escogidas por el usuario como se comentará más adelante.

Figura 17: Jerarquía gameobjects menú principal

En esta escena se encuentran los siguientes componentes o scripts para controlar la lógica:

- **SliderValueDisplay**: Este script es llamado cuando un slider es modificado y modifica el campo de texto para indicar al usuario el valor actual. No tiene dependencias con otros componentes, sólo es usado por los objetos que lo necesiten.

- **SettingsController**: Esta clase se encarga de guardar y cargar las preferencias de usuario. Es utilizada desde el menú para guardar la información y posteriormente desde todas las escenas para cargar dicha información. Además, los datos se guardan entre diferentes sesiones del usuario.

- **PlayerSelectionController**: En este componente se almacena información del número de jugadores es cogidos para la partida. Esta selección puede ser variable en cada iteración, por lo que no será necesario guardarla en las preferencias de usuario, simplemente debe viajar entre escenas.

- **MenuController**: Se encarga de cargar las diferentes escenas. Además, si es necesario, guardará las preferencias del usuario antes de navegar.

- **DisappearAnimationScript:** Este script se utiliza para realizar una transición suave entre escenas. Realiza una animación y posteriormente llama a la función correspondiente de MenuController.

- **CustomPageShadows**: Este componente se encarga de añadirle sobras al objeto del menú. Así se evita el efecto de estar flotando y no estar integrado en la escena.

- **BookController**: Se encarga de controlar la lógica del menú principal y de manejar la visualización de las distintas subsecciones. Hace uso de la clase BookPro del asset externo[27].

En la siguiente figura se ven las dependencias entre clases que tenemos. Este gráfico se ha generado haciendo uso del asset gratuito Script Dependency Visualizer[28].

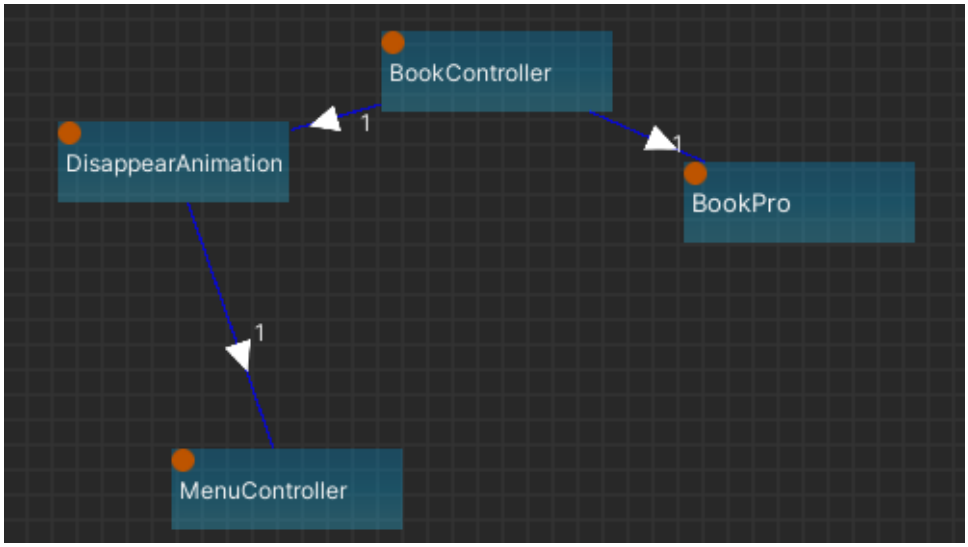
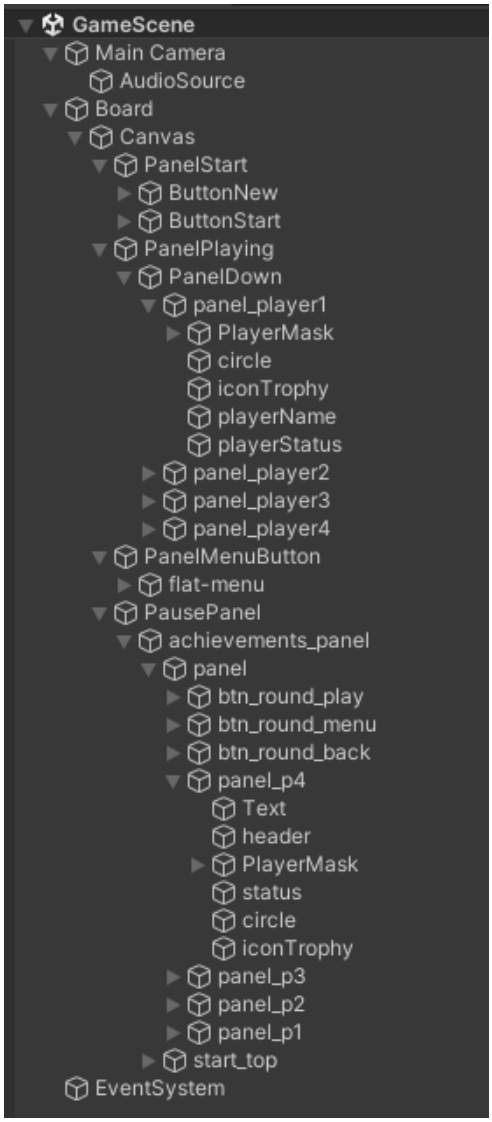


Figura 18: Diagrama dependencias entre clases escena menú principal

Como se observa, el controlador principal es BookController que actúa como fachada y hace uso de funciones de BookPro para cambiar la sección actual y funciones de DisappearAnimation para hacer una transición fluida antes de cargar otras escenas haciendo uso de MenuController.

4.2.3. Arquitectura juego



En la jerarquía de objetos de esta escena se tiene una cámara principal con sus scripts de control con un hijo Audio Source que se comentará en la sección de diseño de sonido, y un objeto Board que implementa toda la lógica del juego. Todos los objetos creados en esta escena serán hijos del GameObject Board.

Dentro de este último objeto se integra un Canvas para gestionar todos los elementos de la interfaz de usuario. Esta interfaz está formada por tres paneles, el primero que se visualiza al comenzar el juego, un segundo durante el juego y un tercero que aparece cuando el usuario pause o finalice la partida. Simultáneamente solo se visualiza uno de estos tres dependiendo del estado del juego, cuando el panel de pausa aparece se congela la partida y se inhabilitan el resto de los paneles.

El panel de inicio contiene los botones de inicio y de nuevo circuito, el de juego la interfaz con la información de los usuarios y el panel de pausa dispone de la información de pausa y botones para cambio de escena o cierre de panel.

Figura 19: Jerarquía gameobjects escena juego

En la escena de juego las principales clases son las siguientes:

- **GridDrawer**: Este script se encarga de dibujar los elementos necesarios para simular que se está jugando sobre una libreta cuadriculada. Es un script para la interfaz gráfica que dibuja una serie de líneas verticales y horizontales separados una unidad. En él se define el tamaño máximo del papel, el tamaño de dichas líneas y la profundidad que deben tener para estar debajo del resto de elementos.

- **CameraController**: Se encarga del control de la cámara, permitiendo hacer zoom o desplazarla. Este script es usado tanto por el usuario a partir de gestos o interacciones a través de ratón y teclado como por el GameController para hacer zoom y desplazarse de forma automáca cuando se requiera de alguna acción por parte del usuario en algún punto del tablero.

- La clase **GameController** que será la encargada de gestionar todos los elementos de la partida. En esta clase de manejan los turnos, el jugador actual que puede realizar un movimiento, maneja el movimiento de este jugador al hacer click, detecta si ha chocado o atravesado un checkpoint y gestiona el final de la partida. Para ello hace uso de la clase Track, la clase Player y la clase IAFunctions para realizar los movimientos automáticos de la IA.

- La clase principal anterior hace uso de la clase **Track** que maneja los elementos del circuito como son los puntos que lo forman o los puntos de control. Además, se encarga de invocar a las funciones de generación del circuito haciendo uso de TrackFunctions. De la misma manera, cuando haya que realizar algún cálculo sobre el circuito será la encargada de invocar con los parámetros adecuados.

- **TrackFunctions:** es la clase encargada de realizar todas las operaciones o algoritmos matemáticos necesarios para generar el circuito o hacer cálculos sobre el mismo como colisiones. Esta clase no conoce las características de los circuitos, siempre es usada recibiéndolos parámetros sobre los que debe realizar los cálculos.

- **Player:** se encarga de almacenar el estado de cada jugador, su posición, su penalización en caso de tenerla, los puntos de control restantes y si ha finalizado la partida o no. GameController crea y maneja tantas clases Player como jugadores haya en la partida.

- **IAFuncions:** es la clase que realiza los cálculos de búsqueda del siguiente movimiento óptimo, utilizando algoritmos de grafos para hacer la búsqueda.

- **PauseController**: Se encarga de manejar el menú de pausa y de final de juego. Dibuja el estado de la partida con todos los jugadores y permite salir o reiniciar la partida.

- **SettingsController**: Como ya se ha comentado en el apartado de arquitectura menú principal, esta clase se utiliza para cargar las preferencias del usuario que guardó en el menú.

A continuación, se muestran dos diagramas para entender la jerarquía y las dependencias entre las clases. Se observa que la clase principal GameController que actúa de fachada y hace uso del resto de clases para la lógica de la partida.

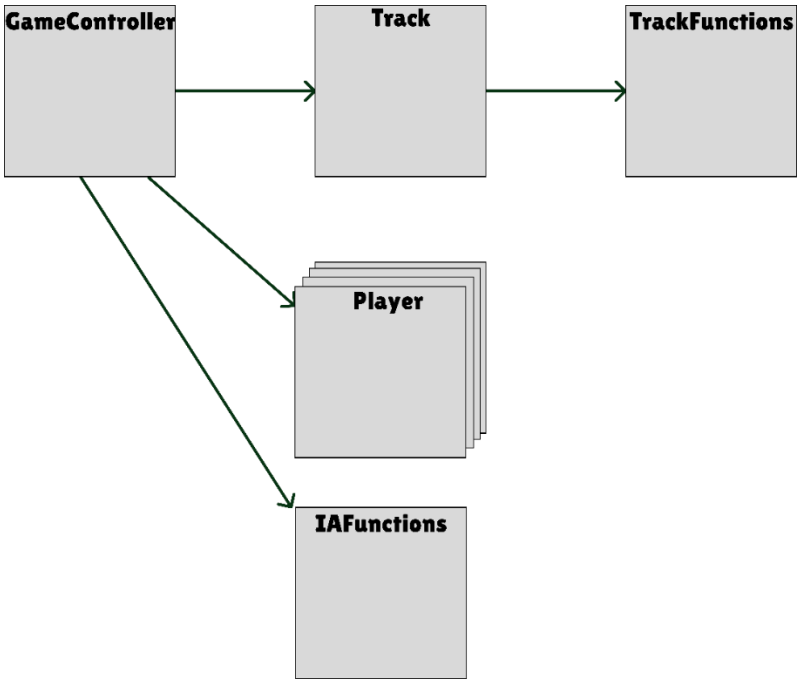


Figura 20: Diagrama de clases escena de juego

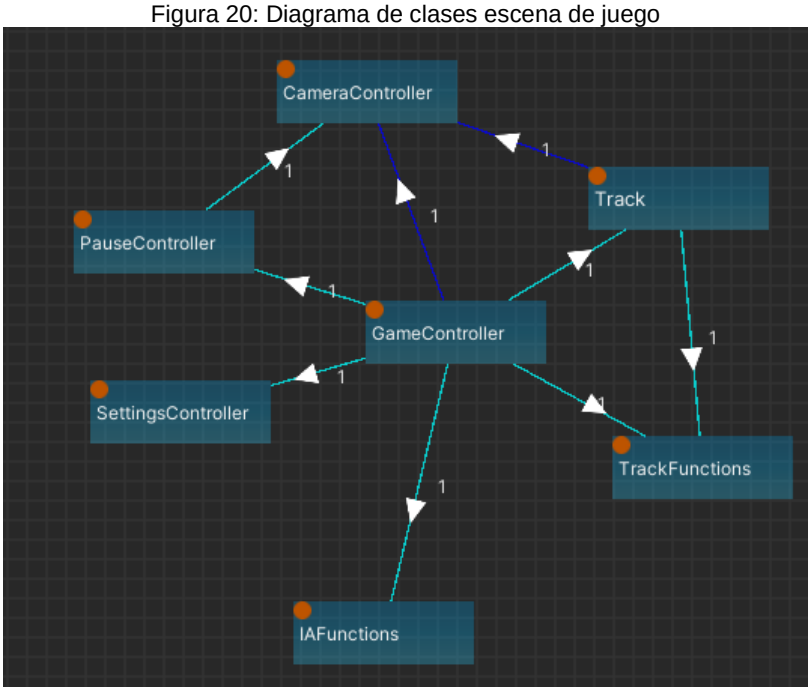


Figura 21: Dependencias entre clases escena de juego

4.2.4. Arquitectura tutorial

La arquitectura del tutorial es similar a la implementada en la escena del juego. Sin embargo, la clase que hace de fachada en este caso es OnboardingController y la clase Track ha sido sustituida por una OnboardingTrack que genera un circuito predefinido.

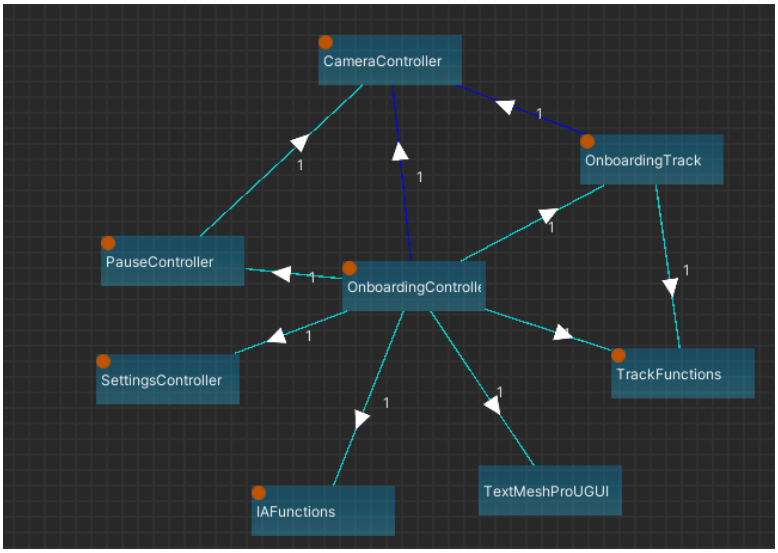
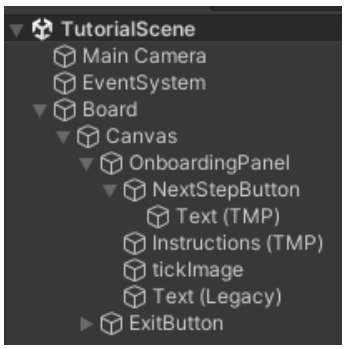


Figura 22: Dependencias entre clases escena de tutorial

En cuanto a la jerarquía de objetos se observa que en este caso han desaparecido los paneles de inicio de juego y de pausa, y se ha creado un nuevo panel para mostrar los textos con las instrucciones que el usuario debe realizar. Dentro de este panel se encuentra también un botón para los pasos en los que el usuario deba leer un texto.



El resto de los elementos son similares en esta escena a la de juego ya que por todo lo demás se trata de una partida similar.

Figura 23: Dependencias entre clases escena de juego

Se van a comentar las diferencias entre esta escena y la anterior:

- **OnboardingController**: Esta clase será la encargada de manejar el tutorial. Se quiere mostrar al usuario unas instrucciones y pedirle que realice una acción. Así iremos guiando al usuario para completar el tutorial. En este caso solo vamos a tener un jugador, sin IA y al obligándole en un primero momento, a realizar unos movimientos forzosos y posteriormente se le dejará libertad para completar el circuito.

- **OnboardingTrack**: Es similar a la clase Track que perteneciente a la escena de juego, pero en este caso, siempre crea un trazado predefinido. De esta manera el tutorial siempre se realiza sobre un circuito controlado.

El resto de las clases utilizadas son similares entre ambas escenas.

