

# SUPER MARIO KART

**GRUPO 4**

**DIEGO ROYO MENESES 740388**

**VICTOR MARTINEZ BATLLE 736478**

**JAVIER GIMENEZ GARCES 738866**



# Índice

<b>1. Documentos de diseño</b>	<b>4</b>
1.1. Resumen	4
1.1.1. Información básica	4
1.1.2. Género y plataforma	5
1.2. Público objetivo	5
1.3. Lógica de juego	6
1.3.1. Ranking	6
1.3.2. Trofeos	7
1.4. Circuitos	7
1.4.1. Obstáculos	10
1.4.2. Power-Ups	11
1.5. Personajes y vehículos	13
<b>2. Documentos de diseño</b>	<b>14</b>
2.1. Organización del proyecto	14
2.2. Diagrama de clases	15
2.3. Diagramas de navegación	20
<b>3. Detalles de implementación</b>	<b>22</b>
3.1. Lógica principal del juego y estados	22
3.2. Máquina de pila y flujos de información	23
<b>4. Aspectos gráficos</b>	<b>24</b>
4.1. Cámara 2.5D y Mode7	24
4.2. Elementos horizontales	26
4.3. Elementos verticales	27
4.3.1. Efectos	30
4.4. GUI y jugador	31
4.4.1. Lakitu	32
4.4.2. Jugador	33
4.4.3. Minimapas	35

4.5. Cinemáticas	35
<b>5. Aspectos físicos</b>	<b>37</b>
5.1. Vehículos	37
5.2. Mapa de terrenos	40
5.3. Colisiones	41
5.3.1. Con elementos horizontales	41
5.3.2. Con elementos verticales	43
5.4. Gestión de estados del personaje	44
<b>6. Aspectos de audio</b>	<b>45</b>
6.1. Gestor de sonido	45
6.2. Música	45
6.3. Efectos de sonido o SFX	46
6.3.1. Sonido de motores	47
<b>7. Jugabilidad</b>	<b>48</b>
7.1. Controles	48
7.1.1. Opciones y configuración	49
7.2. Diseño y generación de niveles	49
7.3. Niveles de dificultad	52
<b>8. Inteligencia Artificial</b>	<b>52</b>
8.1. Direcciones y aceleración	52
8.1.1. Acciones especiales	55
8.2. Relocalización de corredores	56
8.3. Power-Ups	56
<b>9. Tecnologías y herramientas de implementación</b>	<b>57</b>
9.1. Lenguajes y bibliotecas	57
9.2. Control de versiones y herramientas de edición digital	58
<b>10. Gestión del proyecto</b>	<b>59</b>
10.1. Problemas encontrados	59
10.2. Reparto de tareas y cronograma	60
<b>11. Referencias</b>	<b>63</b>

# 1. Documentos de diseño

## 1.1. Resumen

### 1.1.1. Información básica

Super Mario Kart es un juego de carreras para ordenador (PC), donde un jugador se enfrenta a la máquina (CPU) para intentar recorrer los circuitos a mayor velocidad. Dispones de varios circuitos ordenados en diferentes copas o «*Grands prix*». En cada uno de ellos, ocho corredores competís por la primera posición y obtenéis un número de puntos en base a vuestra posición final. Para lograr ganar deberás usar, en el momento adecuado, diferentes *power-ups* distribuidos por los circuitos. Tu clasificación del «*Grand prix*» será la suma de los puntos que consigas en cada una de las carreras. ¡Sólo los tres primeros competidores subirán al pódium!

El juego está ambientado en un mundo ficticio en el que varios tipos de personajes (fontaneros, princesas, tortugas, dinosaurios, setas, monos, etc.) recorren circuitos situados en zonas de campo, castillos o incluso el espacio.

Este juego casual está orientado a todos los públicos, aunque dispone de opciones de dificultad para ofrecer un reto considerable a usuarios más experimentados.



Figura 1. Pantallas de ejemplo de una partida normal en dos mapas diferentes con distintos corredores.

### 1.1.2. Género y plataforma

El género de carreras consiste en recrear competiciones entre vehículos de diferentes características. En este caso, Super Mario Kart ofrece este género de forma arcade dándole un aspecto más divertido.

La plataforma elegida para el desarrollo del videojuego es el ordenador convencional (PC) debido a su gran disponibilidad, por lo que el formato del juego es digital.

Otros aspectos a destacar son el modo de juego: un jugador; y el uso del teclado para controlar las acciones del corredor.

Super Mario Kart	
<b>SUPER MARIO KART</b>	
<b>Plataforma(s)</b>	PC
<b>Fecha(s) de lanzamiento</b>	26 de mayo de 2020
<b>Modos de juego</b>	Un Jugador
<b>Formato(s)</b>	Digital
<b>Controles</b>	Teclado

### 1.2. Público objetivo

Este juego va dirigido a todos los públicos. Para ello se dispone de tres modos de dificultad: fácil, normal y difícil. En el nivel más bajo, 50cc, los jugadores menos experimentados podrán tener un mayor manejo del vehículo al disminuir la velocidad punta y aumentar la aceleración de giro. Estos parámetros son adaptados en los siguientes niveles, 100cc y 150cc, para suponer un desafío para los usuarios más expertos. Además, el jugador verá cómo la agresividad de la IA aumenta nivel a nivel, obstaculizando su carrera y poniéndole realmente difícil hacerse con el primer puesto de la carrera.



## 1.3. Lógica de juego

El juego se divide en diferentes pantallas que dan lugar a una partida completa de dos maneras o modos posibles: Grand Prix (Copa) o Versus. En el primero se deben recorrer los cinco circuitos disponibles para terminar la partida; mientras que en el modo Versus se puede elegir uno de los circuitos y correrlo individualmente.

En el menú principal se permite seleccionar, junto con los ajustes de configuración y controles, los dos modos anteriormente descritos. Tras elegir el modo, se procederá a seleccionar la dificultad entre 50 (fácil), 100 (normal) o 150 (difícil) centímetros cúbicos (CC). A continuación, en caso de haberse decidido por el modo Versus, el jugador pasará a la pantalla de selección de circuito. Finalmente, se ofrece la elección de personaje entre los ocho avatares disponibles.

Una vez seleccionados todos los parámetros del juego, da comienzo la partida con una presentación del circuito y la preparación de la parrilla de salida. Un semáforo marcará la cuenta atrás antes de empezar a correr. La carrera cuenta con 5 vueltas completas a la pista. Al terminar, en función del modo elegido, o bien se muestran los resultados de la última carrera y se acumulan a las puntuaciones anteriores (modo Grand Prix), o bien se finaliza la partida con la pantalla de entrega de premios (modo Versus).

En el modo Grand Prix, tras acabar un circuito, se repite el mismo procedimiento (preparación, carrera, resultados) hasta llegar al último, en el que se da paso a la pantalla de premios. Al acabar ambos modos se vuelve al inicio o menú principal para poder jugar una nueva partida o salir del juego.

### 1.3.1. Ranking

En el modo Grand Prix, al terminar una carrera, se otorgan puntos a los corredores en función de la posición alcanzada. Si un corredor cruza la línea de meta en mal lugar, deberá esforzarse por recuperar su posición en el ranking, sumando más puntos en la siguiente carrera. De esta forma, el juego es más divertido y desafía al jugador. Tras finalizar el Grand Prix, los tres jugadores con mayor puntaje subirán al podio para celebrar los logros conseguidos.

En el modo Versus, sólo los tres corredores más rápidos de la prueba conseguirán ganar un hueco en el podio. No habrá segundas oportunidades.

Tabla de puntuaciones	
	10 puntos
	8 puntos
	6 puntos
<b>4º</b>	4 puntos
<b>5º</b>	3 puntos
<b>6º</b>	2 puntos
<b>7º</b>	1 punto
<b>8º</b>	0 puntos

### 1.3.2. Trofeos

Al terminar un circuito, se celebra una entrega de trofeos donde se premia a los mejores corredores. Los premios aumentan con la dificultad. ¡Compete en varias carreras y colecciona todos ellos!

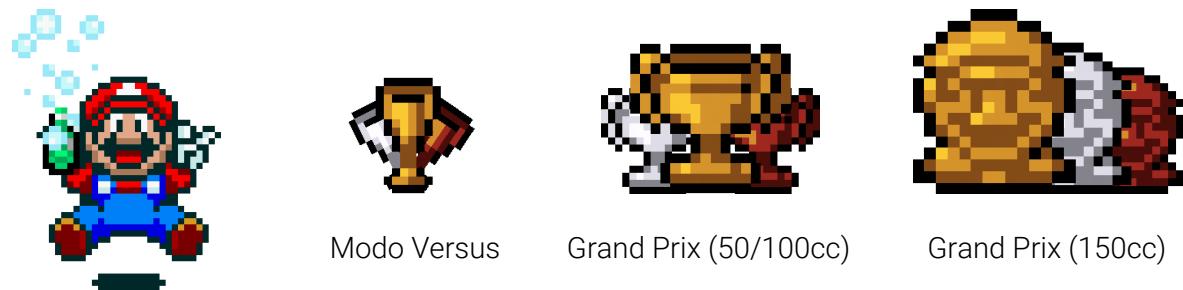
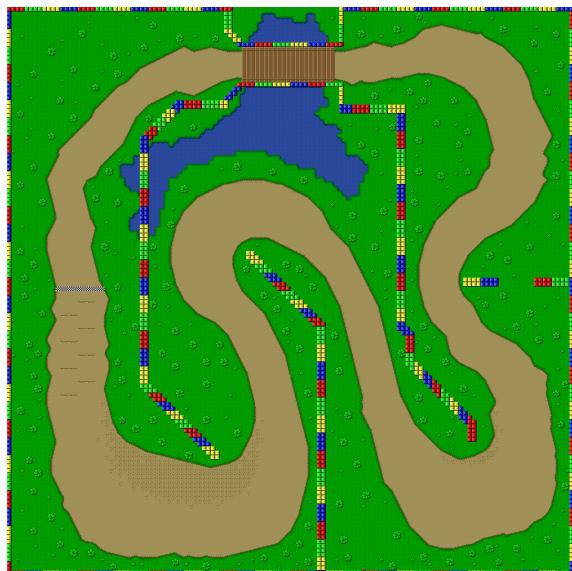


Figura 2. Trofeos otorgados en las diferentes modalidades.

El trofeo del Mario dorado solo se entrega a los mejores corredores. ¿Serás capaz de lograr la victoria?

### 1.4. Circuitos

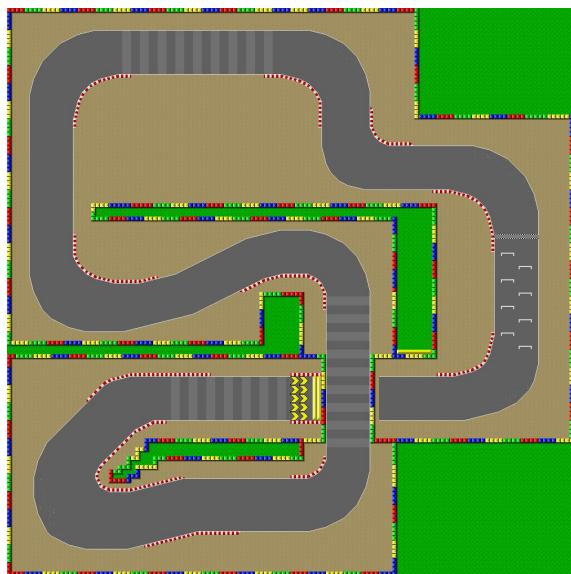
El juego recoge un total de 5 circuitos diferentes, los cuales forman el «Grand Prix». Cada uno tiene sus características y peculiaridades (obstáculos, terrenos, ambiente). Éstas quedan resumidas en las Figuras 3, 4, 5, 6 y 7.



Terrenos	
Tierra (Principal)	
Hierba (Lento)	
Agua (Caída)	

Elementos	
Tuberías	

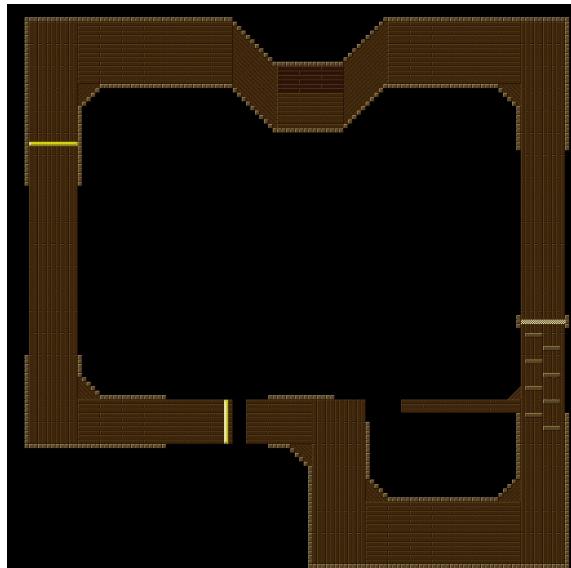
Figura 3. Circuito "Donut Plains 1" con sus terrenos y elementos característicos.



Terrenos	
Asfalto (Principal)	
Tierra (Lento)	

Elementos	
Tuberías	
Rampas	
Charco de aceite	
Aceleradores	

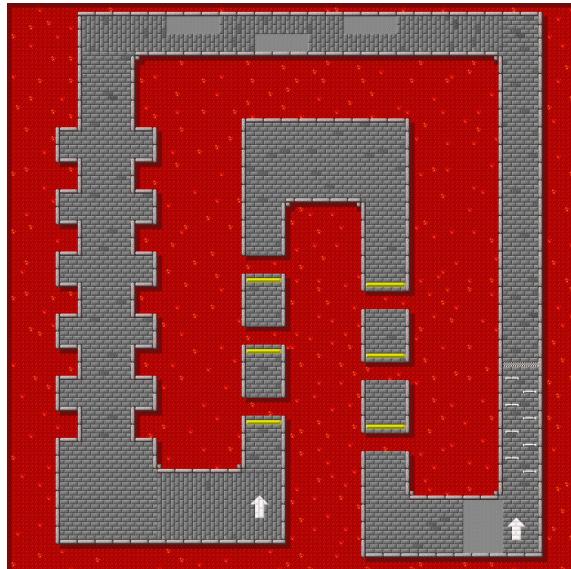
Figura 4. Circuito “Mario Circuit 2” con sus terrenos y elementos característicos.



Terrenos	
Madera (Principal)	
Vacio (Caída)	

Elementos	
Rampas	

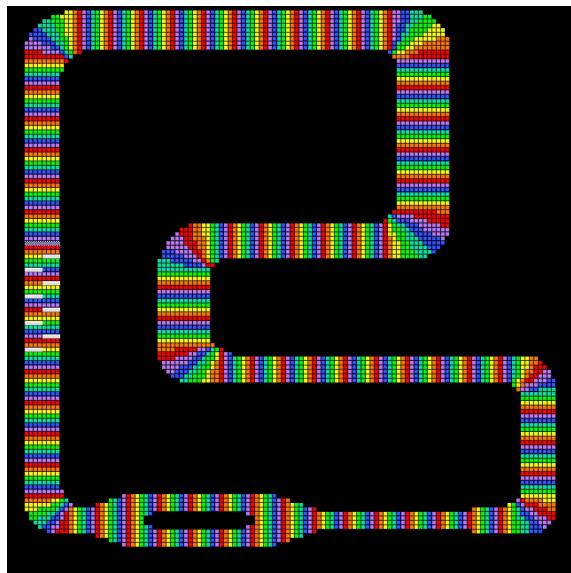
Figura 5. Circuito “Ghost Valley 1” con sus terrenos y elementos característicos.



Terrenos	
Ladrillo (Principal)	
Lava (Caída)	

Elementos	
Rampas	
Aceleradores	
Roca picuda	

Figura 6. Circuito “Bowser Castle 1” con sus terrenos y elementos característicos.



Terrenos	
Arcoiris (Principal)	
Vacío (Caída)	

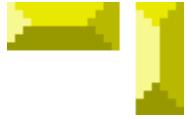
Elementos	
Súper roca picuda	
Rampas	

Figura 7. Circuito “Rainbow Road” con sus terrenos y elementos característicos.

Además de los elementos específicos de cada circuito, en todos ellos se encuentran monedas ( y cajas de objetos () repartidas por varias zonas de la pista.

#### 1.4.1. Obstáculos

A lo largo de cada circuito se encuentran distintos objetos que obstaculizan el paso de los corredores para forzar su trayectoria, darles un impulso extra, elevarlos del suelo o hacerlos girar sin control.

Sprites	Descripción
	<b>Nombre:</b> Muro <b>Efecto:</b> Impide el paso del jugador. Al chocar contra uno de ellos, el corredor rebotará en dirección contraria. Este choque no le hará perder monedas.
	<b>Nombre:</b> Zipper <b>Efecto:</b> Cuando el corredor pasa por encima, recibe un impulso extra, aumentando instantáneamente su velocidad.
	<b>Nombre:</b> Caja de objetos <b>Efecto:</b> Activando el panel, el jugador recibe un objeto aleatorio que podrá utilizar cuando desee para obtener una ventaja en la carrera.
	<b>Nombre:</b> Charco de aceite <b>Efecto:</b> El coche del corredor resbala y hace perder el control al jugador durante un tiempo. Se restarán dos monedas de su marcador.
	<b>Nombre:</b> Moneda <b>Efecto:</b> Acumulando monedas el corredor podrá aumentar su velocidad máxima.
	<b>Nombre:</b> Rampa <b>Efecto:</b> Al subir por una rampa, el vehículo del corredor se eleva del suelo, pudiendo incluso sobrepasar algunos muros del circuito.

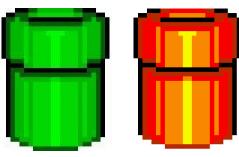
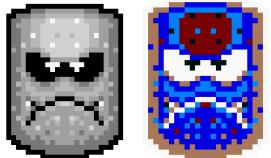
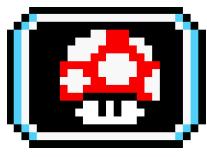
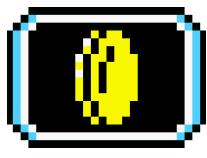
	<b>Nombre:</b> Tubería  <b>Efecto:</b> El jugador rebota en la dirección contraria a la del choque. Le hará perder una moneda.
	<b>Nombre:</b> Thwomp (Roca picuda)  <b>Efecto:</b> Podrá aplastar al jugador con todo su peso al caer. Haciéndole perder tres monedas. Si sólo choca contra él, actuará como una tubería.

Figura 8. Obstáculos y elementos especiales de los circuitos.

#### 1.4.2. Power-Ups

Los objetos especiales o *Power-Ups* ayudan a los corredores a adelantar posiciones, concediendo una ventaja frente al resto de competidores u obstaculizando la carrera de uno o varios oponentes. El jugador puede decidir cuándo activa el objeto para hacer uso del efecto del *Power-Up*.

Sprites	Descripción
	<b>Nombre:</b> Champiñón  <b>Efecto:</b> Da un impulso extra al jugador y le permite superar su velocidad límite durante un segundo y medio para adelantar y ganar posiciones en la carrera.
	<b>Nombre:</b> Moneda  <b>Efecto:</b> Otorga al jugador dos monedas, cuyo efecto es análogo al descrito en la Figura 8.

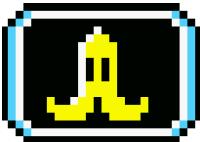
	<p><b>Nombre:</b> Estrella</p> <p><b>Efecto:</b> Vuelve al jugador invencible durante diez segundos. No tendrá penalización por choques ni por circular sobre terreno lento. Si choca contra otro jugador, el oponente girará sin control y perderá dos monedas. Un choque entre dos jugadores invencibles se trata como un choque normal.</p>
	<p><b>Nombre:</b> Cáscara de plátano</p> <p><b>Efecto:</b> El jugador podrá depositarla tras de sí o lanzarla hacia delante. Cuando un corredor pisa una cáscara comienza a girar sin control.</p>
	<p><b>Nombre:</b> Caparazón verde</p> <p><b>Efecto:</b> Si se posee una buena puntería, se puede alcanzar a un oponente lanzando un caparazón hacia delante o hacia atrás. Los caparazones verdes pueden rebotar contra las paredes del circuito para alcanzar a un corredor y que éste pierda el control del su vehículo durante unos instantes.</p>
	<p><b>Nombre:</b> Caparazón rojo</p> <p><b>Efecto:</b> El objetivo de este caparazón es golpear al corredor que precede al jugador en la carrera. El contrincante será perseguido hasta ser alcanzado y golpeado.</p>
	<p><b>Nombre:</b> Rayo</p> <p><b>Efecto:</b> Afectará a todos los corredores que preceden al jugador en el momento de ser activado. Todos ellos serán golpeados y su tamaño y velocidad quedarán reducidos durante diez segundos. Excepto si son inmunes gracias a una estrella.</p>

Figura 9. Power-Ups y sus efectos.

## 1.5. Personajes y vehículos

El juego cuenta con ocho personajes elegibles, cada uno con su respectivo vehículo. Existen cuatro tipos de vehículos diferentes en cuanto a características físicas (aceleración, velocidad máxima, manejo y peso). Podemos diferenciarlos por los calificativos Bajo, Medio, Alto, Muy alto. En la Figura 10, se recogen los diferentes personajes y sus correspondientes vehículos.