4.3. Diseño gráfico e interfaces

Una de las decisiones que se han tomado a nivel de diseño es siempre utilizar la orientación horizontal. De esta manera es más sencillo adaptar tanto a ordenadores, consolas como dispositivos móviles. Por lo tanto, todos los elementos del juego han sido diseñados con esta orientación, desde los menús hasta los circuitos.

4.3.1. Diseño logo

Para el diseño del logo se ha realizado un brainstorming utilizando la herramienta DALL-E[29] con el prompt “logo with words RACETRACE where the letters 'R' and 'T' are designed as racing circuits drawn on a squared notebook, while the rest of the letters are handwritten in a childish style”. Esta herramienta nos ha proporcionado unas primeras versiones e ideas con las que poder trabajar.

A partir de las ideas generadas se han seleccionado en una primera iteración los diseños que nos han parecido más atractivos y se han editado para corregir artefactos extraños producidos por la IA. Finalmente se ha hecho una segunda selección eligiendo únicamente cuatro diseños, los marcados en verde en la siguiente figura.

Figura 24: Selección de ideas generadas para el logo.

A partir de esta segunda discriminación, se ha decidido hacer una selección de ideas de varios de ellos y se ha creado en Figma[30] y Gimp[31] una versión final para el logo, quedando de la siguiente forma:

Figura 25: Logo definitivo.

A partir de este logo se ha diseñado también el icono y el splash del juego.

4.3.2. Diseño menús

El estilo buscado para el proyecto es un aspecto visual que simule estar jugando en una libreta con un con un bolígrafo. Se pretendía tener un acabado sencillo y visualmente parecer un juego de niños. Por lo tanto, como inspiración del apartado artístico se han utilizado juegos con aspecto infantil como *Candy Crush*[32] o *Clash Royale*[33].

Figura 26: Menús Candy Crash

En los menús del Candy Crash se observa que se utilizan colores pastel de fondo y los botones del mismo color, pero con tono más fuerte. En este caso los botones son

redondeados, con colores planos en los que encontramos una sombra interior en la parte inferior y un brillo en una esquina superior. Los botones aparentan ser un caramelo y los interruptores mantienen la misma estética. En la parte superior de cada menú encontramos una cabecera en forma de nube con una tipografía decorativa.

La organización de los menús son los típicos usados en juegos como servicio, donde aparece una subsección de juego clásico, una de eventos para mantener al usuario activo y otra de tienda para tentarlo de hacer microtransacciones. En este proyecto no se pretende hacer un juego como servicio, por lo tanto, se descartará la opción de incluir sección de tienda o eventos.

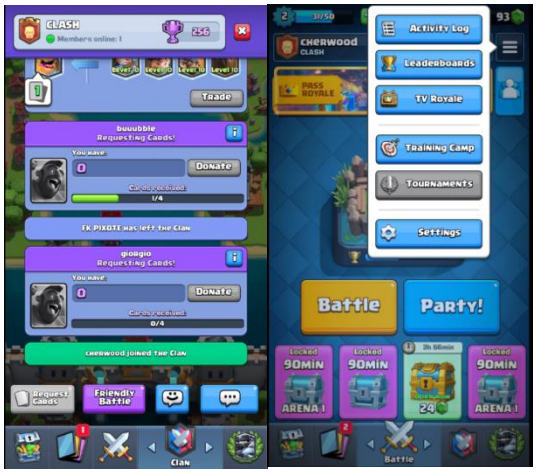


Figura 27: Menús Clash Royale

Los menús del Clash Royale tienen algunas similitudes con los anteriores. El estilo de los botones utiliza colores planos, con las esquinas ligeramente redondeadas, en la parte inferior una sombra y en una esquina superior un reflejo. Este juego es menos amigable con el nuevo usuario ya que se le ofrecen demasiados estímulos de forma simultánea. Además, cuenta con una estructura también de juego como servicio donde encontramos a primer nivel elementos para incentivar al jugador a volver a él mediante eventos o compras dentro del juego.

En este proyecto se han tomado los dos juegos anteriores como inspiración para realizar un juego con estética infantil. La versión final de los botones ha quedado de la siguiente forma. Se han utilizado las ideas de los juegos anteriores para crear unos botones similares:

Figura 28: Botones RaceTrace

Mediante un proceso iterativo se ha ido refinando la estética hasta conseguir el aspecto deseado. En la siguiente imagen se observa cómo ha ido evolucionando el menú principal hasta adquirir el aspecto final.

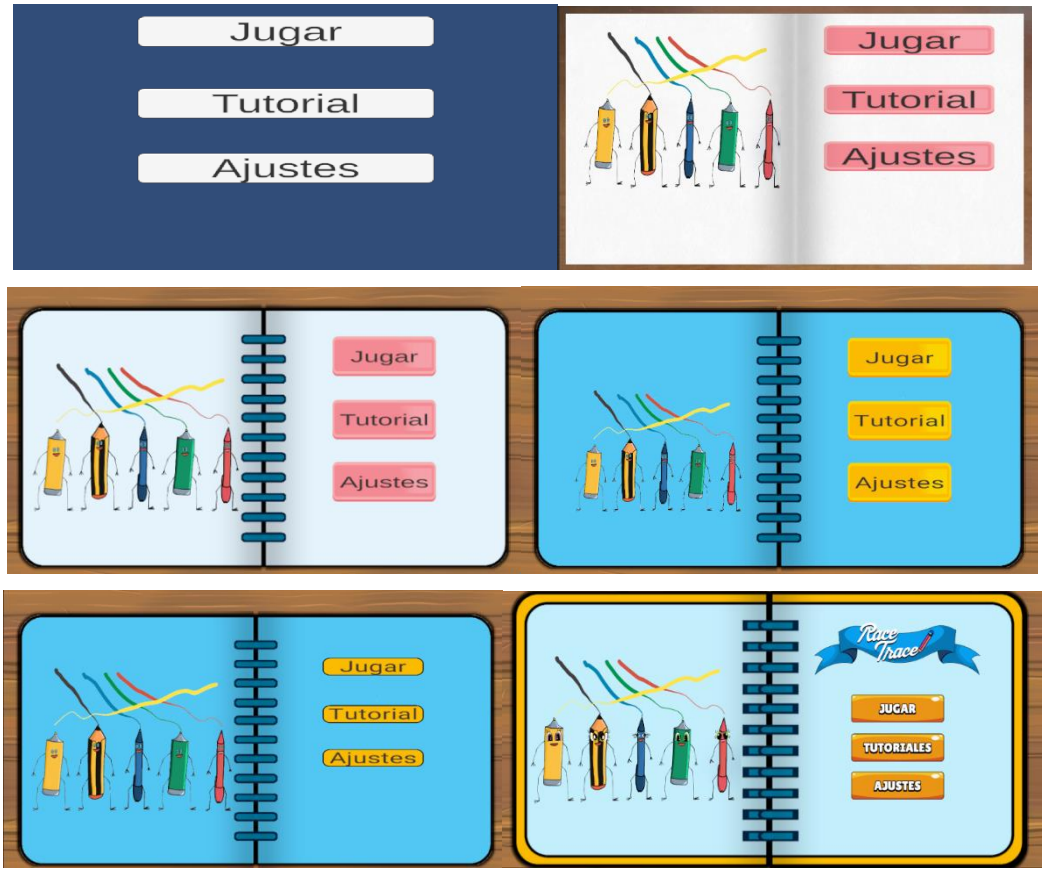


Figura 29: Iteraciones sobre el menú principal

En una primera versión se implementó la lógica del menú principal sin considerar ningún aspecto de diseño. Posteriormente se importaron los siguientes recursos de la tienda:

- "Cute Cartoon Mobile GUI - 97 png files!"[34]
- "Book - Page Curl Pro" [27]

Con estos dos recursos se implementó una segunda versión del menú simulando ser un libro y con unos botones ya creados. Posteriormente se iteró cambiando colores hasta llegar a un diseño artístico propio, sin usar los botones del paquete de recursos cartoon. Para las cabeceras de cada página se ha hecho un diseño inspirado en el pack de recursos anteriormente mencionado.

Los recursos propios que finalmente se han utilizado han sido los siguientes:

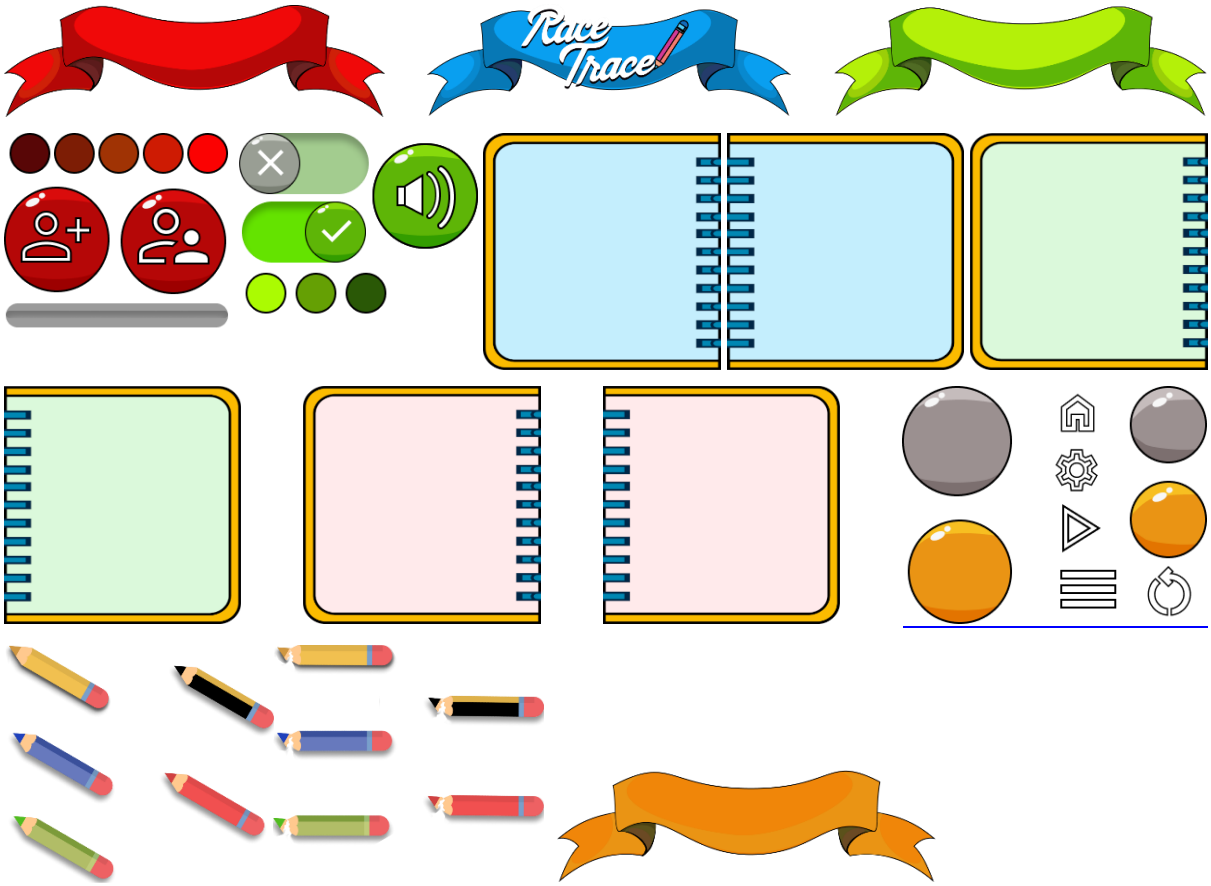


Figura 30: Recursos propios utilizados en los menús

Como fondo del menú se quería dar la apariencia de estar jugando en una mesa y se ha utilizado un recurso de “FREE Stylized PBR Textures Pack”[35]. De este pack de recursos se ha seleccionado una textura de madera con apariencia de dibujo a la que se le ha añadido una tinta roja y editado defectos hasta obtener la apariencia final deseada:

Figura 31: Fondo menú principal

Para la portada y la contraportada del libro que forma el menú principal se ha hecho una variación de la imagen del logo principal. Utilizando los recursos gráficos propios, el resultado final del menú obtenido es el siguiente:

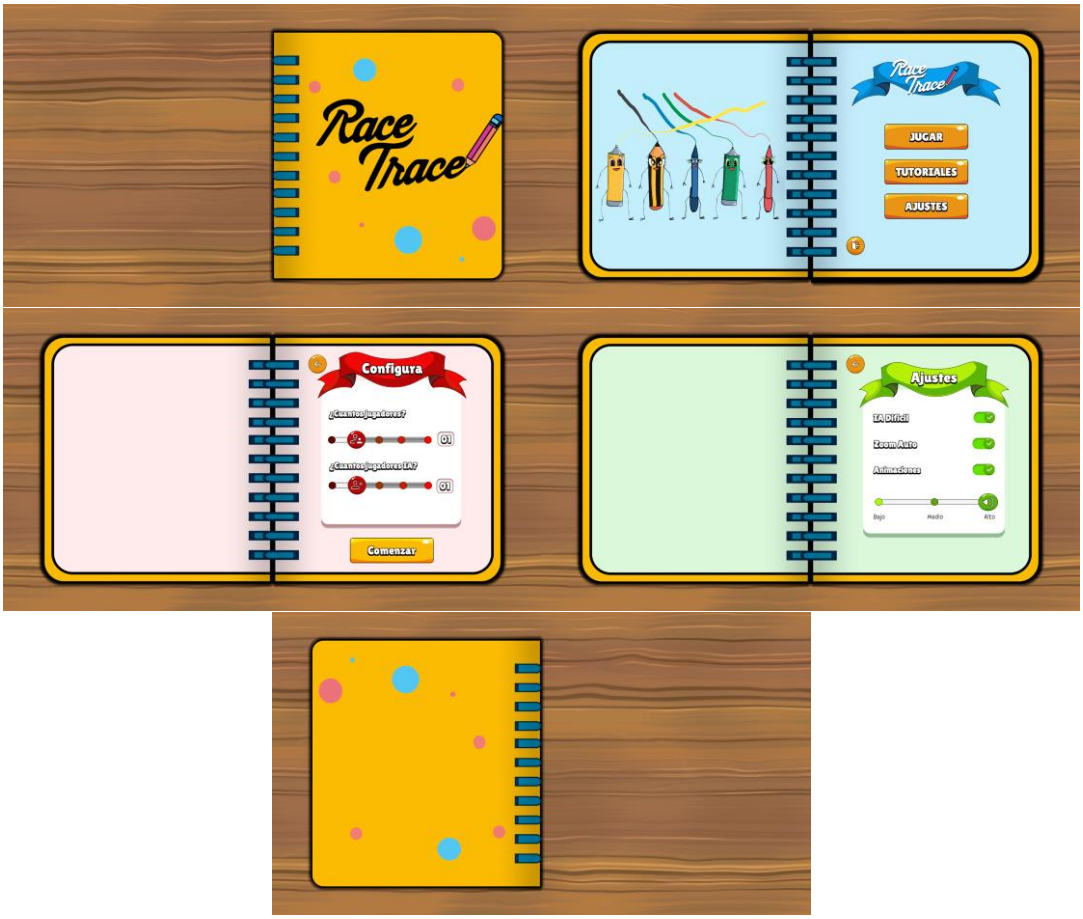


Figura 32: Menú definitivo

4.3.3. Diseño juego

Como se ha comentado, para el diseño del juego se quería conseguir un aspecto de estar jugando en una libreta con lapiz y bolígrafos. Por lo tanto, se han utilizado recursos propios para simular este efecto. En un apartado posterior se comentarán las animaciones generadas para dibujar y borrar el circuito.

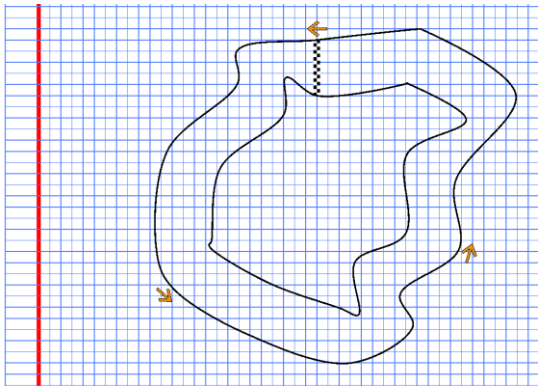


Figura 33: Diseño de circuito

Una vez dibujado el circuito se le permite al jugador elegir un circuito nuevo que borraría el actual y dibujar uno nuevo, o comenzar la partida. Al comenzar se situará el número de jugadores que el usuario haya escogido en la línea de salida, que coincide con la meta.

El jugador inicial se escoge de forma aleatoria entre todos los posibles. Si el turno actual es de un jugador controlado por la IA se calcula el movimiento que debe realizar, y si es un jugador humano se da el control para que elija la posición a la que se quiere mover. Durante la partida se hace un zoom a la posición donde está ocurriendo el movimiento actual y el jugador debe elegir la posición donde moverse.

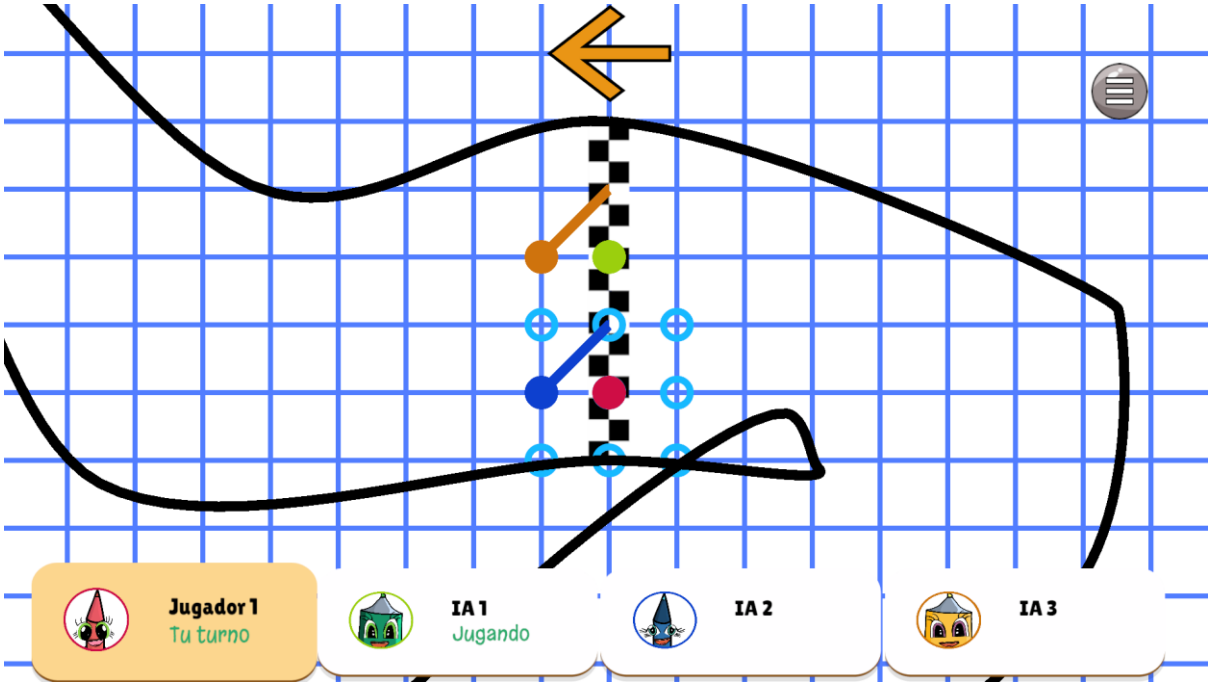


Figura 34: Turno del jugador humano

En la parte superior hay un botón con el que se puede abrir el menú de pausa. Este menú es el mismo con el que se abre al finalizar la partida y desde éste se puede salir al menú principal o comenzar otra partida.

Figura 35: Menú pausa y final de partida

4.3.4. Diseño de tutorial

Para el tutorial se ha tomado la base de una partida normal, pero con un circuito predefinido y únicamente un jugador. El jugador debe seguir instrucciones y realizar acciones para avanzar. Se ha decidido no hacer uso de pop-ups intrusivos que se le abran al jugador en cada paso y optar por integrar en la misma ventana la acción que el jugador debe realizar. En la parte superior se le muestra el texto con las instrucciones que debe realizar en cada paso.

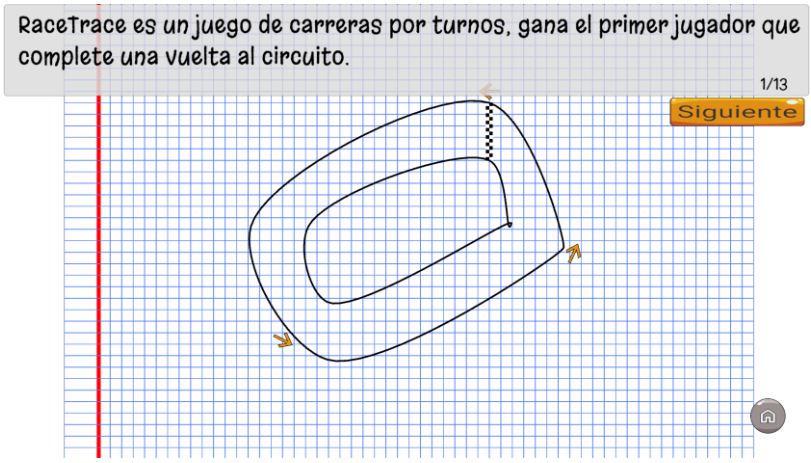


Figura 36: Tutorial juego

Se ha decidido predefinir un circuito sencillo rectangular que vaya introduciendo una curva poco gradualmente. De esta manera el jugador aprende como ir tomando las curvas y frenando, y si no lo hace se le indicará que hubiera sido sancionado al chocar. Los dos primeros turnos estará obligado a ir en línea recta y a partir del tercero ya se le deja libertad para moverse donde quiera. De esta manera se busca que el jugador entienda cómo funciona el movimiento en este juego. Además, se le explica los controles básicos como el zoom y el desplazamiento sobre el circuito. Tras completar una vuelta, el tutorial acabará.

4.3.5. Animaciones

En este proyecto se han implementado animaciones para aportar dinamismo a las escenas. El objetivo era que todo pareciera reactivo a las acciones del usuario y que no hubiera transiciones bruscas entre escenas o entre componentes.

Se ha optado por implementar dos tipos de animaciones, las que se han considerado estáticas siempre en el mismo punto y las dinámicas que dependen del circuito. Las animaciones estáticas se han implementado con el componente Animator[36], en el que se ha creado una máquina de estado en la que cuando se cumplan determinadas condiciones se active la animación.

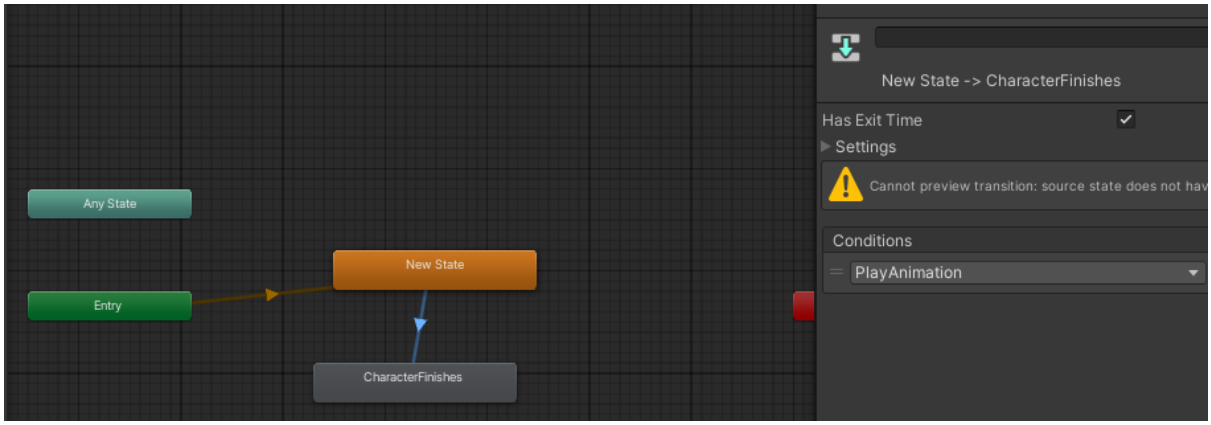


Figura 37: Animator personajes

Las animaciones dependientes de la forma del circuito se han implementado mediante scripts, creando corutinas [37] de Unity para procesarlas en paralelo en otro hilo de ejecución y no bloquear la interfaz de usuario.

Además, en todos los botones se ha activado un efecto de tinte para darle feedback al usuario cuando se ha situado sobre ellos y cuando los ha pulsado.

Figura 38: Animación pulsar un botón

Se van a detallar las animaciones en cada una de las escenas.

- Escena de menú principal:

  Al iniciar el juego aparece el libro en la portada y tras unos segundos se hace una animación para ir a la siguiente página con el menú principal. Todas las subsecciones del menú principal están en una página de un libro, y al cambiar de una a otra se hace una animación de girar la página. Además, el usuario puede manualmente cambiar de una página a otra si selecciona la esquina inferior de la página y desliza como si estuviera manipulando un libro.

  Al darle a tutorial o iniciar una partida se hace una animación de cerrar el libro hacia la contraportada y posteriormente se quita el libro de la vista. Esto se ha hecho para evitar una transición brusca entre escenas.

Figura 39: Animación cambio de página.

- Escena de juego:

  Al entrar a una partida o al tutorial se realiza una animación para dibujar o borrar el circuito. El circuito se va dibujando a medida que el lápiz recorre las líneas. Por otro lado, cuando el usuario elige un nuevo circuito aparece una goma que va borrando el circuito. Esta animación no es tan precisa como la anterior como comentaremos en capítulo de implementación, pero tiene una precisión suficiente para simular que se está borrando.

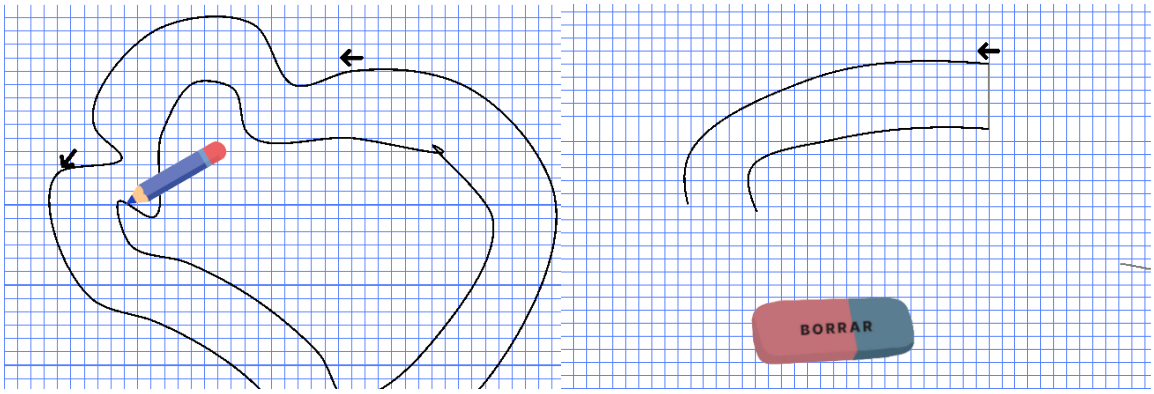


Figura 40: Animaciones dibujar y borrar circuito

  Cuando el usuario realiza un movimiento en su turno se ha implementado una animación similar a la de dibujado de circuito para pintar el trazo que recorre. Además, si el usuario choca con el borde del circuito se indica el punto de choque y se hace una animación de mostrar lápiz roto y se muestra un texto animado. Por último, cuando un jugador cruza la línea de meta se muestra el texto animado de la posición en la que ha quedado y mediante el componente Animator se activa la animación de su imagen.

  Cuando el jugador debe realizar alguna acción en pantalla se hace un zoom desde la posición actual hasta la posición final. De esta forma se le indica que movimiento debe hacer.

- Escena de tutorial:

  Las animaciones de esta escena son similares a las vistas en la escena de juego, sin embargo, en este caso no se le permite al jugador borrar el circuito que siempre está predefinido. En la parte superior hay un texto que indica el paso a realizar para

avanzar en el tutorial. Para mostrar este texto se realiza una animación de ir escribiendo letra a letra el texto y se consigue darle una señal visual al usuario para indicarle que la acción anterior ha sido finalizada con éxito. Además, mediante el componente Animator cuando finaliza un paso del tutorial se le muestra un icono verde creciente.

4.3.6. Fuentes de letra

En este proyecto se han utilizado distintas fuentes de letra para que tuviera un aspecto desenfadado y juvenil, utilizando tipografías decorativas[38] gratuitas de Google. Se han utilizado las siguientes fuentes:

- Margarine[39]: Proporciona un aspecto informal y la hemos utilizado para todos los textos largos.
- Lilita One[40]: Es una fuente que genera unas letras muy anchas y grandes. Se ha utilizado para botones y cabeceras de menús. El color escogido en este caso ha sido en blanco, añadiendo un borde negro en el contorno y una sombra hacia la parte inferior. Estos efectos han sido más sencillos de implementar en un componente Text (legacy), por lo que se ha descartado el uso del componente actualizado TextMeshPro.

Figura 41: Ejemplos de las fuentes utilizadas, Lilita One y Margarine.

4.3.7. Diseño de personajes

El desarrollo de personajes es un punto clave y muy importante en los videojuegos, Sin embargo, debido limitaciones en habilidades artísticas, se ha optado por realizar un diseño de personajes simple y funcional para centrar los esfuerzos y recursos en otras áreas como la programación, la lógica y el diseño de mecánicas de juego.

Se han diseñado cinco personajes, cada uno de ellos se corresponde con un instrumento de escritura que pinta de un color diferente. Cada jugador utilizará uno de estos personajes que se corresponderá con el color de su desplazamiento. Estos diseños son sencillos, con colores planos, una sombra y una luz. Como inspiración para los ojos se han utilizado diseños de la serie de dibujos animados “My Little Pony”[41], ya que son muy llamativos y saltones.

Se ha dibujado los personajes del juego utilizando la herramienta Concepts[42] para Ipad.

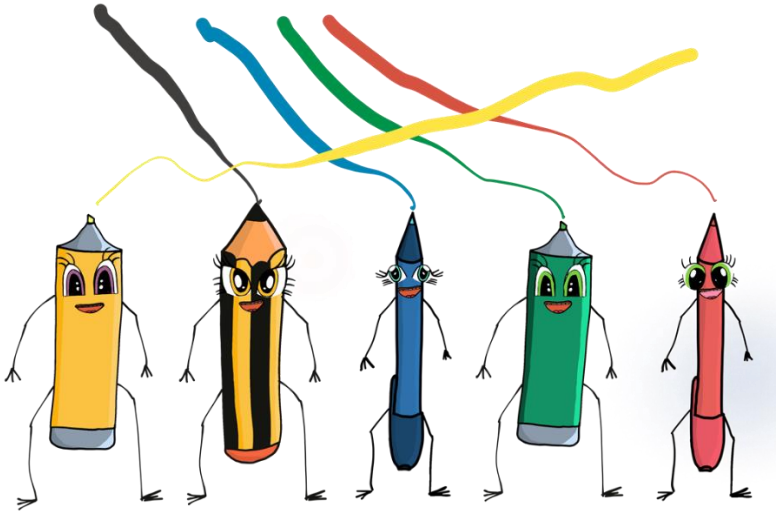


Figura 42: Personajes del juego

4.3.8. Usabilidad /UX

Como se ha comentado en el alcance, el objetivo de este TFG es optimizar el juego para teclado y ratón. Aunque se puede ejecutar desde un dispositivo móvil a través de un navegador, la precisión de las pantallas táctiles en ocasiones no es suficiente para jugar con comodidad salvo si su tamaño de ésta es muy grande. No obstante, se han estudiado otras posibles formas de interacción.

En los dispositivos móviles se suele mostrar un HUD[43] superpuesto donde el usuario debe hacer click, de esta forma se sustituyen los botones y se aumenta la precisión en las pantallas táctiles. Como se detallará en el capítulo de líneas de futuro, se podría superponer una interfaz con los movimientos que el usuario puede elegir.