BALANCEADOS	POTENTES	ÁGILES	PESADOS
			
Mario	Peach	Koopa Troopa	Bowser
			
Luigi	Yoshi	Toad	Donkey Kong
<b>Características</b>			
<b>Aceleración:</b> Media <b>Velocidad:</b> Alta <b>Manejo:</b> Medio <b>Peso:</b> Medio	<b>Aceleración:</b> Muy alta <b>Velocidad:</b> Media <b>Manejo:</b> Bajo <b>Peso:</b> Medio	<b>Aceleración:</b> Alta <b>Velocidad:</b> Baja <b>Manejo:</b> Alto <b>Peso:</b> Bajo	<b>Aceleración:</b> Baja <b>Velocidad:</b> Muy alta <b>Manejo:</b> Bajo <b>Peso:</b> Alto

Figura 10. Características principales por grupo de personajes.

## 2. Documentos de diseño

### 2.1. Organización del proyecto

El proyecto consta de 105 ficheros de código fuente, organizados según criterios de máxima cohesión y mínimo acoplamiento. Todos ellos se pueden resumir en el diagrama de organización mostrado en la Figura 11.

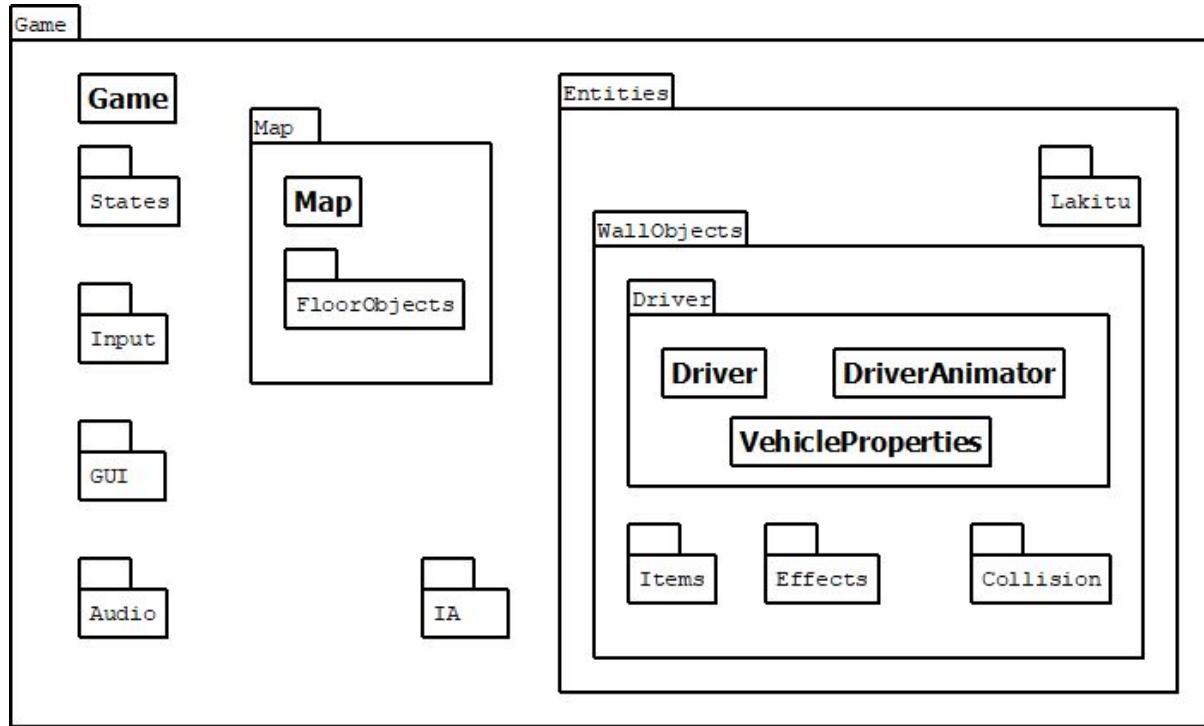


Figura 11. Diagrama de organización del proyecto.

En la Figura 11 aparecen algunos paquetes y clases representativos de la estructura externa del proyecto. A continuación se resume el contenido y la semántica de cada uno.

- **Paquete «Game»:** Incluye la totalidad del juego.
- **Clase «Game»:** Maneja la lógica general de la aplicación. Permite la gestión del estado del juego, ofrece primitivas para cargar, mostrar y descargar estados.
- **Paquete «States»:** Contiene la definición de los distintos estados del juego, como son: el menú de inicio, la presentación del circuito, el desarrollo de la carrera, entre otros.
- **Paquete «Input»:** Permite la interacción con el jugador a través del teclado.
- **Paquete «GUI»:** Controla la apariencia de la interfaz mostrada al usuario durante las carreras. Esto incluye: marcador de posición, contador de monedas, cajón de objetos... También gestiona la lógica ampliada para mostrar al jugador principal en la pantalla.

- **Paquete «Audio»:** Ofrece una interfaz al resto de la aplicación para el manejo de la música de fondo y la reproducción de efectos de sonido.
- **Paquete «Map»:** Engloba toda la lógica dependiente del circuito, incluyendo: límites de la pista, tipos de terreno y gestión de objetos.
  - **Clase «Map»:** Constituye la clase principal del paquete. Actúa como interfaz entre la mayoría de las clases del paquete, así como con elementos externos.
  - **Paquete «FloorObjects»:** El resto de las clases del paquete representan objetos horizontales que aparecen sobre el suelo del circuito.
- **Paquete «Entities»:** Agrupa las clases que simbolizan objetos, fijos o móviles, que forman parte del circuito durante el transcurso de la carrera pero que no se encuentran adosados al suelo horizontal del mapa.
  - **Paquete «WallObjects»:** Representa a todas las clases de objetos verticales que comparten una interfaz común. Es decir: corredores, objetos y efectos.
    - **Paquete «Driver»:** Contiene las características de los vehículos («VehicleProperties»), las físicas de los corredores; así como lo necesario para mantener el estado del corredor («Driver») y representarlo adecuadamente por pantalla («DriverAnimator»).
    - **Paquete «Items»:** Agrupa todos los objetos verticales con entidad propia, que existen con independencia de los corredores. Habitualmente Power-Ups, pero también tuberías y otros.
    - **Paquete «Effects»:** Son efectos visuales que acompañan a los corredores e ítems durante algunas fases de la carrera, sean: partículas de choque, animación de consecución o pérdida de monedas, salpicaduras de agua o lava, etc.
  - **Paquete «Collision»:** Calcula y ayuda a la gestión de las colisiones entre corredores o de éstos con el resto de objetos verticales (Items).
  - **Paquete «Lakitu»:** Gestiona las animaciones de Lakitu.
- **Paquete «IA»:** Implementación de la inteligencia artificial aplicada a los personajes no jugables durante el transcurso de las carreras.

## 2.2. Diagrama de clases

En este subapartado se detalla el contenido de los principales paquetes descritos en el diagrama de la Figura 11. Los diagramas que se presentan a continuación no pretenden describir exhaustivamente los atributos y métodos de las clases diseñadas, sino dar al lector una visión global de las decisiones de diseño tomadas para el desarrollo del videojuego.

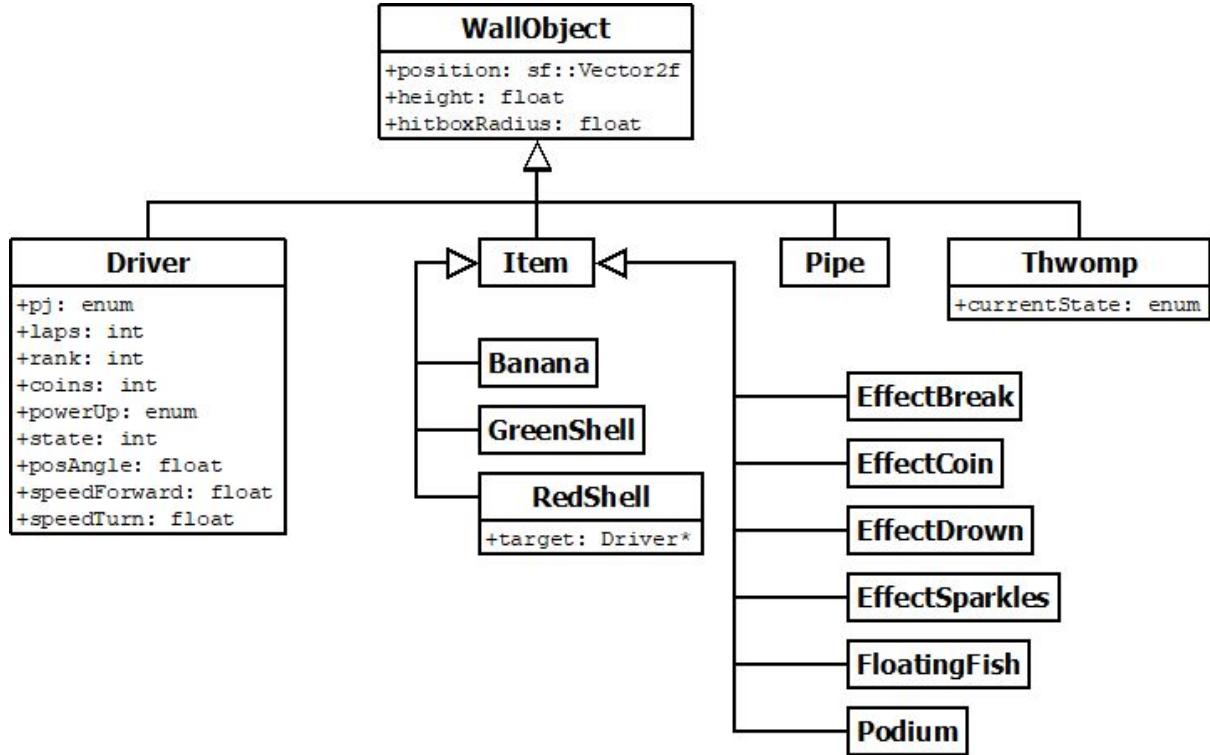


Figura 12. Diagrama de clases del paquete «WallObjects».

En la Figura 12 se muestra la organización diseñada para las entidades contenidas dentro del paquete «WallObjects» (Figura 11). Cabe destacar:

- **WallObject:** Todo objeto vertical situado en cierta posición (coordenadas cartesianas en el plano del mapa) y a una altura determinada (con origen en el plano de mapa). En el caso de objetos sólidos, se especifica su radio de colisión.
- **Driver:** Cada uno de los corredores de la carrera. Un corredor tiene asignado un personaje y acumula un número de vueltas completadas. También forma parte del estado del corredor su posición en el *ranking*, el número de monedas que conserva en cada momento, así como el objeto especial que dispone tras atravesar una caja de objetos. Los efectos causados por la interacción con enemigos o *Power-Ups* se representan en los bits menos significativos de un entero (p. ej. velocidad extra, inmunidad, velocidad reducida, descontrolado, etc.). Finalmente, las variables físicas de dirección, velocidad linea y velocidad angular permiten simular el movimiento de cada corredor a lo largo de circuito.
- **Item:** Se conoce como ítem a todo objeto vertical capaz de moverse, exceptuando a los corredores. Algunos de ellos son la materialización de un *Power-Up* (cáscara de plátano, caparazón verde y rojo – con su objetivo), otros son efectos visuales (ruptura de un *Power-Up*, adición o pérdida de monedas, ahogamiento y choque). Por último, pertenecen a esta clase los peces flotantes y el pódium mostrados al finalizar una carrera o un *Grand Prix*.