Figura 43: Interfaz móvil GTA San Andreas

4.3.9. Diseño de sonido

En el proyecto se ha decidido no incluir ninguna música que pueda ser repetitiva para el jugador. No se ha incluido ningún clip de audio que se reproduzca durante todo el juego. Sin embargo, si se ha implementado un sistema de sonido que permita darle una realimentación al usuario cuando haga clic en los botones o realice acciones.

Los clips de audio se han descargado de la web https://freesound.org/ y se han buscado con licencia
Creative Commons 0 para poder usarlos de forma libre y modificar para usos comerciales sin permiso del autor. Posteriormente se han editado a través del programa gratuito Audacity con el que se han recortado y se han eliminado los silencios para ajustarlos a las necesidades del proyecto.

Se han utilizado los clips para los siguientes elementos:

- Giro de página[44]
- Click de botón[45]
- Choque de jugador[46]
- Cruce de meta[47]
- Dibujar circuito[48]
- Borrar circuito[49]
- Paso tutorial[50]:(Finalmente descartado por ser repetitivo. Se ha reutilizado para los pasos del tutorial el mismo sonido que el de giro de página).
- Movimiento de jugador[51]

Se ha implementado un script PlaySound que al adjuntarlo como componente de un GameObject permite reproducir el archivo de sonido que tenga también asociado. Además, para evitar que los sonidos sean tan repetitivos para el usuario, se ha añadido una pequeña variación en el pitch cada vez que se reproduce un sonido. De esta manera si el mismo sonido es reproducido varias veces consecutivas no sonará de la misma forma y el usuario no tendrá una sensación de repetición.

Además, se ha implementado un sistema que da información al jugador a través del pitch[52]. Cuando el script esté asociado a un slider se irá aumentando el pitch y haciendo el sonido

más agudo a medida que se vaya desplazando el cursor hacia la parte superior del mismo y será más grave a medida que se vaya bajando.

Durante el juego se ha utilizado esta misma técnica para el movimiento del jugador, cada vez que un jugador se desplace se reproducirá un sonido que será más agudo cuanto más rápido se mueva el personaje. De esta manera el usuario puede asociar consciente o inconscientemente el sonido a una sensación de velocidad excesiva y necesidad de decelerar a través del sonido. De la misma manera, el clip de audio que se reproduce cuando un jugador llega a la línea de meta irá variando y haciéndose más grave según la posición en la que el jugador haya llegado a meta y podrá conocer la posición sin necesidad de consultar ningún elemento visual.

Otra de las decisiones de diseño que se ha tomado con respecto al sonido es bajar el volumen de los movimientos que realiza la inteligencia artificial para no resultar excesivamente molesto, únicamente los movimientos que realiza un jugador humano serán reproducidos con el volumen al completo.

El usuario puede bajar el volumen del juego a través de un slider en el apartado de ajustes en el menú principal. Se ha implementado un controlador que almacena el valor en una variable del sistema y lo aplica en los Audio Listeners de las escenas. De esta manera tenemos el volumen unificado en un mismo controlador.

Se ha implementado un GameObject en la parte superior de la jerarquía de objetos para contener un Audio Source genérico que permita reproducir los sonidos de los botones que cierren paneles. Se ha hecho de esta forma ya que un botón no puede reproducir un sonido si el panel que lo contiene ha sido ocultado. Al ocultarse el panel todos los objetos interiores se deshabilitan, incluido el objeto que contiene el botón. De esta manera cuando se interactúe con el botón, éste utilizará un Audio Source[53] que no va a ser deshabilitado y el sonido se reproduce sin problemas.

Además, al cargar escenas se ha introducido un pequeño retraso que permite hacer sonar el clip de realimentación del botón que ha hecho cargar la escena y no ser cortado de forma abrupta.

4.4. Lenguajes de programación y APIs utilizados

Durante el desarrollo del videojuego se han utilizado difentes lenguajes de programación y herramientas dependiendo de la fase. Se decidió realizar el primer prototipo Lo-Fi en Python por la rapidez de prototipado. Este lenguaje de programación permite realizar cambios y probarlos de manera inmediata ya que no requiere compilación, se interpreta el código durante la ejecución. Por lo tanto, se pudo probar la viabilidad de diferentes algoritmos antes de pasar a crear un proyecto real en Unity. Para la programación de este prototipo se utilizó PyCharm[54] como IDE.

Después se porteó el código a C# para ejecutarlo desde Unity. En este caso el IDE escogido fue Visual Studio Code[55] que permite editar el código en este lenguaje de forma muy sencilla. Este código es compilado y ejecutado por Unity.

Para la edición de audio se ha utilizado Audacity. Se han descargado los archivos de audio deseados y posteriormente se han editado para ajustarse a las necesidades del proyecto.

Para los prototipos de diseño se han utilizado varios programas y plataformas. Para el prototipado del apartado visual se ha usado la plataforma la herramienta en la nube “Figma” para crear gráficos vectoriales. Además, se han creado los diseños de los personajes con la aplicación Concepts que nos ha permitido diseñar también de forma vectorial los elementos del juego y finalmente han sido editados con Gimp.

En cuanto al hardware ha sido necesario un ordenador Steam Deck con el sistema operativo Windows 11 y una tablet Ipad con lápiz bluetooth que ha permitido hacer los diseños.

5. Implementación

En este capítulo se detallará la implementación de los distintos algoritmos y scripts, además, se explicará cómo han sido implementadas las partes más importantes de cada script y las decisiones relevantes que se han tomado. Además, se comentarán los principales errores encontrados y como han sido solucionados. Por último, se definirán los requisitos de instalación.

5.1. Implementación Menú

- **SliderValueDisplay**: Este script recibe dos componentes, un Slider y un Text. Cuando el slider es modificado se asigna su valor al text componente de texto. Se ha decidido usar el formato de dos dígitos, por lo que siempre formatearemos con “00” para añadir un 0 al valor.

- **SettingsController**: Como se ha comentado en el capítulo anterior, esta clase se encarga de guardar y cargar las preferencias de usuario. Hace uso de PlayerPrefs[56] de Unity que permite guardar la información en la caché del usuario y ser cargada. Para borrar esta información el usuario puede borrar las preferencias o la caché del navegador.

- **PlayerSelectionController**: Esta clase se necesita que viaje entre escenas y no se destruya ya que la información es necesaria para configurar la partida. Por lo tanto, en la función Awake se ha definido *DontDestroyOnLoad(gameObject)* para evitar que este GameObject sea destruido al cambiar a otra escena.

  Además, es necesario tener únicamente una instancia, no se quiere que al volver al menú principal se vuelva a crear otra vez este objeto. Si se tuviera varios no se sabría cuales son las opciones que se deben utilizar, por lo tanto, se ha definido como *public static PlayerSelectionController Instance* y en el awake se comprueba si ya existe una instacia de este tipo. Si al ser creado el nuevo objeto encuentra que ya existía uno anterior, se destruye este nuevo. Así se asegura de que únicamente se tiene uno.

- **MenuController**: Para cargar las diferentes escenas se hace uso de la función de SceneManager LoadScene[57] indicando el nombre a la que se quiere ir. Antes de cargar la escena de juego se guardan los valores de número de usuarios que haya escogido el usuario. Para ello se obtienen los valores de los sliders y se introducen en el objeto que controla el número de usuarios y viajará a la nueva escena (PlayerSelectionController). Se ha centralizado la carga de escenas en un único script para que sea más fácil de mantener y añadir nuevas escenas.

- **DisappearAnimationScript**: Este script se encarga de hacer una animación antes de cargar una nueva escena para tener una transición más fluida. Se ejecutará tras la animación de cerrar el libro y lo hace desaparecer por el borde de la pantalla. Permite realizar diferentes animaciones dependiendo de la escena que se cargará, sin

embargo, en este caso por simplicidad, se ha implementado solo la animación de desaparecer a la izquierda.

Para ello se debe calcular hasta que posición se debe desplazar el objeto a partir del punto de vista de la cámara. Según la relación de aspecto del dispositivo, la cámara tiene una mayor o menor amplitud, por lo tanto, si se tiene que calcular la visión. Se utiliza la función ViewportToWorldPoint[58] que transforma la posición de la cámara a la posición del mundo real.

Para hacer la animación de forma fluida se ha hecho uso de las funciones Lerp y SmoothStep que permiten hacer un movimiento no lineal y más natural[59][60]. Estas funciones se han utilizado para todas las animaciones donde haya un desplazamiento. Como se puede ver en la siguiente figura, el movimiento conseguido con SmoothStep comienza con una fase de inicial de aceleración, seguida por una fase central de movimiento constante y finalmente una parte de deceleración. Con esto se consigue un movimiento mucho más natural que si se hace un movimiento lineal.

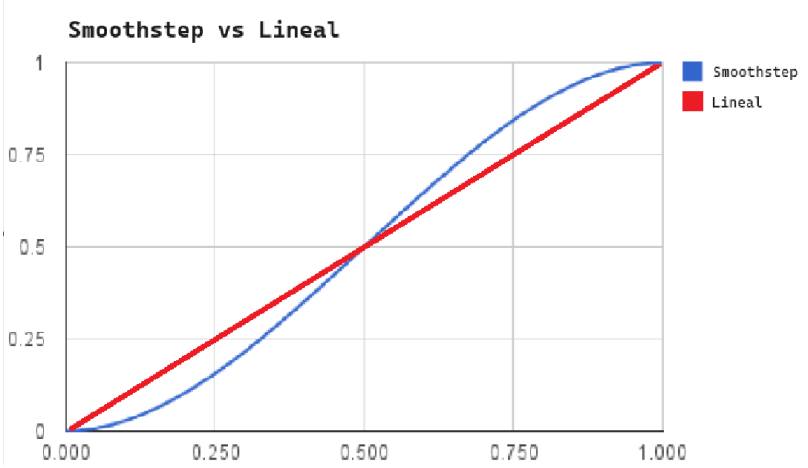


Figura 44: Movimiento lineal vs Smoothstep

Una vez finalizada la animación se hace la carga de la escena llamando a la función de MenuController.

- **CustomPageShadows**: En este componente se hace uso del componente de Unity Shadow para añadir una sombra en el objeto. Las sombras se añadirán dependiendo de en qué página se encuentren. Si se está visualizando la primera página se añadirán sombras en todas las direcciones, si se ven las páginas izquierdas no se añadirán sombras hacia la página derecha, pero si hacia las otras tres direcciones y para las páginas derechas, al contrario, no se añadirán sombras hacia la izquierda.

  Así se crea un efecto sencillo de sombra sobre el objeto libro que hace que no parezca que está flotando. El inconveniente que se tiene es que este efecto no es visible desde el editor como si hubiera sido si hubieran añadido las sombras con componentes de unity en vez de con un script. A cambio el mantenimiento y modificación de todas las sombras es más sencillo ya que se puede cambiar únicamente tocando un único script y sin tener que modificar tantos componentes como páginas y sombras haya.

- **BookController**: Este componente está basado en los ejemplos que se proporcionan con los assets del libro. Se definen las características que se quieren para el libro y las animaciones del mismo. En el script se han añadido las páginas en las que se tienen las diferentes subsecciones de menú y se utilizan esas funciones para ir de forma automática cuando el usuario haga click en los enlaces.

  Al iniciar el juego el libro estará cerrado mostrando la portada y tras unos segundos se abrirá y mostrará la primera página donde está el menú principal. También se han definido las funciones para activar las animaciones de quitar el libro para cambiar de escena. Esta se ejecutará tras finalizar la animación de cerrar libro, es decir, después de ir a la última página del mismo.

5.2. Implementación Juego

- **GridDrawer**: Este script se encarga de dibujar los elementos necesarios para simular que se está jugando sobre una libreta cuadriculada. Es un script para la interfaz gráfica que dibuja una serie de líneas verticales y horizontales separadas una unidad. En él se define el tamaño máximo del papel, el tamaño de dichas líneas y la profundidad que deben tener para estar debajo del resto de elementos.

  Se ha hecho uso de la función de UnityEngine LineRenderer[61] que nos permite dibujar una línea entre dos puntos en un espacio en tres dimensiones. En este caso al estar trabajando con vectores en dos dimensiones la posición Z la se mantiene constante y siempre más alejada de la cámara que el resto de los componentes para ser dibujadas por debajo.

  No obstante, LineRenderer se ejecuta en tiempo de ejecución y no en el editor de unity, por lo tanto, para poder visualizar como son las líneas y poder interactuar con el resto de GameObjects desde el editor se han implementado la función OnDrawGizmos[62] que permite dibujar en el editor, pero no en tiempo de ejecución. Haciendo uso de las funciones Lines[63] y Vertex3[64] de la librería de bajo nivel GL se han dibujado las líneas para visualizarlas en el editor de Unity.

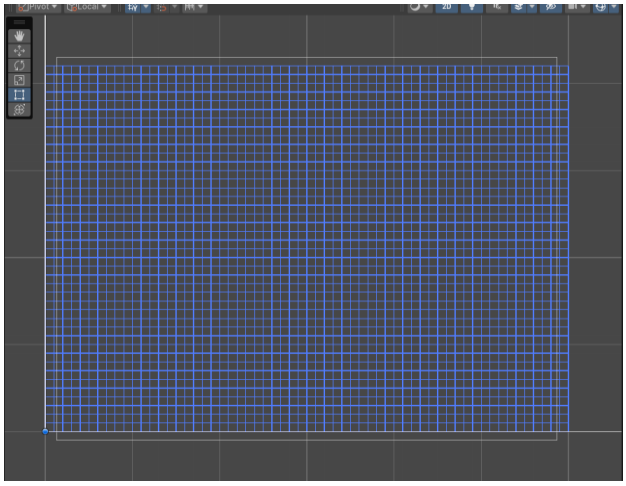


Figura 45: Líneas dibujadas en el editor con GL.

Aunque la librería GL se podría haber utilizado también para tiempo de ejecución, se ha decidido utilizar LineRenderer ya funciona como un componente de Unity y se puede implementar de forma más sencilla, creando GameObjects que posteriormente se pueden manipular con menor dificultad.

Se ha detectado que este script puede tener problemas de aliasing al utilizar pantallas de poca resolución. Cuando el tamaño de la línea a dibujar es menor a un píxel, el Z-Buffer[65][66] puede decidir que dicho píxel no debe ser pintado. Como se trata de un defecto gráfico menor, que no ocurrirá en los terminales con gran densidad de píxeles, y que no afecta a la jugabilidad, se ha decidido no invertir más tiempo en la resolución de este error gráfico para este TFG. En la siguiente imagen se puede apreciar cómo algunas líneas no son pintadas y no se ve una separación uniforme entre ellas.

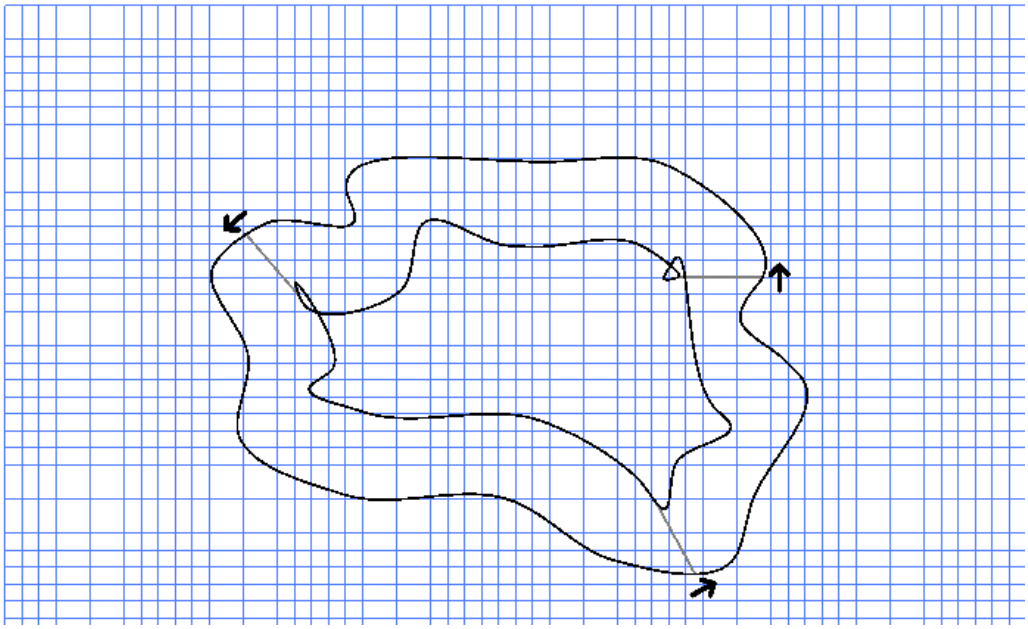


Figura 46: Efecto aliasing en las líneas.