- **Pipe:** Representa una tubería fija que obstaculiza el paso de los corredores.
- **Thwomp:** Simboliza una roca, enemigo que puede variar su altura y su estado, aplastando o haciendo chocar a los corredores.

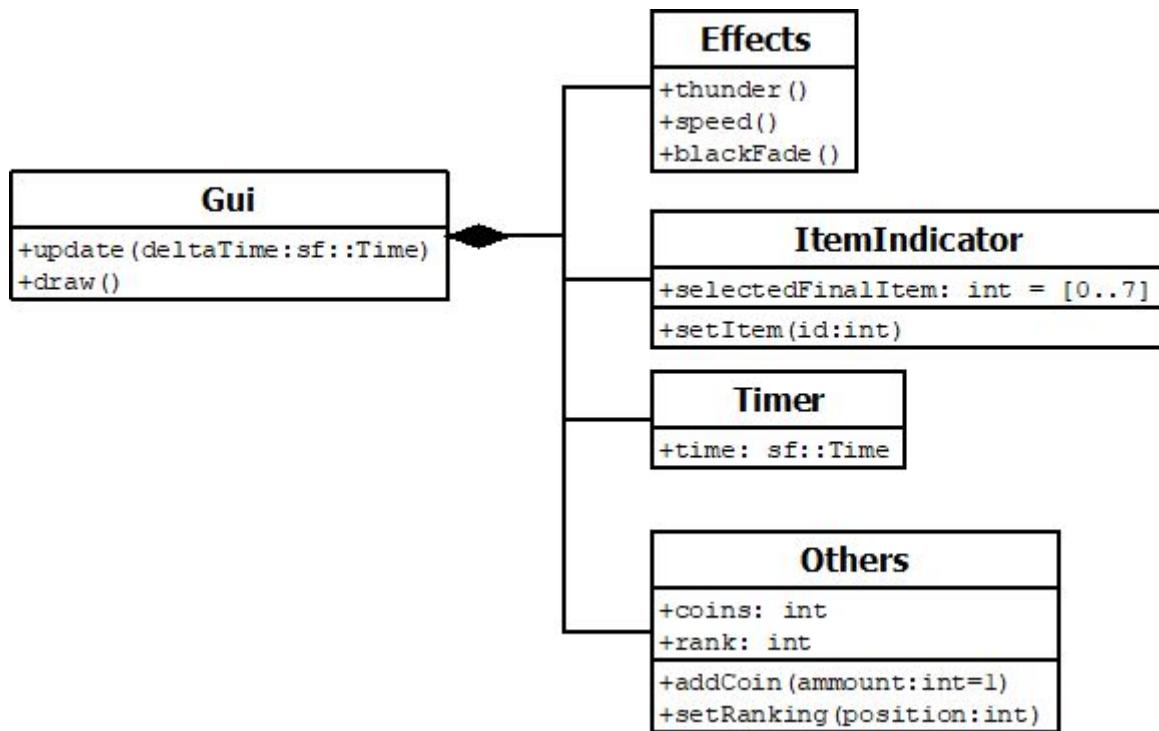


Figura 13. Diagrama de clases del paquete «GUI».

En la Figura 13 puede apreciarse la relación de composición entre las clases que forman el paquete «GUI» (Figura 11). Cada una de ellas se describe como:

- **Gui:** Es un `Singleton` cuya única instancia contiene el resto de elementos de la interfaz de usuario. Los métodos de actualización y dibujado, estáticos para la clase, permiten modificar el estado interno del resto de elementos mediante una interfaz que invocará directamente a los métodos correspondientes de cada clase.
- **Effects:** Maneja el estado de los efectos visuales que afectan a la pantalla del jugador. Estos son: destellos blancos al activarse un rayo, líneas de velocidad y fundido a negro cuando el jugador sale fuera de los límites del terreno.
- **ItemIndicator:** Controla las animaciones y el dibujado del cajón de *Power-Up* disponible para ser utilizado por el jugador.
- **Timer:** Almacena y representa en pantalla el cronómetro de tiempo de carrera.
- **Others:** Agrupa el resto de elementos de la GUI, como el contador de monedas o el marcador de la posición del jugador en el *ranking* de la carrera actual.

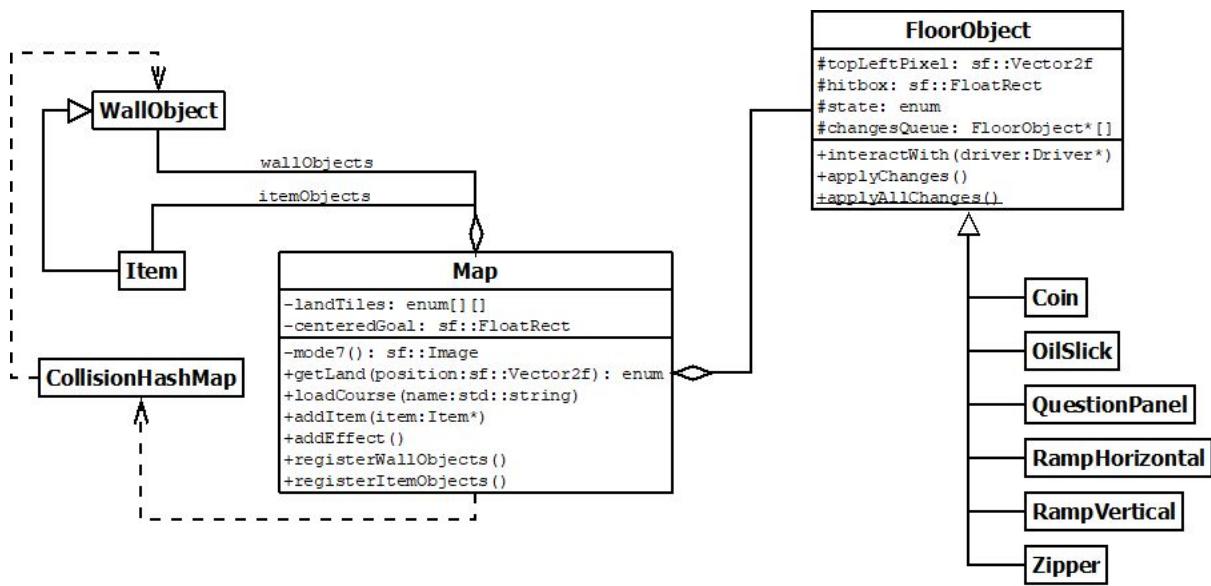


Figura 14. Diagrama de clases del paquete «Map» y sus dependencias.

En la Figura 14, la clase principal del paquete «Map» se constituye como una agregación de todos los *WallObjects* y *FloorObjects* presentes en el circuito:

- **Map:** Actúa como interfaz con las clases de otros paquetes (*WallObject*, *Item*, *CollisionHashMap*) y agrupa también los *FloorObjects* presentes en el plano del mapa. Esta clase guarda la información necesaria para diferenciar los tipos de terreno de cada zona del mapa, así como los límites de la pista y la posición de la línea de meta. Además, contiene la lógica para dibujar los assets y sprites mediante Mode7.
- **CollisionHashMap:** La clase, perteneciente al paquete «Entities», se encarga de gestionar las colisiones entre los distintos *WallObjects*. Estos objetos son registrados desde el método correspondiente de la clase «Map».
- **FloorObject:** Engloba a todos los objetos horizontales que aparecen sobre el suelo del circuito. Estos tienen una posición y tamaño fijos, pero su estado puede ser variable. Los cambios de estado se gestionan una vez por cada frame, mediante una cola. Al ser pisados, estos objetos interactúan con el jugador. Existen seis tipos: monedas, chacos de aceite, cajas de objetos, rampas horizontales, verticales y zippers.

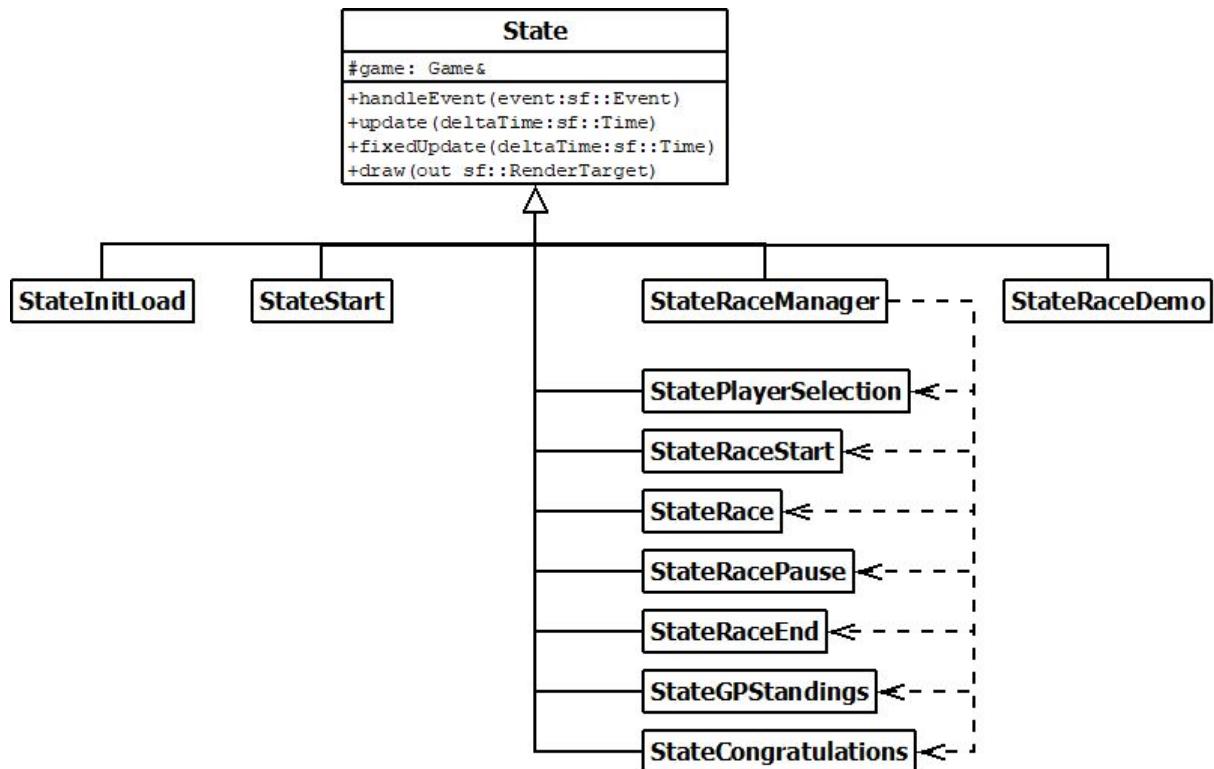


Figura 15. Diagrama de clases del paquete «States».