Este script se utiliza para la interfaz gráfica y no tiene relación con el resto de lógica de la partida.

- **CameraController**: Este script implementa las funciones de hacer zoom y desplazarse hasta un punto. Puede ser llamado por el usuario al interactuar con el teclado y ratón o al hacer gestos con una pantalla táctil. Estas funciones pueden utilizarse desde otros scripts si se necesita que se desplace automáticamente a un punto, por ejemplo, cuando es el turno del jugador, se hace un zoom y un movimiento automático para mostrar el punto donde se encuentra el jugador y los posibles puntos de movimiento.

  El usuario puede con un teclado mover la cámara usando las flechas o arrastrando con un ratón, para ello se comprueba que han sido pulsadas las teclas o ha hecho click con el botón izquierdo. Con las flechas se hace un movimiento lineal dependiente del tiempo de pulsación. Sin embargo, con el ratón se calcula cuanto ha desplazado el usuario el ratón para calcular el punto de inicio de la cámara y el punto final.
  Si el usuario dispone de una pantalla táctil se ha implementado también el control con gestos. Primero se debe comprobar cuantos dedos tiene el usuario sobre la pantalla. Si el usuario tiene un dedo es porque quiere hacer un movimiento de cámara y si tiene dos es porque quiere hacer un zoom. Para el movimiento se debe ver cuál ha sido el punto de inicio donde el usuario ha pulsado y hasta donde se ha desplazado, con esto se sabe cuánto se debe mover la cámara. Por otro lado, si el usuario tiene dos dedos en la pantalla se hará un zoom dependiendo de si acerca o aleja los dedos.

  Se detectó que el editor de unity interpretaba estos pellizcos de forma inversa a la compilación WebGL, por lo que se tuvo que invertir el sentido del zoom en el caso de estar en el editor de unity:

```
#if UNITY_EDITOR
      difference = -difference;
#endif
```

  Por otro lado, cuando el usuario estaba haciendo un gesto de zoom y soltar primero uno de los dos dedos se producía un desplazamiento no deseado de la cámara ya que se interpretaba que el usuario había hecho click en un punto y soltado en otro, por lo tanto, cuando el usuario esté haciendo zoom no se permite hacer un desplazamiento hasta que no haya soltado ambos dedos. Con eso, se evita que la cámara realice este desplazamiento incómodo.

Cuando se quiere hacer zoom desde otro script se le debe indicar que dos posiciones se quiere mostrar en la cámara, habitualmente será el punto de inicio del movimiento y los posibles puntos finales. Con estos dos puntos se calcula el punto central donde se debe colocar la cámara y se calcula el zoom a partir de la diferencia de estos puntos sumando un margen exterior. Para este desplazamiento se van a usar corutinas, para ejecutar en paralelo y no bloquear el resto de las acciones del usuario, y funciones para hacer animaciones no lineales más fluidas (SmoothStep).

Se ha definido un máximo y un mínimo de zoom y de posición de cámara, por lo que siempre se usará la función Clamp[67] para no exceder de estos límites.

- **GameController**: Esta clase se encarga de manejar la partida, los turnos y la finalización del juego. Al iniciar genera tantos jugadores como se indique en la clase PlayerSelectionController que no habrá sido destruida al cargar esta escena. Al correr directamente en el editor la escena de juego no se tiene esta clase, por lo que se ha asignado un valor por defecto para hacer pruebas.

  También se encarga de la visibilidad de los paneles en función del estado de la partida. Al comienzo se mostrará el panel de inicio de juego y durante la partida éste se ocultará y se mostrará el panel de juego.

  Esta clase, como se ha comentado en la arquitectura de las clases, actúa de fachada y confía en que habrá otras clases especializadas que habrán generado un circuito. Cuando el usuario haga click en comenzar partida se inicializa la partida.

  El turno inicial se asigna aleatoriamente entre todos los jugadores y va rotando entre el resto. Cuando el turno sea de un jugador humano se pintarán los posibles puntos donde el jugador se puede mover y se esperará a que haga click en uno de ellos.

  Cuando el turno sea de un jugador controlado por la IA se llamará a las funciones encargadas de calcular el movimiento. Esta función responde en pocos milisegundos y para simular que está pensando se ha añadido un corutina para que espere un tiempo aleatorio.

Otra de las funciones de esta clase es comprobar si un jugador ya ha acabado. A estos jugadores se les muestra como que ya han acabado y su posición y no se les deja jugar, el turno pasa automáticamente al siguiente jugador. Si todos los jugadores han acabado mostraremos el panel de fin de partida, que es el mismo panel de pausa.

Además, se encarga de comprobar si un jugador está sancionado por haber chocado con el borde del circuito, si es así, le quitará un turno de sanción, pero no le dejará jugar y pasará el turno al siguiente.

En la siguiente figura se detalla cómo es la secuencia de turno en forma de máquina de estados. Este script se encarga de manejar esta máquina de estados hasta que todos los jugadores han finalizado su partida.

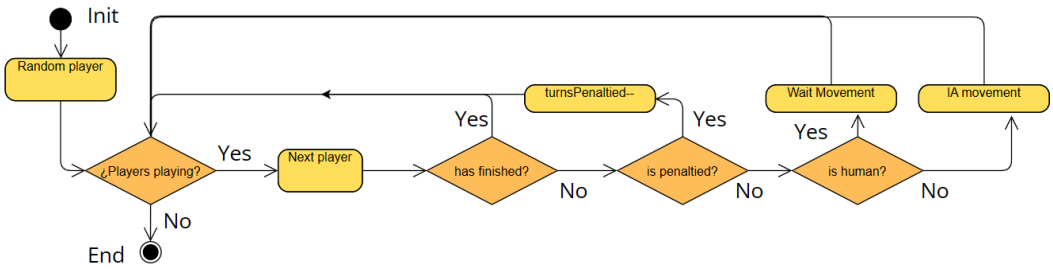


Figura 47: Secuencia de turnos de una partida

Cuando el jugador ha finalizado la partida se lanza la función de final de partida, mostrando una animación y abriendo el panel de final de partida.

- **Track** esta es una clase sencilla que únicamente se encarga de guardar las componentes del circuito y hacer de proxy frente a las funciones de generación de circuito por parte del controlador de juego. De esta forma se independiza la fachada de la implementación que haya por debajo.

  Aquí se tiene toda la configuración necesaria para generar un circuito, el tamaño, número de puntos aleatorios que se quieren, parámetros de dificultad usados para generar polinomios, anchura del circuito, número de puntos de interpolación entre dos puntos o tipo de interpolación que se va a usar en los algoritmos. Además, se guardarán los vectores de puntos exteriores, interiores, centrales, los puntos de control y los puntos iniciales una vez se genere un circuito. Será necesario también guardar los GameObjects que forman un trazado para tener una referencia a ellos si se requiere borrarlos.

Los circuitos se forman por tres listas de vectores bidimensionales (X, Y). Cada circuito tiene una lista de puntos con las posiciones del borde del circuito exterior, las del borde inferior y las del centro del circuito. Este script se encarga de implementar la lógica de negocio y orquestar las llamadas a las funciones matemáticas delegadas al script TrackFunctions, que se detallará a continuación.

Al comenzar la partida se llama a la función de generar circuito, se obtienen las componentes de trazado, y llama a la corutina para dibujar el circuito con una animación. En paralelo invoca al controlador de cámara para centrarla y mostrar en el circuito recién calculado completo. Cuando el usuario presiona sobre el botón de generar nuevo circuito se lanza corutuna con la animación de borrado y posteriormente se repiten todos los pasos anteriores.

Cuando el controlador de juego necesite saber cualquier dato sobre el circuito llamará a esta clase que es la encargada de guardarlos. El resto de los métodos envuelven a la clase de cálculos matemáticos, pero haciendo uso de los parámetros de este circuito en concreto. Por ejemplo, cuando el controlador de juego necesita saber si hay colisión entre dos puntos simplemente indica el punto inicial y el final, y se abstrae de como sea el circuito realmente. Esta clase, que es la que conoce el circuito, se encargará de enviar a calcular la colisión a partir de los dos puntos y de los parámetros del circuito.

- **TrackFunctions**: Es la clase más compleja, su función es realizar las operaciones o algoritmos matemáticos necesarios para generar el circuito o para hacer cálculos sobre él. La generación de circuitos se va a detallar en un apartado posterior. Además, se encarga de hacer las siguientes operaciones.

  Se encarga de dibujar los circuitos. Crea una corutina para ir dibujando el circuito de manera progresiva, utilizando funciones de suavizado para que el trazo sea fluido. Crea un LineRenderer y en cada frame calcula el punto del circuito por donde la animación va dibujando. Cada vez que atraviesa uno de los puntos que forman el circuito se añade a un LineRenderer que va creciendo. Así se simula que el circuito se genera poco a poco. Esto se repetirá para la línea exterior del circuito y la interior.

Otra de las funciones es la animación de borrado de circuito. Para esta animación, se calcula el punto inferior izquierdo y superior derecho y se traza una línea que une ambos. Este será el desplazamiento que haga la goma para recorrer todo el circuito. Igualmente, la goma recorriendo la línea perpendicular hasta llegar a los extremos mínimos y máximos. En la siguiente figura se recrea uno de los posibles movimientos que realizaría la goma. En la práctica el número de perpendiculares que recorre es mayor para cubrir todo el espacio. La línea roja es la diagonal que une ambos extremos y la línea verde el recorrido de la goma en el tiempo.

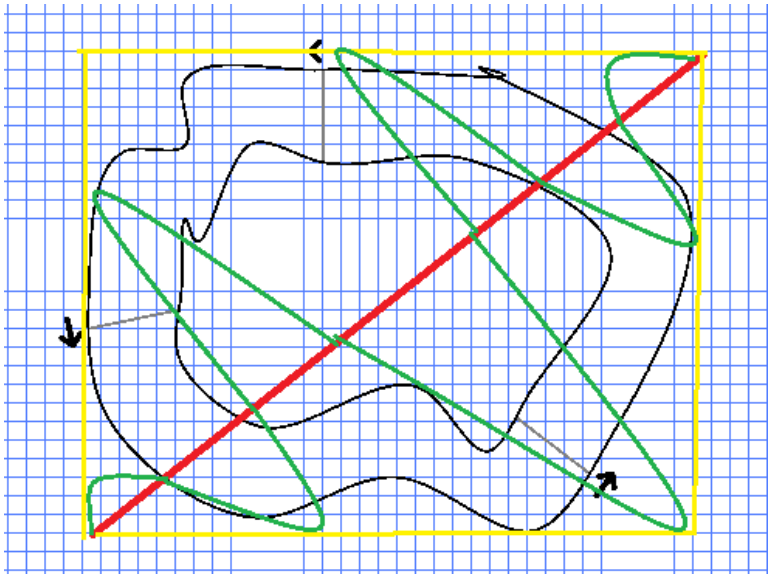


Figura 48: Animación de borrado de circuito

En cada frame se calculan las colisiones con los GameObjects que dibujan el circuito. Si se tiene una colisión con una de las flechas se elimina completamente. Sin embargo, si hay una colisión con los LineRenderer que formal el circuito se calcula el punto dentro del objeto donde hay colisión y se borra desde ese punto hasta el final del vector.

Este borrado de LineRenderer no es totalmente preciso, pero es suficientemente bueno. Para mejorar la precisión de esta animación se requeriría crear nuevos LineRenderer por cada colisión para tener líneas separadas. Debido a la gran complejidad de esa variante, se ha decidido utilizar la animación simplificada.

Otra de las funciones es el cálculo de colisiones. Para ello se han implementado algoritmos matemáticos para el cálculo de cruce entre dos vectores[68] [69] [70].

De la misma manera, se ha implementado el algoritmo ray-casting[71][72] que permite calcular si un punto está dentro o fuera de un polígono. Esta función se usa dos veces para calcular si está dentro de la línea exterior del circuito y fuera de la línea interior y así saber si el punto está dentro del circuito.

Otra de las funciones importantes es la búsqueda del punto dentro del circuito más cercano donde poder colocar al jugador. Esta función se utiliza cuando se produce un choque. El algoritmo comienza buscando el punto más cercano y comprueba si está dentro del circuito con el anterior algoritmo, si no está dentro debemos recorrer siguientes puntos más cercanos, sumando y restando uno a las coordenadas X e Y y comprobando. Así se va aumentando la distancia hasta encontrar un punto dentro del circuito. En ocasiones los circuitos tienen un lazo en su trazado y se debe evitar que el jugador se coloque en él porque no se podría salir, por lo tanto, para evitar esto, se debe comprobar si existe un camino posible al que movernos en el siguiente turno, si no existe, lo descartamos.

Otra de las funciones importantes es el cálculo de si un movimiento es en la dirección correcta o incorrecta. Para ello de la lista de puntos que forman el centro del circuito se busca el más cercano de la posición inicial y el más cercano de la posición final, si el punto de la posición final es mayor que el de la inicial significa que se va en la dirección correcta. Aquí se debe tener cuidado al principio y al final del circuito ya que en se debe sumar o restar una vuelta al valor de los puntos.

También no encontramos funciones para calcular el tamaño del circuito, encontrar la menor distancia entre la línea exterior y la inferior, cálculo de ángulos de vectores y funciones para destruir GameObjects.

El resto de las funciones de esta clase se utilizan para generar el circuito y debido a su gran complejidad y a ser uno de los pilares fundamentales del proyecto, se comentarán en un apartado posterior de la memoria.

- **Player**: Esta clase actúa prácticamente como data transfer object, se encarga de almacenar el estado de cada jugador y sus GameObjects. Todas las funciones que tiene son de modificación o consulta de datos y de creación de objetos.

Uno de los objetos que alberga es una lista con los puntos de control que le queda por cruzar. Estos puntos se inicializan al principio de la partida y cada vez que cruce uno, en orden, se elimina. Cuando al jugador ya no le queden checkpoints se considera que ha finalizado la vuelta completa. Se pretendía evitar que el usuario decida hacer la vuelta en sentido contrario y para eso se inicializó el vector con al menos tres puntos de control, y así se asegura que el jugador no completa la vuelta en sentido contrario. Si el jugador decide ir en sentido contrario se cruzará primero con el último punto de control sin contar la vuelta y al no ser el esperado no se eliminará del vector. Hasta que no cruce el primero de los esperados no se eliminará.

Por defecto en C# las listas se pasan entre funciones y clases como referencia[73]. Hay que tener cuidado e inicializar los puntos de control con una copia del vector y no con el vector original. Si se inicializa con el vector original calculado en la clase Track se estará asignando a todos los jugadores las mismas referencias a los checkpoints y cuando uno de ellos lo cruce se le borrará al resto de los participantes.

- **IAFuncions**: Esta clase se encarga de calcular el punto donde se va a mover la IA. Es llamada desde el controlador de juego. Al ser otra de las más complejas e importantes del juego, se comentará en detalle en un apartado posterior.

- **PauseController**: Este script activa el panel de pausa y final de juego donde se muestra información del estado de la partida. Lo primero que hace es detener el tiempo para que las animaciones no sigan calculándose. Al volver al juego se cierra el panel y se vuelve a activar el tiempo.

  En los paneles centrales se muestran los jugadores por orden de finalización de la carrera o por orden de aparición si no han acabado.

- **SettingsController**: Esta clase ya se ha comentado en la anterior escena y en este caso se encarga de cargar las preferencias del jugador. Se comprueba si la preferencia existe y se carga su valor del PlayerPrefs.

5.3. Implementación generación de circuito

Para la generación de circuitos se ha implementado el algoritmo[74] que se puede ver en la siguiente figura.