En la Figura 15, se resumen las clases que representan los posibles estados del juego:

- **State:** Se trata de la clase genérica de la que derivan todos los posibles estados de la aplicación. Mantiene una referencia a la instancia del juego y permite tratar eventos de teclado generados por el usuario. Los estados son actualizados periódicamente y contienen la lógica de alto nivel para el dibujado de los frames del juego.
- **StateInitLoad:** Es el estado de arranque del juego, encargado de la carga de contenido.
- **StateStart:** Presenta los menús de selección de modalidad, circuito, así como la personalización de los controles y la configuración.
- **StateRaceManager:** Se encarga de orquestar el inicio de una partida, dando paso a los estados de selección de personaje, presentación del circuito, carrera, pausa, fin de la carrera, resultados y proclamación de los ganadores.
- **StateRaceDemo:** Este estado, similar a «StateRace» permite mostrar una *demo* del juego durante la pantalla del título si el jugador no interactúa con el teclado.

### 2.3. Diagramas de navegación

Como se ha indicado en el anterior apartado [Lógica de Juego](#), una ejecución pasa por diferentes pantallas. El siguiente diagrama de navegación (Figura 16) resume cómo es dicha navegación a través de las pantalla de la Figura 17.

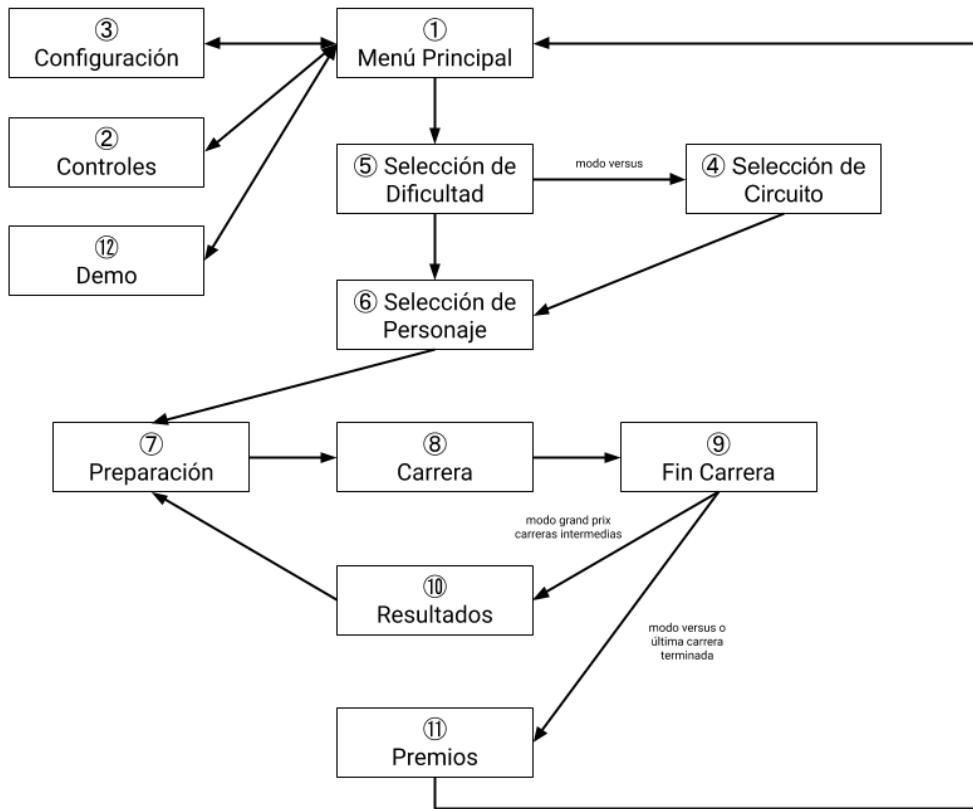


Figura 16. Mapa de navegación del videojuego.

La transición del “Menú Principal” a “Demo” se hace por tiempo de inactividad en dicha pantalla.

Además de las transiciones descritas en la Figura 16, en la mayoría de pantallas excepto las del juego (7, 8, 9, 10 y 11) se puede volver a la pantalla anterior utilizando la tecla “atrás” (ESC por defecto), en las pantallas de carrera (7, 8 y 9) es posible pausar el juego con la misma acción.

Pantallas del juego			
① Menú principal	② Controles	③ Configuración	④ Selección de circuito
⑤ Selección de dificultad	⑥ Selección de personajes	⑦ Preparación	⑧ Juego/Carrera
⑨ Fin de carrera	⑩ Resultados	⑪ Premios	⑫ Demo

Figura 17. Resumen de las pantallas del videojuego.

### 3. Detalles de implementación

#### 3.1. Lógica principal del juego y estados

El bucle principal del juego se encarga de realizar los cálculos necesarios y presentarlos al jugador al dibujarlos en la pantalla. El juego no debe depender del período de refresco: si se dibujan 60 imágenes por segundo, los coches deben correr de la misma forma que si se dibujaran 30 [1]. Es por ello que se presenta la siguiente lógica del juego (Figura 18), donde se introducen los conceptos de *update* convencional y *fixed update*.

Las actualizaciones, sobre todo de [Aspectos físicos](#), dependen del tiempo transcurrido desde la última actualización. Al hacer este tiempo variable, se ha de controlar el paso para asegurar que siempre use el mismo tiempo, haciendo los cálculos más correctos y fáciles de replicar.

```
time = clock()
while ( !quit ) {
    delta_time = clock() - time
    time = clock()

    update(delta_time)
    draw()
}

fixed_step = seconds(1) / framerate
elapsed_time = seconds(0)
last_time = clock()
while ( !quit ) {
    elapsed_time += clock() - last_time
    time = clock()

    while ( elapsed_time >= fixed_step ) {
        fixed_update(fixed_step)
        elapsed_time -= fixed_step
    }
    draw()
}
```

(a) Actualización por *update*.

(b) Actualización por *fixed update*.

Figura 18. Diferentes métodos de actualización del estado del juego. Solamente ilustran la diferencia entre los dos métodos, la lógica principal del juego necesita más pasos.

Por otro lado, si el tiempo empleado para actualizar y dibujar el estado del juego es mayor al tiempo de paso, se produce un bucle de realimentación donde cada vez se emplea más tiempo en cálculos. Para evitar esto, se limita el tiempo transcurrido entre dos iteraciones para que no suba por encima de un umbral. Si el juego es incapaz de realizar los cálculos a tiempo, el juego funciona a menos del tiempo real sin problemas.

Finalmente se introduce el concepto de **estado**. El juego tiene varios estados, y cada uno de ellos lleva control de un subconjunto de la configuración en curso. Las Figuras 17 (5 al 12) y 15 muestran varios ejemplos. Cada estado debe implementar su forma de actualizar y dibujar el juego utilizando su subconjunto de partes asignadas. Puede implementar ambos métodos de actualización *update* y *fixed update*, pero en la implementación realizada solo emplean uno de los dos.

La Figura 15 muestra una lista de todos los estados del juego. Existen dos estados para la gestión de la lógica general del juego, los cuales no implementan el método de dibujo y simplemente emplean otros estados: *InitLoad* y *RaceManager*.

### 3.2. Máquina de pila y flujos de información

Se introduce el concepto de pila de estados del juego, donde cada uno de sus componentes abarca un subconjunto de las variables del juego. El control del juego siempre es llevado por el estado de la cima de la pila. Las acciones de apilar o desapilar estados transfieren el control de unos a otros. Cada uno de los elementos de la pila controla una parte del juego, pero existen varios puntos comunes a los que todos pueden acceder. Es posible modificar variables de otro estado o acceder a variables comunes a través del patrón *Singleton*. Algunos ejemplos de esto último son la GUI (Figura 13) o funcionalidades del mapa (Figura 14). La Figura 19 muestra estos flujos de información en acción, concretamente cuando el estado del juego se encuentra en la animación inicial de una carrera.

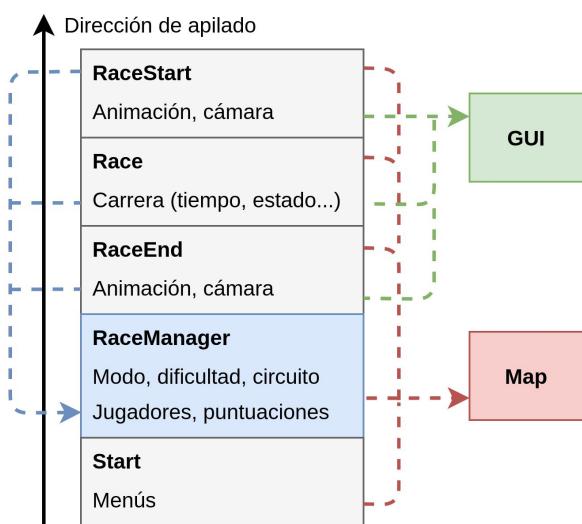


Figura 19. Flujos de información al iniciar una carrera.

El menú inicial, *Start*, apila un estado *RaceManager* con la configuración del conjunto de carreras a jugar. Para cada una de estas carreras (más de una si se encuentra en modo *Grand Prix*) se usan tres estados: uno de animaciones para el inicio, otro para la propia carrera y un tercer estado para el final.

Todos ellos hacen uso de los jugadores y puntuaciones del control central, y emplean diferentes clases *Singleton* para las funcionalidades comunes, ya sea leer/cargar datos del mapa o dibujar elementos comunes de la interfaz (GUI).

## 4. Aspectos gráficos

El juego simula un efecto tridimensional mediante el uso de diferentes proyecciones, cuyo objetivo es dar sensación de perspectiva y profundidad. La Figura 20 muestra los diferentes pasos a seguir para lograr un *frame* según el orden de dibujado:

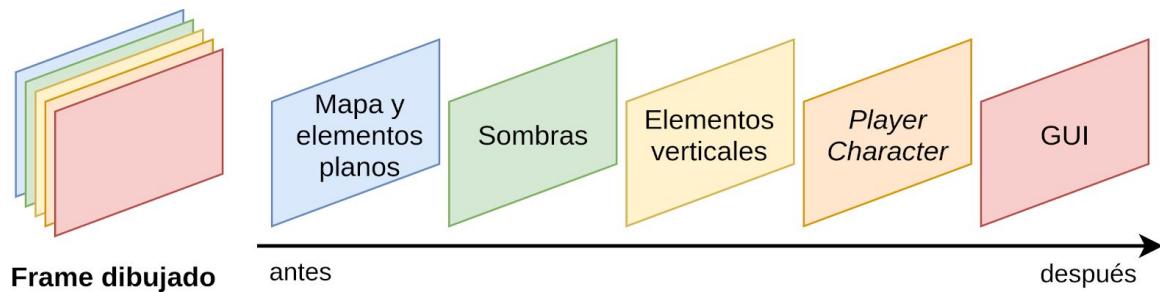


Figura 20. Capas que forman cada uno de los frames del juego.