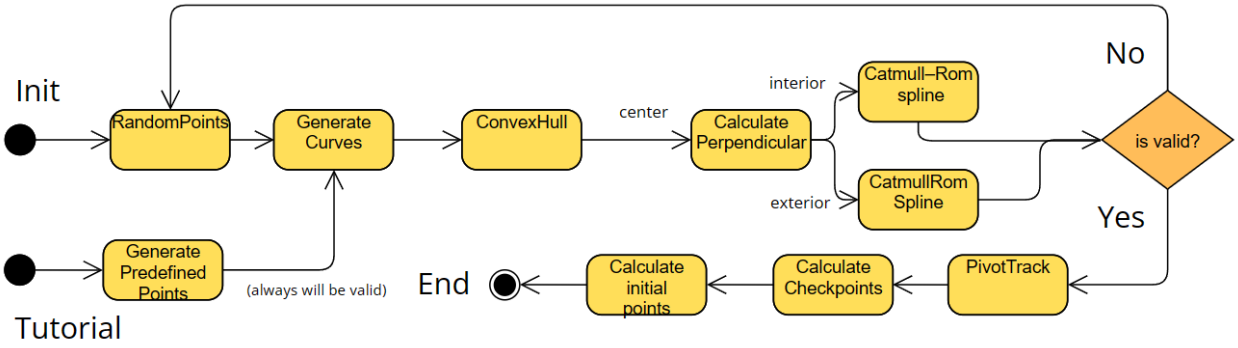


Figura 49: Flujo algoritmo generación de circuitos

Inicialmente se generan un número aleatorio entre 10 y 20 puntos bidimensionales y sobre estos se aplica el algoritmo Convex Hull[75] para encontrar los puntos exteriores. Esto genera un polígono convexo, es decir, la suma de ángulos interiores es de 180º. Sin embargo, este no sería un circuito muy atractivo. Para solucionar esto y conseguir con polígono cóncavo que tenga curvas interiores se busca el punto medio en cada segmento y a ese punto se le sumará un vector aleatorio. Así se evitan las rectas demasiado largas. Sin embargo, si los puntos están demasiado juntos no calcularemos el punto central ya que generaría curvas demasiado pronunciadas. En la siguiente figura se pueden ver los puntos aleatorios generados en amarillo, el contorno encontrado dibujado con rectas rojas y los puntos intermedios a los que se le ha sumado un vector aleatorio en verde.

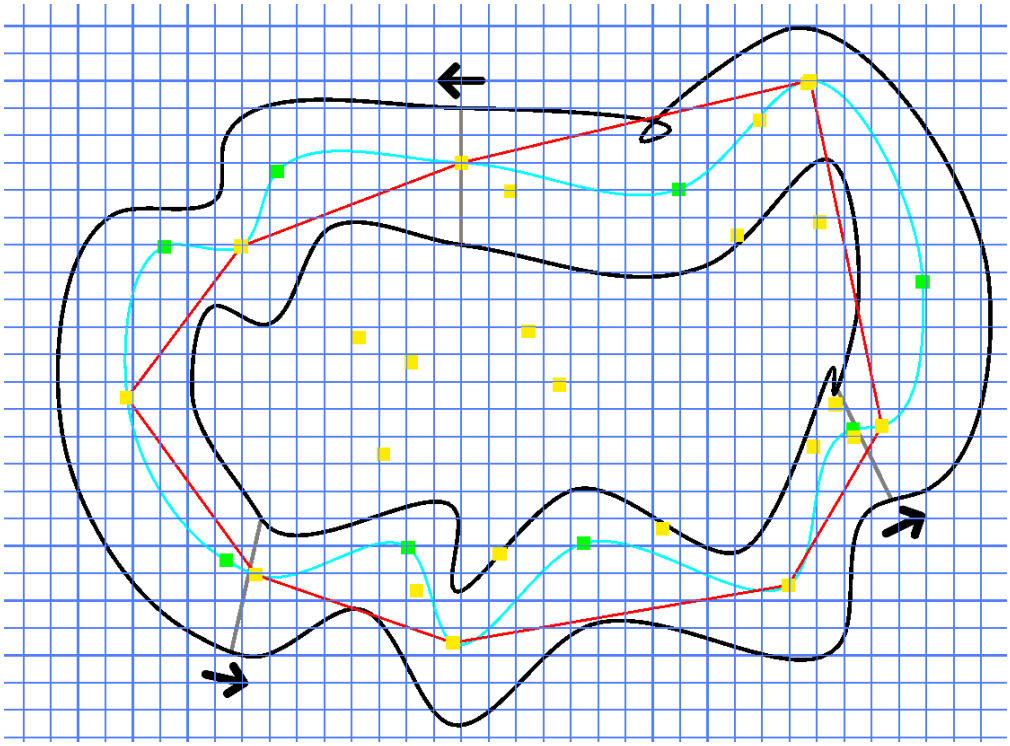


Figura 50: Componentes de generación de circuito

Una vez ya tenemos los puntos que van a formar el centro de nuestro circuito debemos aplicar el algoritmo Catmull–Rom spline[76] que nos permite de forma recursiva generar una curva que pasa por nuestros puntos de control. Aplicando este algoritmo conseguimos el polígono dibujado en azul.

El siguiente paso es calcular en cada punto la perpendicular hacia fuera y hacia dentro, para que se pueden generar nuevos puntos de control para la parte exterior e interior del circuito. Una vez tenemos estos nuevos puntos de control aplicamos de nuevo el algoritmo Catmull–Rom spline y obtenemos los polígonos dibujados en negro.

Una vez se ha obtenido un circuito se debe calcular si es un circuito jugable. Para ello se busca la distancia mínima ente la parte interior del circuito y la exterior. Si esta distancia es muy pequeña, en este caso se ha definido en dos unidades el mínimo, se descarta el circuito ya que sería demasiado difícil o imposible de completar. Además, se comprueba si el número de jugadores cabe dentro de la línea de salida del circuito. Si el circuito no ha sido válido se repite el algoritmo completo hasta encontrar uno válido, habitualmente en pocas iteraciones. Otra posibilidad hubiera sido encontrar los puntos donde la distancia fuera demasiado cercana y alejarlos, sin embargo, la complejidad era mucho mayor y se descartó esta opción.

En este momento ya se tiene un circuito correcto que se va a procesar. Primero se va a centrar el circuito en el centro del tablero, y se va a rotar para que siempre la componente horizontal sea mayor que la vertical, ya que es la orientación para la que se ha diseñado el juego.

Por último, se debe hacer coincidir los puntos iniciales y la meta con puntos válidos donde el jugador se pueda colocar (números enteros), así que esto también se ha considerado en la anterior rotación y desplazamiento. La figura anterior ha sido dibujada tras hacer la rotación y desplazamiento de todas las componentes.

Uno de los parámetros que utiliza el algoritmo Catmull–Rom spline es el parámetro alfa[77]. Este parámetro permite variar ligeramente como se calcula la interpolación entre los cuatro puntos de control. Es posible variar entre una interpolación centrípeta, uniforme o chordal. En la siguiente figura se observa cómo varía la generación del polígono dependiendo de este parámetro. En este proyecto se ha escogido usar una interpolación centrípeta (0.5) ya que es la que genera menos lazos.

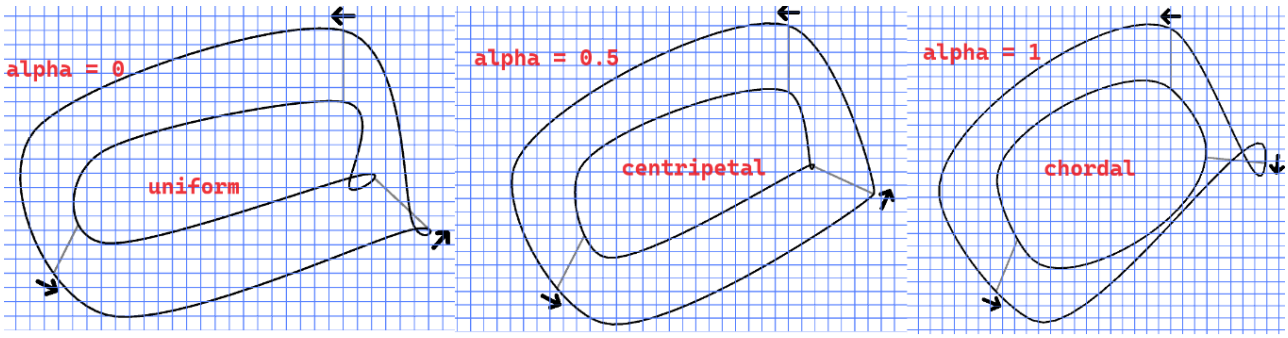


Figura 51: Algoritmo Catmull-Rom variando el parámetro alfa.

Con esto ya tenemos el circuito generado y devolvemos los parámetros en forma de lista de vectores bidimensionales que será guardado en la clase Track.

5.4. Implementación IA

La implementación de la inteligencia artificial se ha hecho en la clase IAFunctions. En ésta se implementa la función CalculateBestMoveIa que ha efectúa haciendo uso de la clase Threading.Task para ser ejecutada de forma asíncrona. Se ha hecho así para poder ejecutarse en un hilo separado del hilo principal y no bloquear la interfaz de usuario en caso de resultar demasiado pesada. El funcionamiento es el siguiente:

En un primer momento se obtienen los posibles puntos iniciales que se evalúan para ver si están dentro del circuito y si están en la dirección correcta haciendo uso de las funciones de TrackFunctions.

La función CalculatePointWillCrash calcula si un movimiento chocará en los siguientes turnos. Se crea un árbol de altura N, siendo N el máximo de velocidad entre la componente X o la componente Y, es decir, el número de turnos necesarios para quedarse parado. Para cada rama del árbol se calculan todos los posibles movimientos y si alguno de ellos no choca. Para todas las ramas que no choquen se repite hasta llegar a la altura N y al chocar se poda. Si se encuentra alguna rama sin podar con altura N significa que el jugador es capaz de quedarse parado antes de chocar. Esta función es muy pesada ya que tiene una complejidad ciclomática alta. El tiempo de cálculo crece a medida que el jugador va más rápido por lo que se usará solo en los casos necesarios antes de devolver si un punto es óptimo.

A partir de los puntos iniciales y sabiendo de cada uno de ellos si va a chocar y si va en la dirección correcta ejecutamos el siguiente algoritmo para encontrar el punto óptimo:

- En una primera iteración se recorre solo los puntos que estén **dentro del circuito** y que vayan en la **dirección correcta**, descartando los demás. Todos estos puntos se ordenan según la velocidad. Se hace uso de la función anterior CalculatePointWillCrash en estos puntos ordenados, y si se encuentra alguno que no choque en los próximos turnos se devuelve como punto óptimo sin seguir calculando el resto.

- Si la anterior iteración no ha encontrado ningún punto óptimo, se va a recorrer los puntos ordenados por velocidad que **no choquen** y vayan en la **dirección incorrecta**. Esto se hace así para conseguir salir de curvas extrañas generadas en los circuitos. De la misma manera, si encontramos un punto que no choque en los próximos turnos lo se devolvéra como punto óptimo.

- Si aún no se encuentran puntos óptimos, se devolverá el punto **dentro del circuito** que vaya en la **dirección correcta**, pero con **menor velocidad**, para ser sancionados con menos turnos en un futuro al colisionar.

- Se repite lo mismo, pero con los puntos en la dirección incorrecta, devolviendo el de menor velocidad.

- Si no hay ninguno que no choque en el turno siguiente, se devolverá el punto con menor velocidad de todos.

Haciendo este algoritmo se consigue evitar chocar con los bordes del circuito en este o en los próximos turnos, y en caso de no poder evitar el choque, si nos han cerrado la trazada, hacerlo a la menor velocidad posible.

Uno de los puntos más importantes para el correcto funcionamiento del algoritmo es el cálculo de la velocidad y la definición de lo que se considera velocidad. La “inteligencia” de este algoritmo se centra en esta ordenación ya que si se consigue hacerlo de la forma correcta siempre encontrará el punto de movimiento óptimo.

En una primera iteración del algoritmo se consideró calcular la velocidad a partir del desplazamiento en las coordenadas X e Y. Al hacerlo de esta forma el algoritmo priorizaba hacer diagonales sobre el circuito por delante del avance real sobre el mismo. Cuando el usuario selecciona la dificultad fácil se utiliza este tipo de ordenación.

En una segunda iteración se implementó una ordenación basada en el punto central del circuito más cercano, es decir, cuanto más alto sea ese punto se considera que se está avanzando más.

En la siguiente imagen se tiene ver una comparativa entre ambos tipos de ordenación y su trazada. La trazada verde usa el tipo de ordenación basado en la primera y la roja la segunda. El jugador rojo acabó en tres turnos menos.

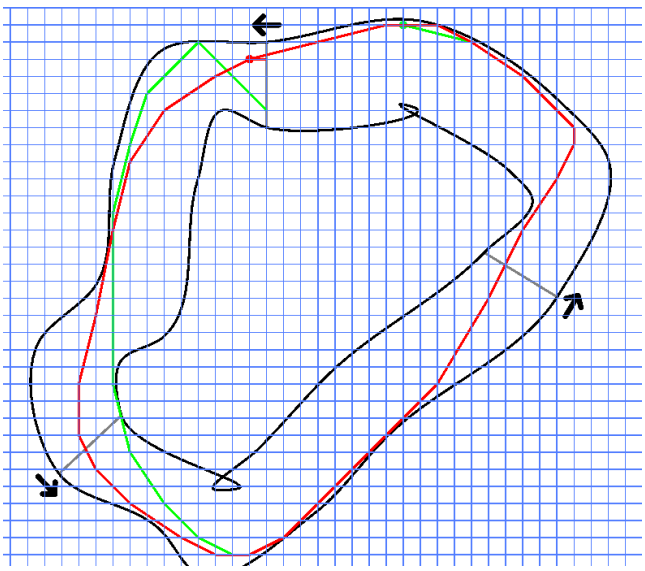


Figura 52: Comparativa IA dependiendo del tipo de ordenación

Una posible mejora no implementada sería encontrar el punto central del circuito más cercano, igual que se hace en la segunda ordenación, pero dentro de N turnos y quedarse con el que permita llegar más lejos. De esta manera se puede calcular la mejor trazada para las curvas.

5.5. Implementación tutorial

La prioridad de la implementación del tuturial era que fuera escalable y fácilmente mantenible, pudiendo añadir, modificar o borrar nuevos pasos de una forma sencilla. Para ello se desarrolló una máquina de estados en la que en cada paso se le pide al jugador una acción, se espera a que la realice y ese momento se pasa al siguiente estado hasta que no haya más pasos. En la siguiente figura se puede ver un diagrama del funcionamiento de esta máquina de estados.

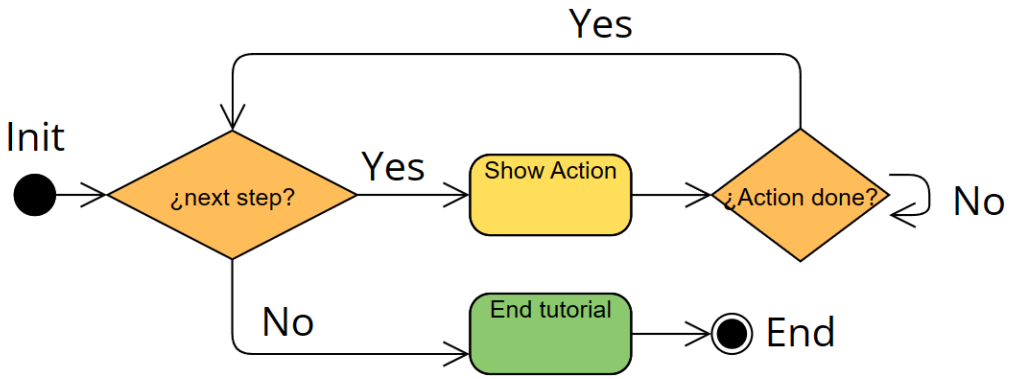


Figura 53: Dependencias entre clases escena de tutorial

Se ha definido una clase TutorialStep que contiene el texto que se va a mostrar al usuario, el tipo de paso que el jugador debe realizar (si es pulsar un pulsar un botón o realizar una acción) y la especificación de la acción que debe hacer. Después, se ha creado una lista de pasos que se irán recorriendo. Dos de estos estados serían los siguientes:

```
tutorialSteps = new List<TutorialStep>        {
        new TutorialStep("¡Bienvenido al tutorial de RaceTrace! Haga clic en  el botón \"Siguiente\"
        para continuar.", StepType.Text),
        new TutorialStep("Quita zoom con la rueda del ratón o pellizcando sobre la pantalla táctil.",
        StepType.Action, SpecificAction.ZoomOut),
            ...
};
```

En el primero se le muestra al usuario un texto y se le pide que haga click en un botón mientras que en el segundo se le pide que haga la acción de hacer un zoom out. En cada frame se debe comprobar si la acción realizada se ha completado y si es así pasaremos al siguiente paso. Cuando ya no queden más pasos en la lista se concluirá el tutorial.