En primer lugar se proyecta el mapa, un plano 2D, a la cámara, empleando una simulación del modo gráfico Mode7 presente en el hardware de la consola SNES. Posteriormente, se proyectan los diferentes elementos del circuito (tuberías, thwomps, otros corredores) junto con sus sombras. Finalmente se dibuja al propio corredor o *Player Character* y a todos los elementos de la interfaz gráfica o *GUI*.

### 4.1. Cámara 2.5D y Mode7



El primer elemento dibujado en la pantalla es el mapa. El mapa se divide en tres zonas: el propio circuito, del cual se hablará más adelante, y dos partes que forman el cielo. La Figura 21 muestra la división en tres piezas.

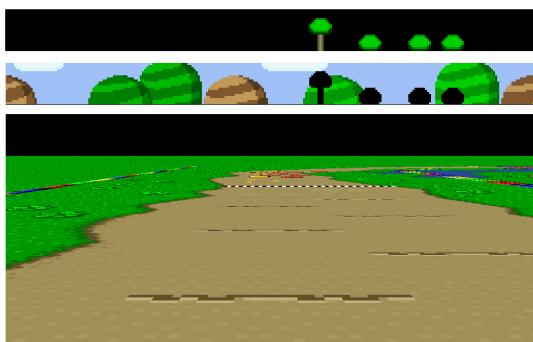


Figura 21. Partes que forman el mapa.

El cielo está formado por dos piezas que se mueven hacia la izquierda o la derecha según el giro de la cámara, en diferentes magnitudes para simular cercanía o lejanía.

Estas dos partes (“cerca” y “lejos”) sirven para dar un efecto de paralejo al jugador, aumentando la sensación 3D.

Para el circuito se emplea una simulación del hardware dedicado al Mode7. Este se encarga de proyectar un plano 2D al espacio del *frame* resultante, por ejemplo los circuitos de las Figuras 3 a 7.

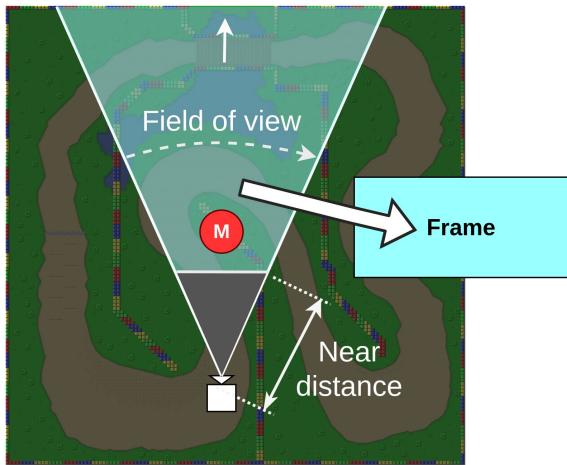


Figura 22. Proyección 2.5D con Mode7.

El muestreo se realiza sin ningún tipo de interpolación para seguir con la estética 8-bit del juego. En el algoritmo desarrollado, *near distance* marca la línea más cercana que será proyectada. Aumentar o disminuir su valor simula el efecto de subir o bajar la cámara, y es diferente del hecho de mover la cámara hacia adelante o hacia atrás.

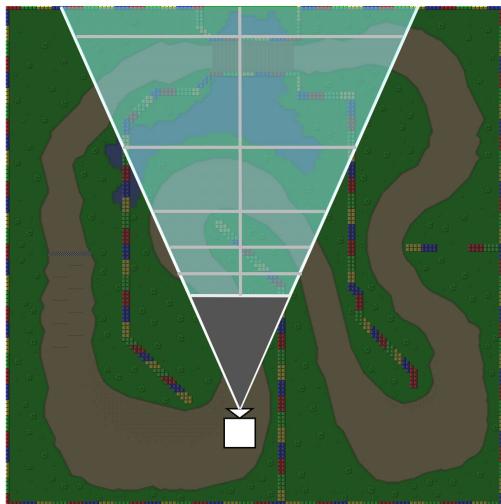


Figura 23. Proyección 2.5D con Mode7.  
Las zonas marcadas son cuadradas y de igual tamaño al ser proyectadas al frame.

La idea principal del Mode7 consiste en realizar un muestreo o *sampling* del plano 2D por cada píxel de la cámara. Para el píxel  $(u, v)$  del *frame*, su color es:

$$\text{frame}(u, v) = \text{plane}(\text{mode7}(u, v))$$

Donde  $\text{mode7}(u, v)$  es la función que calcula las coordenadas a muestrear del plano mencionado. La perspectiva se puede configurar con dos variables:

- *Field of view*
- *Near distance*

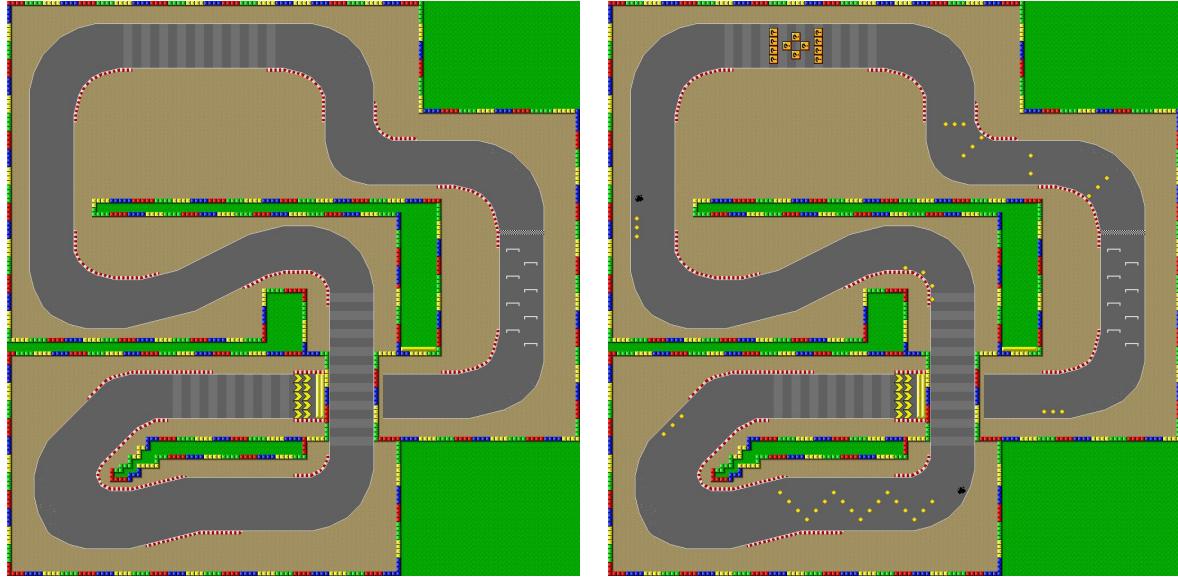
Para lograr el efecto de perspectiva, la relación entre el eje Y del *frame* y la profundidad del *sampling* no es directa sino inversamente proporcional, y es necesario ajustarla para que la proyección sea correcta y no muestre cierta curvatura.

Las zonas cercanas a la cámara son muestreadas con un mayor detalle que las zonas más lejanas. Cabe destacar que el "trapezoide" representado se extiende infinitamente, ya que teóricamente la fila superior del *frame* corresponde con puntos situados a profundidad infinita.

Finalmente, el *field of view* aumenta o reduce el rango de muestreo del eje proporcional a la profundidad.

Por último, ya que es posible muestrear puntos en el infinito o fuera del plano original, se introduce el concepto de "textura exterior" que consiste en otro plano repetido infinitamente en ambas direcciones el cual es muestreado en sustitución del original.

No obstante, todavía hace falta tener en cuenta más elementos que forman parte del circuito. Los **elementos horizontales** hacen referencia a objetos que usan la proyección planar para su dibujo (ver Figura 14). Algunos ejemplos son monedas (🟡) y cajas de objetos (🟠), como se puede ver en la Figura 24. La lógica para dibujarlos e interactuar con ellos se presenta en apartados posteriores.



(a) Sin elementos horizontales.

(b) Con elementos horizontales: monedas, cajas de objetos, *oil slicks*, *zippers* y rampas.

*Figura 24. Elementos horizontales que forman parte del mapa Mario Circuit 2.*

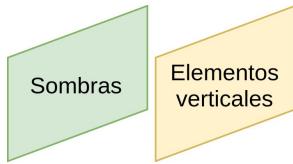
#### 4.2. Elementos horizontales

Los elementos horizontales deben aparecer sobre el suelo del circuito. Para ello, mediante el procedimiento explicado en el apartado de [Diseño y generación de niveles](#), se obtiene su tamaño y sus coordenadas sobre el asset del mapa.

Los *sprites* de cada elemento son superpuestos sobre la imagen del circuito actual. Sin embargo, algunos de estos objetos horizontales puede desaparecer como consecuencia de la interacción con los corredores. Por ello, es necesario conservar dos copias del asset del circuito: una con dichos elementos (Figura 24b) y otra sin ellos (Figura 24a), que mantenga la información original de los terrenos.

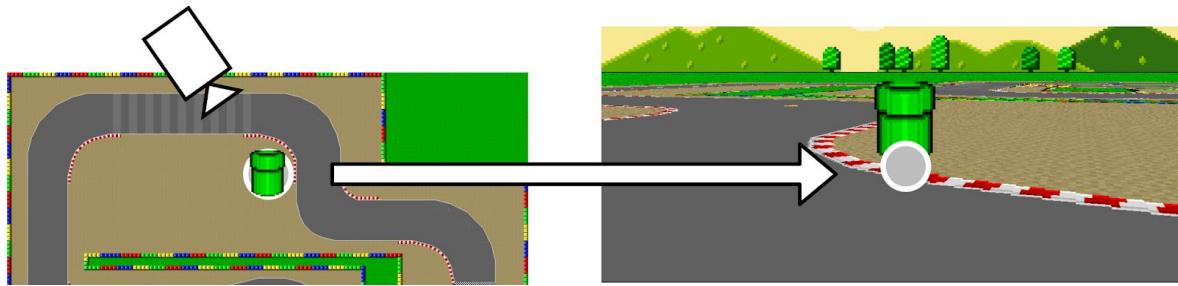
Los corredores pueden interactuar con estos elementos al circular sobre su zona del mapa. La comprobación de las colisiones se realiza sobre una matriz indexada con la posición de cada jugador. Véase el apartado de [Colisiones con elementos fijos](#) para más información. Cuando se interactúa con un elemento horizontal, éste puede ver su estado alterado. Los cambios necesarios, derivados de la modificación del estado interno, se recogen en una cola para ser aplicados al finalizar un ciclo de actualización completo. Este proceso modificará la copia del asset del mapa (Figura 24b) para mostrar por pantalla el nuevo estado.

#### 4.3. Elementos verticales



La otra parte principal que forma un *frame* son los elementos verticales y sus sombras. Los **elementos verticales** son las entidades que forman parte de la carrera y no forman parte del circuito o de la GUI: corredores, power-ups (plátanos, shells...), tuberías, *thwomps*, etc. (ver Figura 12).

Todos estos elementos tienen una posición 2D en el circuito. Para dibujarlos en el *frame* es necesario proyectar la posición en el circuito a la cámara. Solamente se proyecta dicho punto, no se proyecta la imagen entera. La Figura 25 muestra un ejemplo de este proceso:



(a) Espacio del plano (unidades de distancia). (b) Espacio de pantalla (píxeles).

Figura 25. Proyección del plano a la pantalla para dibujar elementos verticales.

La proyección de este punto es similar al Mode7, pero de forma inversa: en lugar de emplear píxeles para obtener coordenadas del plano, se transforman coordenadas del plano ( $x, y$ ) para obtener su posición la pantalla ( $u, v$ ):

$$\text{frame}(u, v) = \text{mode7}(\text{xy}_{\text{elemento}})$$

Cabe destacar que las coordenadas ( $u, v$ ) no tienen que corresponder perfectamente con un píxel concreto de la pantalla. El punto que representa la posición de cada elemento no corresponde con su centro. En concreto, para una imagen de ( $\text{width}, \text{height}$ ) píxeles se emplea el punto  $(\text{width} / 2, \text{height})$  ya que se supone que es el punto que toca el suelo.

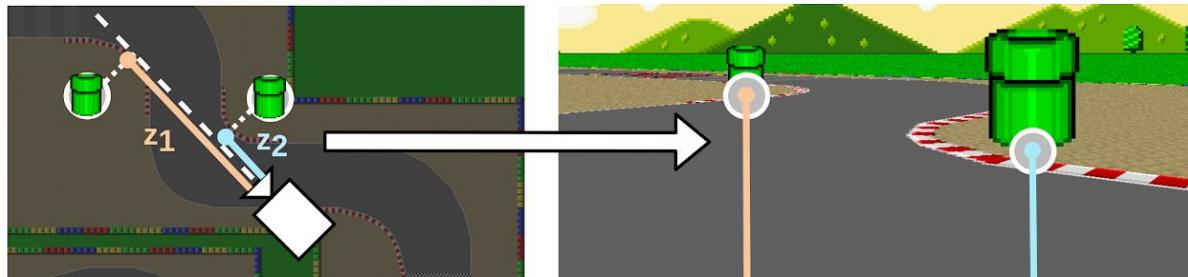
Como último detalle, cada elemento tiene un “radio visual”. El punto proyectado no es el de su posición, sino el punto que intersecta la recta que lo une con la cámara con una circunferencia de dicho radio y con centro en la posición del elemento. De esta forma, se logra cierta sensación de volumen al imitar las paredes de un cilindro en lugar de un plano giratorio.

Por otro lado, estos elementos son escalados: los más lejanos deben ser dibujados con menor tamaño que los más cercanos. Su tamaño depende de su profundidad (nótese que es diferente de su distancia a la cámara, ya que es la proyección de esta última en el eje de profundidad). La Figura 26 muestra un ejemplo de este escalado.

La escala sigue la siguiente relación dependiendo de su profundidad z:

$$\text{escala} \propto 1 / (3.5 * \log(1.02 + 0.8 * z))$$

Los tres parámetros 3.5, 1.02 y 0.8 ajustan la escala general y a profundidad 0 y su escalado, respectivamente.



(a) Profundidad en el plano.

(b) Escalado en pantalla. Nótese que la coordenada vertical de la imagen no corresponde exactamente con la profundidad, pero está directamente relacionada.

Figura 26. Escalado de los elementos verticales.

La mayoría de elementos verticales no varían su textura. Otros, como los corredores, cambian su representación dependiendo de su orientación y la de la cámara. En concreto, estos disponen de veintidós texturas diferentes para su rango de  $360^\circ$ . Realmente son menos, ya que algunas de ellas son volteadas para obtener su variante especular.

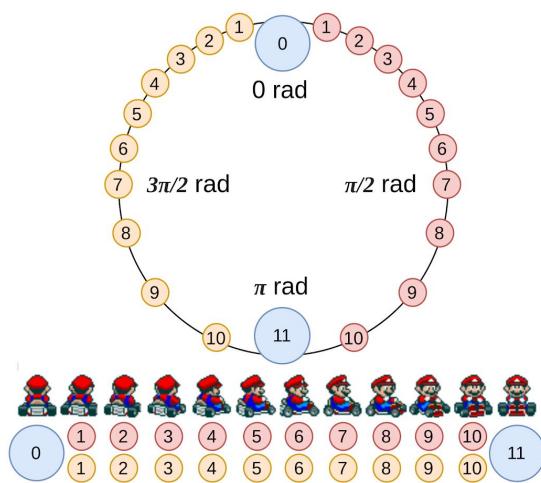


Figura 27. Relación ángulo de visión y texturas empleadas. Las texturas representadas con color naranja representan volteo horizontal.

La Figura 27 muestra la relación entre el ángulo de visión y la textura usada. Ya que los corredores suelen ser vistos desde detrás se ha incluido un mayor número de texturas para este rango de valores.

Las texturas número 0 y 11 no son volteadas ya que son perfectamente simétricas.

Dicho ángulo de visión depende de la posición y orientación de la cámara y el jugador. La Figura 28 ilustra los cálculos necesarios para obtener este ángulo.

$$\theta_{\text{sprite}} = \theta_{\text{driver}} - \theta_{\text{camera}}$$

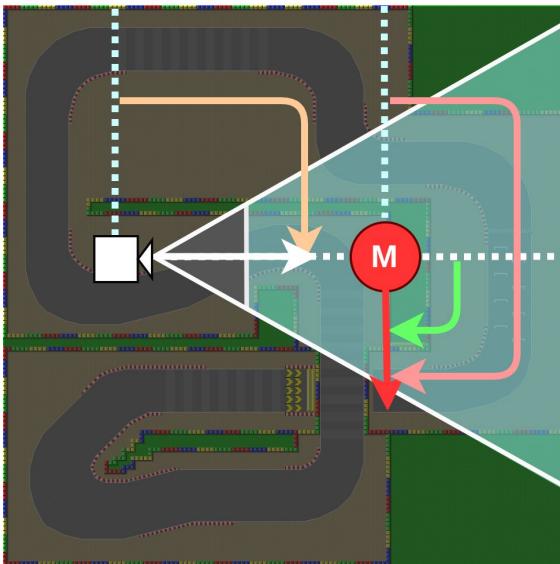


Figura 28. Cálculo de la orientación de los corredores y frame resultante.

La última pieza del sistema es la altura. Algunos elementos, como los corredores y rocas picudas o *thwomps* tienen esta tercera componente en su posición. Para dibujar un objeto con una determinada altura, se mueve hacia arriba en la coordenada Y del *frame*, siendo esta escalada con su profundidad. Como muestra la Figura 20, las sombras deben ser dibujadas al principio del todo para obtener una imagen correcta.

Los objetos con altura mayor que cero proyectan una sombra en el suelo, la cual es dibujada en su misma posición simulando iluminación cenital. Los elementos más altos proyectan una sombra más blanda, y los elementos más bajos proyectan una sombra más dura. Por otro lado, para simular perspectiva las sombras son escaladas con su profundidad: las sombras menos profundas aparecen más grandes, y las más profundas son encogidas en el eje vertical hasta formar una línea horizontal en el suelo. La Figura 29 muestra el resultado de ambas partes en ejecución.



Figura 29. Varios elementos verticales que proyectan sombra en el suelo. Los elementos más altos proyectan sombras menos opacas, y los elementos más lejanos proyectan sombras de menor tamaño y encogidas en el eje vertical para simular perspectiva.

#### 4.3.1. Efectos

El sistema de elementos verticales se ha empleado con varios fines. Además de servir para toda clase de Power-Ups o para el podio y los globos de la entrega de premios (Figura 17), también se ha utilizado para introducir diferentes efectos en el juego. Los **efectos** son partículas o animaciones especiales que siguen las mismas directrices, pero ignoran todo tipo de colisiones, sombras, etc. La Figura 30 muestra los efectos que han sido implementados:



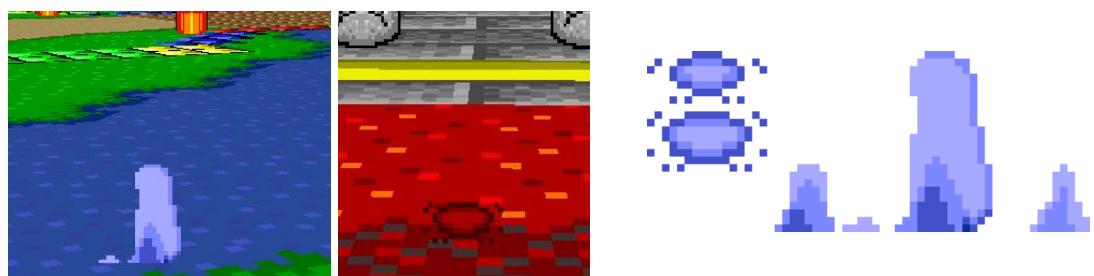
(a) Efecto al ganar monedas, en este caso al usar el Power-Up moneda.



(b) Efecto al perder monedas y efecto al colisionar con un caparazón rojo. El caparazón rojo rebota al colisionar con su objetivo. Los jugadores están mirando en la dirección contraria del circuito al encontrarse dando vueltas por la colisión.



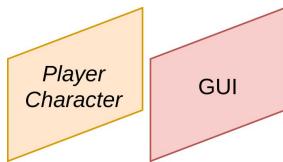
(c) Efecto de chispas. Ocurre en las colisiones con paredes o elementos verticales.



(d) Efecto de chapuzón. Ocurre cuando un objeto cae al agua o a la lava, pero no al vacío.

Figura 30. Efectos implementados y su aparición en el juego.

#### 4.4. GUI y jugador



La GUI, como se ha indicado al inicio del apartado [Aspectos Gráficos](#), es la última capa que se dibuja en pantalla para no ser oculta por ninguna otra. Está compuesta por Lakitu, los elementos de interfaz de usuario (contador de tiempo, posición en la carrera, etc.) y el jugador, dibujados en ese orden.

La interfaz del jugador consta de varios elementos: el indicador de *Power-Up*, el contador de tiempo de juego, el número de monedas y la posición del jugador en la carrera actual. En la Figura 31 están marcados todos éstos elementos.

Además de la interfaz “estática”, que está siempre en pantalla, también existen una serie de “efectos especiales” que se aplican en diferentes ocasiones: p. ej. al usar un *Power-Up* o interactuando con Lakitu. Finalmente, el minimapa se encuentra siempre en la mitad inferior de la pantalla.



Figura 31. Elementos de la GUI marcados en pantalla.

En la tabla de la Figura 32 se describen brevemente los elementos de la Figura 31.

Elemento	Descripción
1. Indicador de <i>Power-Up</i>	Muestra el último objeto especial recogido por el jugador, que todavía no ha sido utilizado.
2. Contador de tiempo	Indica el tiempo transcurrido desde el inicio de la carrera (Cuando la señal de salida es dada por Lakitu)
3. Posición en carrera	Indica la posición del jugador en la carrera actual.
4. Monedas	Muestra el número de monedas actuales del jugador.
5. Jugador	El sprite correspondiente al jugador
6. Lakitu	La entidad Lakitu, que actúa como “ayudante” de los corredores, en especial del jugador principal.