De esta manera se ha conseguido poder añadir nuevos pasos simplemente añadiendo un nuevo elemento en esta lista y en los enumerados StepType y SpecificAction. Después se debe implementar si se quiere alguna acción específica al comienzo del paso y como se debe comprobar que el paso ha sido finalizado.

Otra de las cosas que se ha implementado en el tutorial es la generación de un circuito a partir de unos puntos de control. De esta manera se ha predefinido un rectángulo para crear un circuito sencillo que no tenga complicación para el usuario del tutorial.

5.6. Requisitos de instalación

El juego ha sido compilado para WebGL y para Windows.

Si se quiere probar la versión para navegador, solo se necesita un dispositivo con acceso a internet y un navegador compatible con HTML5 para jugar al juego. El enlace al juego es el siguiente:
https://birrefringencia.itch.io/racetrace

La versión desarrollada para ordenadores requiere la versión Windows 7 o superior de 64 bits. Además debe ser compatible con DirectX 10 o posteriores[78].

6. Demostración

6.1. Instrucciones de uso

Como se ha indicado en el apartado anterior, el juego está disponible en la siguiente ubicación:
https://birrefringencia.itch.io/racetrace

No es necesario disponer de ninguna cuenta ya que está accesible a todos los que dispongan de esta url sin necesidad de cuenta en itch.io.

Si se quiere ejecutar la version para Windows se debe desargar el fichero, descomprimirlo y ejecutar RaceTrace.exe.

Para comenzar el juego se le debe dar al botón central “Run Game” que cargará el player de Unity. Una vez haya finalizado la carga inicial del juego se aterrizará en el menú principal.

Desde el menú principal se pueden cambiar los ajustes generales, ajustes de la partida, iniciar el tutorial o comenzar la partida con los ajustes seleccionados.

Si se escoge ir al tutorial se mostrará en la parte superior un texto que se debe leer para ir completando los pasos del tutorial. En cada paso se debe realizar una acción y una vez se completan todos se habrá finalizado el tutorial.

Si se quiere jugar una partida se seleccionará “Jugar” en el menú principal y se escogerán las opciones de la partida. Al hacer clic sobre comenzar se cargará la partida y se pintará un circuito que si no le gusta al jugador puede cambiar por otro. Una vez escogido el circuito con el botón “Comenzar” se podrá jugar la partida.

Se debe recorrer el circuito para completar una vuelta completa procurando no chocar contra los laterales del trazado o el jugador será sancionado. El primer jugador en completar una vuelta será el ganador.

6.2. Prototipos

Para este proyecto se creó un prototipo Lo-Fi desarrollado en Python y luego otro en Unity que sirvió de base para el proyecto final.

6.2.1. Prototipos Lo-Fi

Se puede ver una demostración del primer prototipo programado en Python el siguiente enlace:
https://www.youtube.com/watch?v=g2YR7LqrvBQ

En este prototipo se estudiaron los diferentes algoritmos matemáticos necesarios para la generación de un circuito de forma procedural y posibles soluciones para implementar una inteligencia artificial.

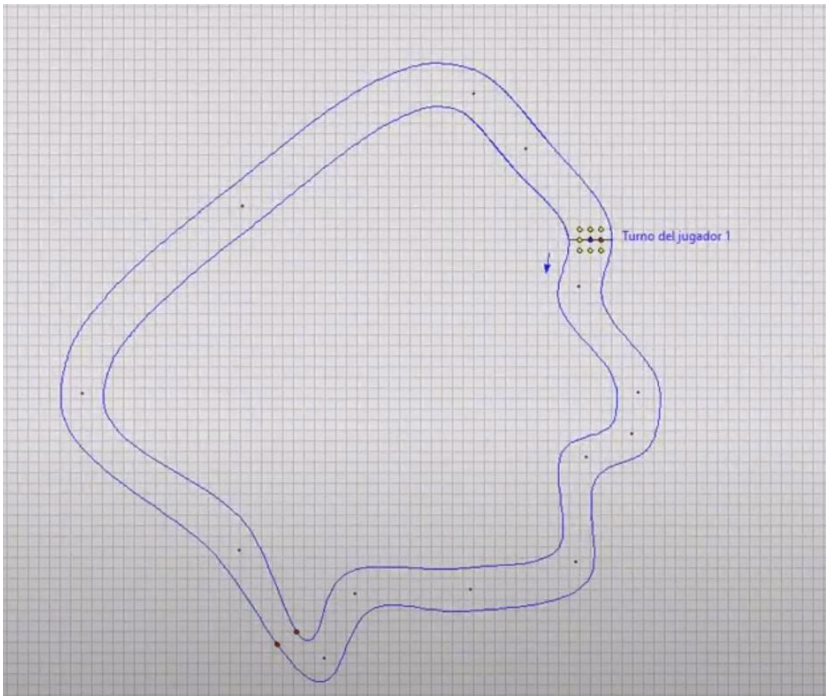


Figura 54: Imagen prototipo desarrollado en Python

Además de este prototipo, se ha diseñado un wireframe usando la herramienta online Figma, donde se han definido los flujos de las pantallas y se han generado los gráficos vectoriales.

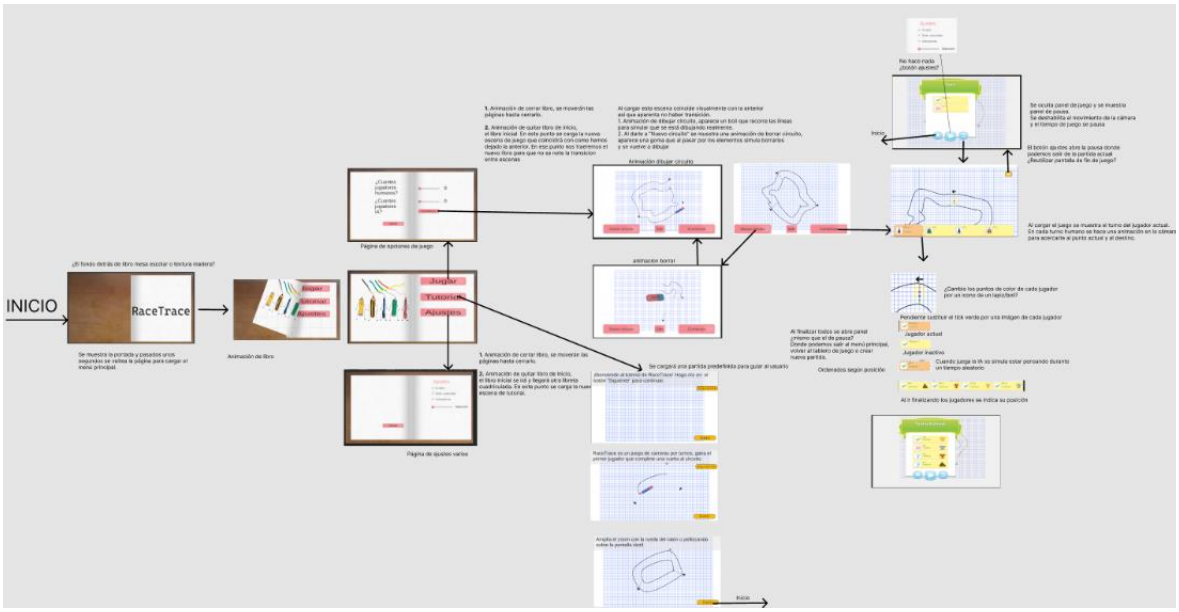


Figura 55: Wireframe del proyecto.

6.2.2. Prototipos Hi-Fi

No se han diseñado prototipos de alta fidelidad, únicamente se ha comenzado el desarrollo de un prototipo funcional, pero sin ningún tipo de consideración en el diseño y sin ningún tipo de estilo artístico hasta que se ha definido el estilo visual definitivo.

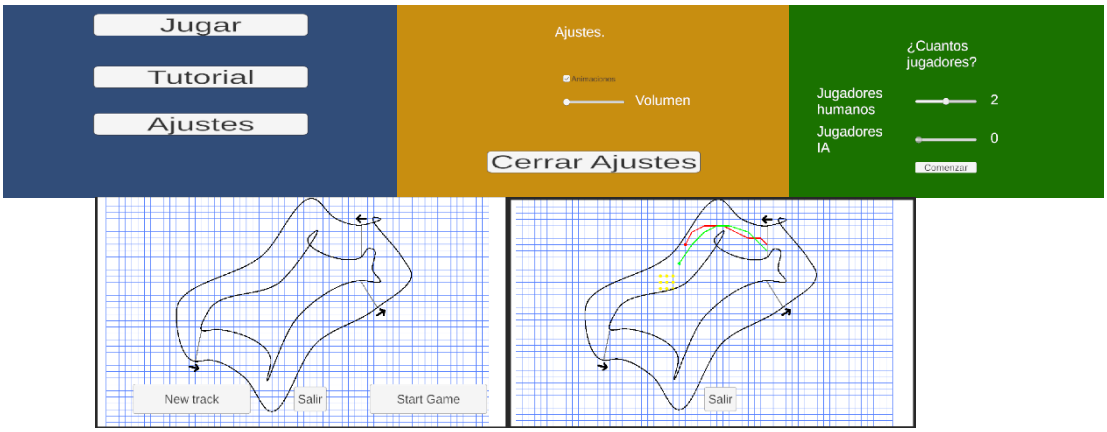


Figura 56: Prototipo en Unity

6.3. Tests

En este proyecto no se han desarrollado tests unitarios, se ha preferido realizar tests integrados con toda la funcionalidad desde el editor de Unity. Además, se han utilizado los logs que proporciona esta herramienta para poder ir haciendo debug del código.

Por otro lado, se han definido todos los parámetros configurables de los scripts como públicos o privados con la propiedad SerializeField para poder ser modificados o consultar su valor en tiempo de ejecución. De esta manera ha sido sencillo ir corrigiendo errores y comportamientos extraños en el código.

Además, se ha realizado un test de usabilidad con personas externas en los que usuarios reales han probado la interfaz para identificar problemas y áreas de mejora en la experiencia de usuario. Se ha observado como los usuarios interactuaban con la interfaz y se han identificado los errores que cometen, las dificultades encontradas. A partir de esta información crucial, se ha optimizado la usabilidad para garantizar que el producto final es sencillo de utilizar.

7. Conclusiones y líneas de futuro

7.1. Conclusiones

Este trabajo me ha resultado una experiencia muy positiva y enriquecedora. He comprobado la dificultad de enfrentarse al desarrollo de un proyecto complejo como es un videojuego desde cero y las grandes complicaciones que siempre surgen. Además, he tenido que ir adaptando el alcance y prioridad de tareas a medida que han ido surgiendo imprevistos y problemas para poder resolverlos de forma ágil, pivotando para ir cumpliendo los plazos de entrega con un producto funcional.

Este juego me ha supuesto un reto por la gran complejidad matemática de los algoritmos implementados, pero, he comprendido la importancia de la documentación de la comunidad y del uso las matemáticas en la resolución de problemas complejos.

Otro de los puntos que me ha parecido relevante es la importancia del diseño para crear un producto atractivo al usuario. Ha sido importante el feedback de amigos y familiares al enseñarles el diseño de los menús o juegos y sus comentarios, ya que, el desarrollador pierde un poco la perspectiva global. Estoy contento con el acabado visual conseguido en el proyecto. Sin embargo, me hubiera gustado llegar más allá en el diseño de personajes. No obstante, debido a mis dificultades a la hora de dibujar a mano no he sido capaz de crear los diseños con el nivel deseado.

Continuando con los recursos en línea, me ha sorprendido la gran cantidad documentación y assets disponibles en la tienda de Unity que permiten ahorrar mucho tiempo al desarrollador. La compra del asset para animar el libro fue una pieza fundamental en el desarrollo del menú y se consiguió el aspecto artístico esperado.

Me hubiera gustado desarrollar la versión para dispositivos móviles, sin embargo, debido al corto periodo que dura este TFG, se despriorizó esta tarea para poder darle más peso al diseño, que requirió mucho más tiempo del inicialmente estimado. No obstante, el juego es perfectamente jugable en dispositivos móviles a través del navegador.

La metodología ágil ha resultado ser un gran acierto para el desarrollo de este proyecto ya que ha permitido adaptar la priorización de las tareas a los imprevistos y cambios de

necesidades. Además, ha permitido construir ir testando el producto y haciendo cambios para refinar la jugabilidad a medida que se iba desarrollando.

A nivel personal estos cuatro meses de duro trabajo me han servido para mejorar mis conocimientos de programación, y más concretamente con el lenguaje C#. Por otro lado, he sido capaz de concebir un proyecto completo, desde el diseño y planificación hasta el desarrollo y las pruebas. He sido capaz de tener una visión global y actuar con todos los roles de desarrollo de software. Este juego, me ha permitido valorar mucho más el trabajo de otros desarrolladores profesionales de la industria, se capaz de tener una menta analítica a la hora de jugar a un juego, y plantearme como sería yo capaz cada uno de los elementos gráficos, artísticos o de scripting. Creo que me permitirá disfrutar y valorar mucho más los videojuegos y del trabajo que hay detrás de ellos.

7.2. Líneas de futuro

Partiendo del trabajo realizado en este proyecto, es posible ampliarlo de varias maneras.

Por un lado, se puede implementar un nivel de dificultad mayor para la inteligencia artificial. En el proyecto hemos considerado hacer el avance máximo en el turno actual, pero sin tener en cuenta las consecuencias de los turnos siguientes. El resultado es que no toma la trazada óptima ya que no es capaz de predecir como serán los siguientes movimientos más allá de evitar chocar. Si se considerara como velocidad la posición del jugador dentro de un determinado número de turnos, sería capaz de optimizar la trazada en las curvas, colocándose en la mejor posición para realizar los futuros desplazamientos. Esta mayor visión del comportamiento a futuro hará que los movimientos de la inteligencia artificial más adecuados y sean más desafiantes para el jugador.

Otro de los puntos que ha quedado pendiente es la creación de una versión para dispositivos móviles. Es necesario hacer un testeo exhaustivo en pantallas táctiles y considerar si es necesario adaptar los controles a estas. Es posible que sea necesario crear un HUD que sea cómodo y preciso en pantallas táctiles pequeñas. Otra posible alternativa sería crear un gesto de deslizamiento desde el punto de origen hasta el punto al que se quiere ir, simulando que se está dibujando con el dedo el trazo.

Por otro lado, es necesario evolucionar el diseño personajes para conseguir un aspecto más atractivo, añadir animaciones más avanzadas y estudiar la introducción de elementos cosméticos y la posibilidad de seleccionar diferentes personajes. De esta manera se incentivaría a que el jugador seguir jugando para desbloquear diferentes objetos y jugar con los que más le gusten.

Se podrían implementar nuevas mecánicas en los circuitos como manchas de aceite que impidan girar al estar sobre ellas, colinas que al subir no permitan acelerar y al bajar frenar u objetos potenciadores que se puedan usar una vez por partida o se encuentren por el circuito para ganar bonificadores o perjuicios para los contrincantes.

Es necesario realizar una traducción de todos los textos a otros idiomas. En este proyecto únicamente se ha considerado el uso del castellano, pero como mínimo sería obligatorio tener los textos en inglés ya que es el idioma predominante en el posible mercado al que va dirigido el juego.

Por último, se sería muy interesante implementar un sistema de partidas y competiciones online. Este punto es muy importante a la hora de crear un juego competitivo exitoso. Se podría estudiar el uso de productos comerciales como Photon Engine[79] que permite el uso de 20 usuarios simultáneos de forma gratuita o desarrollar algún tipo de solución propia que permita abaratar los costes. Si quisiéramos no hacer uso de ningún framework propietario, se podría implementar un backend con alguna tecnología como springboot[80] que exponga los servicios necesarios o buscar alguna solución peer to peer (P2P) para evitar hacer uso de ningún servidor intermedio. Sin embargo, debido a la gran complejidad técnicas que cualquiera de todas estas soluciones tiene y al coste, no se ha contemplado en el alcance del proyecto.

Bibliografía

[1] Juego Apalabrados, Apalabrados website (13/01/2023)
https://apalabrados.com/#!
[2] Juego Draw It, Google Play website (13/01/2023)
https://play.google.com/store/apps/details?id=com.kwalee.drawit&hl=es&gl=US&pli=1
[3] Juego de mesa carreras por turnos Formula D, Zacatrus website (07/01/2023)
https://zacatrus.es/formula-d.html
[4] Juego carreras por turnos T3 Take the turn, Steam website (07/01/2023)
https://store.steampowered.com/app/939210/T3__Take_the_Turn/
[5] Herramienta Bitbucket, Bitbucket website (07/01/2023)
https://bitbucket.org/product/
[6]Herramienta Trello, Trello website (07/01/2023)
https://trello.com/es
[7]Herramienta TeamGantt, TeamGantt website (07/01/2023)
https://www.teamgantt.com/
[8] Manifiesto por el Desarrollo Ágil de Software, AgileManifesto website (07/01/2023)
https://agilemanifesto.org/iso/es/manifesto.html
[9] Games market numbers, revenues and audience, Newzoo website (07/01/2023)
https://newzoo.com/resources/blog/newzoo-games-market-numbers-revenues-and-audience-2020-2023
[10] Google Play Store: 10 estadísticas y curiosidades de la tienda, TN website (07/01/2023)
https://tn.com.ar/tecno/aplicaciones/2022/07/26/google-play-store-10-estadisticas-y-curiosidades-de-la-tienda-en-su-10-aniversario/
[11] El mercado de aplicaciones móviles, IT user website (07/01/2023)
https://www.ituser.es/en-cifras/2023/10/el-mercado-de-aplicaciones-moviles-ingresara-613000-millones-de-dolares-para-2025
[12] Ventas steam 2021, AS Meristation website (07/01/2023)
https://as.com/meristation/2021/07/08/noticias/1625717995_265957.html
[13] Nostalgia Market, AS Meristation website (07/01/2023)
https://www.spiralytics.com/blog/nostalgia-marketing-statistics-benefits-and-examples/
[14] Made with Unity, Unity website (07/01/2023)
https://unity.com/es/made-with-unity
[15] Un sinfín de posibilidades, Unreal Engine website (07/01/2023)
https://www.unrealengine.com/es-ES/solutions/games

[16] The game engine you've been waiting for, Godot website (07/01/2023)
https://godotengine.org/
[17] Unity o Godot, El drama de elegir motor de videojuegos, Canal Findemor Youtube (07/10/2023)
https://www.youtube.com/watch?v=HDSEDG2suDE
[18] Hecho con GameMaker, Gamemaker website (07/01/2023)
https://gamemaker.io/es/showcase
[19] Racetrack game, Wikipedia website (07/01/2023)
https://en.wikipedia.org/wiki/Racetrack_(game)
[20] Gardner, Martin (Enero 1973). "Scientific American Magazine Archives, January 1973, Mathematical Games". Scientific American.
[21] Pencil and Paper Games Race Track, Papg website (07/01/2023)
http://www.papg.com/show?1TPE
[22] Vampire survivors game, Google play website (07/01/2023)
https://play.google.com/store/apps/details?id=com.poncle.vampiresurvivors
[23] Itch platform, Itch.io website (07/01/2023)
https://itch.io/
[24] Patrón de diseño Facade, refactoring.guru website (07/01/2023)
https://refactoring.guru/es/design-patterns/facade
[25] Canvas, Unity Docs website (07/01/2023)
https://docs.unity3d.com/es/2019.4/Manual/UICanvas.html
[26] Documentación Book Page Curl Pro, Abdullah Aldandarawy github website (07/01/2023)
https://dandarawy.github.io/Unity3DBookPageCurlPro-Documentation/
[27]Book Page Curl Pro, Abdullah Aldandarawy Unity asset store (07/01/2023)
https://assetstore.unity.com/packages/tools/gui/book-page-curl-pro-77222
[28] Script Dependency Visualizer, Black Pea Studios Unity asset store (12/01/2023)
https://assetstore.unity.com/packages/tools/utilities/script-dependency-visualizer-57647?locale=es-ES
[29] Dall-e, Open AI website (12/01/2023)
https://openai.com/dall-e-2
[30] Javier Valls Figma project, Figma website (12/01/2023)
https://www.figma.com/file/2EMbr7L7dyZKttZuag4MN9/TFG-Javier-Valls?type=design&mode=design&t=p2YaBgm7525kXFNk-0
[31] The Free & Open Source Image Editor, Gimp website (12/01/2023)
https://www.gimp.org/

[32] Candy Crash, King website (12/01/2023)
https://www.king.com/es/game/candycrush
[33] Clash Royale, Supercell Play store (12/01/2023)
https://play.google.com/store/apps/details?id=com.supercell.clashroyale
[34] Cute Cartoon Mobile GUI - 97 png files!, HONETi Unity asset store (07/01/2023)
https://assetstore.unity.com/packages/2d/gui/cute-cartoon-mobile-gui-97-png-files-35315
[35] FREE Stylized PBR Textures Pack, Lumo-Art 3D Unity asset store (12/01/2023)
https://assetstore.unity.com/packages/2d/textures-materials/free-stylized-pbr-textures-pack-111778#description
[36] Animator, Unity Docs website (07/01/2023)
https://docs.unity3d.com/Manual/class-Animator.html
[37] Corutines, Unity Docs website (07/01/2023)
https://docs.unity3d.com/Manual/Coroutines.html
[38] Familias tipográficas, Nextu website (11/01/2023)
https://www.nextu.com/blog/que-son-familias-tipograficas-rc22/
[39] Fuente de letra Lilita One, Google website (07/01/2023)
https://fonts.google.com/specimen/Lilita+One
[40] Fuente de Margarine, Google website (07/01/2023)
https://fonts.google.com/specimen/Margarine
[41] Ojos my Little pony, Pinterest website (12/01/2023)
https://www.pinterest.com/pin/835277062116026257/
[42] Concepts app, Concepts website (12/01/2023)
https://concepts.app/en/
[43] Grand Theft Auto: San Andreas version móvil, As Meristation website (07/01/2023)
https://as.com/meristation/2013/12/13/analisis/1386936000_126066.html
[44]Small Page.wav by RAFAT913, Freesound website (11/01/2023)
https://freesound.org/people/RAFAT913/sounds/332433/#comments
[45] Switch Light 05.wav by tbrook Freesound website (11/01/2023)
https://freesound.org/people/tbrook/sounds/348225/
[46]breaking a pencil.wav by an_na_ba_na_na, Freesound website (11/01/2023)
https://freesound.org/people/an_na_ba_na_na/sounds/486374/
[47] 8-bit mini win sound effect by EVRetro, Freesound website (11/01/2023)
https://freesound.org/people/EVRetro/sounds/535840/
[48] Marker Lines.WAV by TubbsMedia, Freesound website (11/01/2023)
https://freesound.org/people/TubbsMedia/sounds/377124/

[49] Eraser-Notebook.wav by Paulabej, Freesound website (11/01/2023)
https://freesound.org/people/Paulabej/sounds/650472/
[50] Objective Complete by Jerimee, Freesound website (11/01/2023)
https://freesound.org/people/Jerimee/sounds/527530/
[51] Dbl Click.mp3 by 7778, Freesound website (11/01/2023)
https://freesound.org/people/7778/sounds/202312/
[52] Audio pitching, Gamedeveloper website (11/01/2023)
https://www.gamedeveloper.com/audio/the-power-of-pitch-shifting
[53] Audio Source, Unity docs website (11/01/2023)
https://docs.unity3d.com/es/2019.4/Manual/class-AudioSource.html#:~:text=El%20Audio%20Source%20reproduce%20un%20Audio%20Clip%20en,como%202D%2C%203D%2C%20o%20como%20una%20mezcla%20%28SpatialBlend%29.
[54] PyCharm, Jetbrains website (11/01/2023)
https://www.jetbrains.com/pycharm/download/
[55] Visual studio Code, Microsoft website (11/01/2023)
https://code.visualstudio.com/Download
[56] Player prefs, Unity Docs website (07/01/2023)
https://docs.unity3d.com/ScriptReference/PlayerPrefs.html
[57] Load scene, Unity Docs website (07/01/2023)
https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html
[58] Camera viewport to world point, Unity Docs website (07/01/2023)
https://docs.unity3d.com/ScriptReference/Camera.ViewportToWorldPoint.html
[59] The right way to Lerp in Unity (with examples), Gamedevbeginner website (07/01/2023)
https://gamedevbeginner.com/the-right-way-to-lerp-in-unity-with-examples/
[60] How to Lerp like a pro, chicounity3d website (07/01/2023)
https://chicounity3d.wordpress.com/2014/05/23/how-to-lerp-like-a-pro/
[61] Line Renderer, Unity Docs website (07/01/2023)
https://docs.unity3d.com/es/2018.4/Manual/class-LineRenderer.html
[62] On draw gizmos, Unity Docs website (07/01/2023)
https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDrawGizmos.html
[63] GL Lines, Unity Docs website (07/01/2023)
https://docs.unity3d.com/ScriptReference/GL.LINES.html
[64] GL Vertex3, Unity Docs website (07/01/2023)

https://docs.unity3d.com/ScriptReference/GL.Vertex3.html
[65] Z-buffering, Wikipedia website (07/01/2023)
https://en.wikipedia.org/wiki/Z-buffering
[66] Aliasing, Wikipedia website (07/01/2023)
https://es.wikipedia.org/wiki/Aliasing
[67] Math clamp, Unity Docs website (07/01/2023)
https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html
[68] Line intersection, Unity forum website (07/01/2023)
https://forum.unity.com/threads/line-intersection.17384/
[69] Use math to solve problems in Unity with C#, Hablador website (07/01/2023)
https://www.habrador.com/tutorials/math/5-line-line-intersection/
[70] Intersection two vector, StackOverflow website (07/01/2023)
https://stackoverflow.com/questions/73079419/intersection-of-two-vector
[71] Ray-casting algorithm, Rosettacode website (07/01/2023)
https://rosettacode.org/wiki/Ray-casting_algorithm
[72] Ray-casting algorithm, Wikipedia website (07/01/2023)
https://en.wikipedia.org/wiki/Point_in_polygon#Ray_casting_algorithm
[73] C# pass a list by reference, StackOverflow website (07/01/2023)
https://stackoverflow.com/questions/23490802/does-c-sharp-pass-a-listt-to-a-method-by-reference-or-as-a-copy
[74] Generating procedural racetracks, Gamedeveloper website (07/01/2023)
https://www.gamedeveloper.com/programming/generating-procedural-racetracks
[75] Convex hull algorithm, Wikipedia website (07/01/2023)
https://en.wikipedia.org/wiki/Convex_hull
[76] Catmull-Rom spline algorithm, Wikipedia website (07/01/2023)
https://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline
[77] On the Parameterization of Catmull-Rom Curves, Yuksel C., Schaefer S., Keyser J. (07/01/2023)
http://www.cemyuksel.com/research/catmullrom_param/catmullrom.pdf
[78] System requirements for Unity 2021.1 - Unity Manual (07/01/2023)
https://docs.unity3d.com/es/2021.1/Manual/system-requirements.html
[79] Multiplayer solution, Photon Engine website (07/01/2023)
https://www.photonengine.com/
[80] Framework para backend java, Spring.io website (07/01/2023)
https://spring.io/projects/spring-boot/

Anexos

Anexo A: Glosario

Slider: Un control de interfaz de usuario que permite al usuario ajustar un valor deslizando un indicador dentro de un rango definido.

Script: Código escrito, generalmente en C#, que se adjunta a los GameObjects para definir su comportamiento, interacciones y lógica del juego.

Audio Source: Un componente en Unity que se puede agregar a un GameObject para reproducir sonidos. Se utiliza para asignar clips de audio y controlar propiedades como volumen, tono y distancia de audición.

Audio Listener: Un componente que actúa como un "oído" en la escena. Normalmente se adjunta a la cámara principal y es el punto en el que se recogen los sonidos para su reproducción.

GameObject: El objeto fundamental en Unity. Puede representar personajes, elementos del escenario, cámaras, luces, y prácticamente cualquier tipo de entidad en el juego. Los GameObjects sirven como contenedores para Componentes.

Sprite: Una imagen 2D utilizada en gráficos y animaciones. En Unity, se utiliza para representar entidades en juegos 2D, como personajes, fondos, y objetos.

Componente: Los bloques de construcción de los GameObjects en Unity. Un GameObject se define por los componentes que tiene adjuntos, como físicas, renderizado, scripts, etc.

HUD (Head-Up Display): Una interfaz de usuario superpuesta en la pantalla de juego que muestra información relevante para el jugador, como salud, puntuación, mapas, y más.

Animator: Un componente que controla las animaciones de un GameObject. Se usa para gestionar estados de animación y transiciones basadas en lógica o eventos del juego.

Escenas (Scenes): En Unity, una escena es como un contenedor individual donde se diseñan y construyen una parte específica de tu juego o aplicación. Cada escena puede contener sus propios GameObjects, como personajes, entornos, cámaras y luces. Las escenas se utilizan para organizar y separar diferentes niveles, menús, pantallas de inicio, y otros segmentos de tu juego o aplicación. La transición entre escenas se puede utilizar para mover al jugador entre diferentes niveles o secciones del juego.

Anexo B: Entregables del proyecto

- Código fuente del proyecto:
https://javalca@bitbucket.org/jvallscat/racetrace.git
- Juego para navegador:
https://birrefringencia.itch.io/racetrace
- Zip con juego para Windows.
https://drive.google.com/file/d/1CUJTtHrS27V6AhIZmBd8YZkuwAGENvql/view?usp=drive_link
- Wireframe proyecto:
https://www.figma.com/file/2EMbr7L7dyZKttZuag4MN9/TFG-javier-valls?type=design&node-id=0%3A1&mode=design&t=REJksgMPgFNv1ekh-1

Anexo D: Currículum Vitae

Nombre: Javier Valls Catalá

Contacto: javivc73@gmail.com, https://www.linkedin.com/in/javiervallscatala/

Perfil Personal:

Ingeniero de Telecomunicaciones y estudiante de último año de Ingeniería Informática con especial interés en el desarrollo de software y la inteligencia artificial. Apasionado por los videojuegos y la tecnología.

Educación:

- Ingeniería de Telecomunicaciones en la Universidad Politécnica de Valencia (UPV), 2008-2013.
- Grado en Ingeniería informática en la Universitat Oberta de Cataluña (UOC), 2019-2024.

Experiencia laboral:

- Becario en Telefonica Digital, noviembre 2012- mayo 2013.
    - Apoyo a Startups.
- Becario en Telefonica España, octubre 2013- octubre 2014.
    - Desarrollo de servicios y sistemas.
- Ingeniero de Telecomunicaciones en Telefónica España, noviembre 2014- actualidad.
    - Desarrollo de sistemas.
    - Desarrollador backend de las aplicaciones móviles de Telefónica (Mi Movistar, Mi Movistar Empresas, Mi O2).
    - Jefe de proyecto canales online.

Habilidades técnicas:

- Programación: Java, Python, C#
- Herramientas: Unity, Gimp, Android Studio, IntelliJ IDEA, Docker.
- Desarrollo web: HTML, CSS, JavaScript.

Idiomas:

- Castellano (Nativo).
- Catalán (Fluido).
- Inglés (Fluido).