Figura 32. Elementos de la GUI.

Los “efectos especiales” nombrados anteriormente son los mostrados en la Figura 33.

Efecto	Descripción
Velocidad	<p>Se muestra en la parte superior de la pantalla (no en el minimapa) cuando el jugador utiliza el <i>Power-Up</i> del champiñón, para dar una mayor sensación de aumento de la velocidad.</p> 
Relámpago	<p>El efecto consiste en mostrar frames blancos durante un periodo corto de tiempo simulando un relámpago. Este efecto se aplica cuando cualquier jugador utiliza el <i>Power-Up</i> del rayo.</p> 
Barrido negro	<p>La parte superior de la pantalla (sin incluir el minimapa) se funde a negro. Se utiliza para no mostrar cambios bruscos en pantalla al jugador, por ejemplo, cuando Lakitu recoloca en el circuito al jugador.</p> <p>Este “efecto”, internamente, no es considerado un efecto especial como los dos anteriores.</p> 

Figura 33. Efectos especiales de la GUI.

#### 4.4.1. Lakitu

Lakitu es una entidad especial que actúa como “moderador” de la carrera indicando la señal de salida, el número de vueltas, si el jugador va en dirección contraria y la bandera de fin de carrera. Además de ayudar al jugador a volver a una posición segura cuando se cae al vacío, lava o agua. En la Figura 34 se resumen los diferentes estados de lakitu. Todos ellos aparecen en pantalla realizando un movimiento diferente en base a su acción.

Acción	Descripción
Señal de Salida 	Aparece para dar comienzo a cada carrera una vez finalizada (u omitida) la animación de presentación del circuito. Cuando el semáforo se enciende por completo la carrera da comienzo y Lakitu se retira.
Indicador de Vueltas 	Cada vez que el jugador pasa por meta Lakitu aparece desde la izquierda a la derecha de la pantalla con un cartel mostrando el número de vuelta en la que se encuentra ahora. En caso de ser la última vuelta el cartel mostrará un mensaje de "Final Lap".
Dirección Contraria 	Si el jugador comienza a conducir en dirección contraria, Lakitu aparecerá moviéndose de izquierda a derecha por el centro de la pantalla superior hasta que el jugador vuelva a avanzar en la dirección correcta.
Meta final 	De la misma manera y realizando el mismo recorrido que en la acción "Indicador de Vueltas", Lakitu aparecerá con una bandera de meta al cruzar la meta por última vez.
Recolocar Jugador 	Aparece cuando un jugador se cae a terrenos como vacío, agua o lava, donde no puede conducir, para recolocarlo en el circuito. Se muestra con el jugador enganchado a la caña de pescar desde la parte superior de la pantalla bajando hasta soltar al conductor a media altura.

Figura 34. Acciones de Lakitu.

#### 4.4.2. Jugador

El jugador principal, a diferencia de los demás conductores (IA), se dibuja en pantalla como podemos observar en la Figura 31. Esto se debe a que el jugador tiene animaciones y características más elaboradas que los demás conductores.

Todas estas animaciones se recogen en una clase dedicada, «DriverAnimator», que controla el sprite del conductor principal que se muestra por pantalla. Las posibles posiciones que se representan del jugador son las mostradas en la Figura 35. Téngase en cuenta que estos sprites son espejados para los giros en dirección contraria.

Frontal	Giro 1	Giro 2	Giro 3	Derrape
				
0 rad/ut	< 0.25 rad/ut	< 0.5 rad/ut	> 0.5 rad/ut	> 40% del máx.

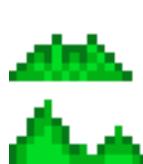
Figura 35. Sprites frontal, giros y derrape del jugador principal con sus velocidades (radianes por unidad de tiempo) angulares correspondientes.

Además de representar el sprite del jugador, también se muestran en pantalla partículas de derrape según en el terreno en el que se encuentre el jugador. Además, un segundo tipo de partículas es mostrado cuando se conduce por terrenos lentos como la hierba. Véase dos ejemplos en la Figura 36 que acompaña este párrafo. La lista completa de partículas se recoge en la Figura 37.



Figura 36. Partículas de derrape (izquierda) y sobre hierba (derecha).

También cabe destacar que, al conducir, el sprite se mueve ligeramente arriba y abajo para dar dinamismo al corredor. Otro movimiento similar, pero de izquierda a derecha, se aplica para dar mayor sensación de derrape.

Derrape Tierra	Derrape Asfalto	Derrape Hierba	Otros derrapes*	Terreno Hierba
				

\*Los terrenos que no son genéricos (tierra, asfalto, hierba) cambian el color de la partícula de derrape blanca por el que corresponda según el color del terreno, esto ocurre en los circuitos "Ghost Valley 1" y "Rainbow Road".

Figura 37. Partículas del jugador principal para terrenos y derrapes.

#### 4.4.3. Minimap

El minimapa se muestra durante la carrera en la parte inferior (mitad de la pantalla) visualizando el circuito por completo y todos los conductores. Para ello se utiliza el mismo sistema «Mode7» explicado en la sección [Cámara 2.5D y Mode7](#). Sin embargo, se establecen parámetros diferentes para situar la cámara arriba y modificar la perspectiva ligeramente. Igualmente, para la proyección de los jugadores en el minimapa se utiliza el mismo principio.

La orientación de los conductores en el minimapa no corresponde con la que se calcula en la zona de circuito (pantalla superior). En este caso la cámara es fija, por lo que se muestra la orientación real de los conductores y no la relativa al jugador principal. Véase la correspondencia aplicada, según la Figura 27, en el ejemplo de la Figura 38.

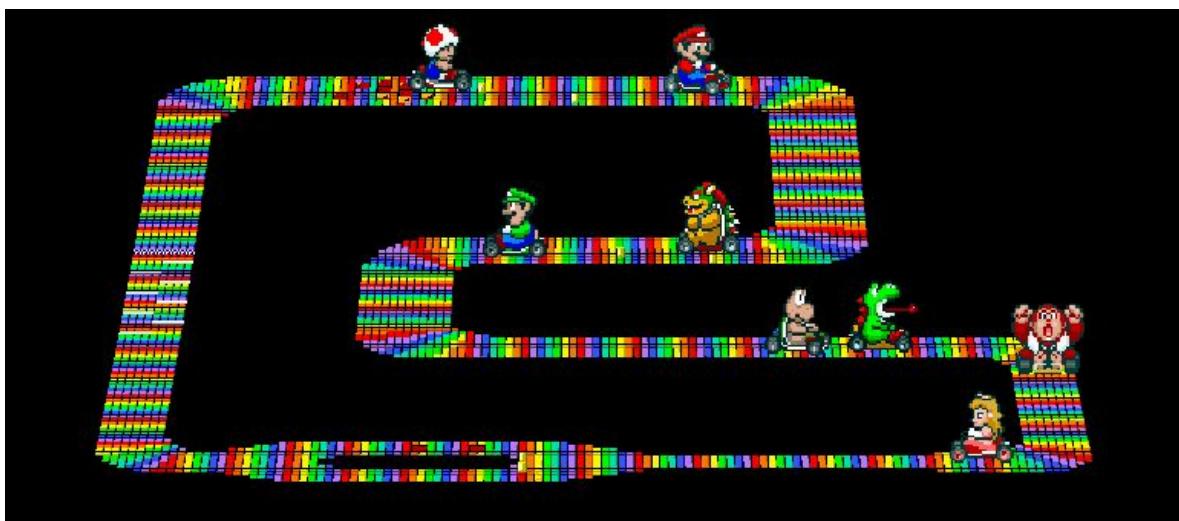


Figura 38. Parte inferior de la pantalla mostrando el minimapa en el circuito “Rainbow Road”.

#### 4.5. Cinemáticas

El modelo explicado para lograr un *frame* ([Aspectos gráficos](#)) establece cómo obtener una imagen a partir de los datos de la cámara y los elementos que forman parte del circuito. En [Aspectos físicos](#) se analizan las interacciones entre los elementos del circuito.

En el juego normal, la cámara está fijada a un corredor, situándose a una cierta distancia por detrás. Sin embargo, la cámara permite crear animaciones más complejas si es desacoplada del jugador principal. Esto permite cierto movimiento libre. A partir de la Figura 20, es posible dejar de dibujar al *Player Character* para pasar a dibujarlo como un elemento vertical más. Gracias a esta característica, se han creado diferentes cinemáticas en el juego, donde el principal componente atractivo son los movimientos de la cámara, con posiciones y orientaciones poco convencionales en una partida normal.



(a) Preparación de carrera.

(b) Fin de carrera.

(c) Modo demo.

Figura 39. Cinemáticas implementadas.

Como se ve en la Figura 39, existen tres cinemáticas principales en el juego:

- **Preparación de carrera:** Contiene varios planos del circuito en el que se correrá a continuación, permitiendo al jugador familiarizarse con él y mostrándole sus elementos principales.
- **Fin de carrera:** Enfoca al jugador desde otro ángulo. Le informa de que la carrera ha terminado y ya puede dejar de conducir. Además, le otorga cierto tiempo de descanso hasta afrontar la siguiente carrera.
- **Modo demo:** Se fija en un corredor y va cambiando de objetivo cada poco tiempo con un movimiento suave de la cámara. No está orientado a mostrar lo que cada corredor tiene delante como si fuera una carrera, sino que recrea planos del circuito desde diferentes posiciones y orientaciones. Mediante entrada de teclado se puede configurar la posición de la cámara. Por defecto se usan las teclas:
  - **TAB:** Permite cambiar de jugador objetivo al que enfoca la cámara.
  - **CTRL:** Activa o desactiva los cambios automáticos de objetivo de la cámara.
  - **Barra espaciadora:** Conmuta entre el modo de primera y tercera persona (cámara fija y cámara libre).
  - **ALT + Números 1-7:** Permite probar los diferentes Power-Ups forzando a los corredores a usar el ítem seleccionado. En orden: champiñón, moneda, estrella, cáscaras de plátano, caparazón verde, caparazón rojo y trueno.

## 5. Aspectos físicos

La aplicación cuenta con una colección de modelos implementados, que definen el motor de físicas del juego. Estos componentes deben ser capaces de simular de manera verosímil la conducción de varios corredores, con características diferentes, sobre una serie de terrenos distintos en función de la zona del mapa en la que se encuentren. Además deberán detectar y reaccionar ante las colisiones entre los distintos elementos presentes en el circuito.

### 5.1. Vehículos

Se ha diseñado e implementado un modelo físico que permite simular los parámetros de aceleración, velocidad, manejo y peso, descritos para cada grupo de personajes en la Figura 10 del apartado sobre [Personajes y vehículos](#). Las especificaciones de cada vehículo se resumen en cuatro parámetros numéricos:

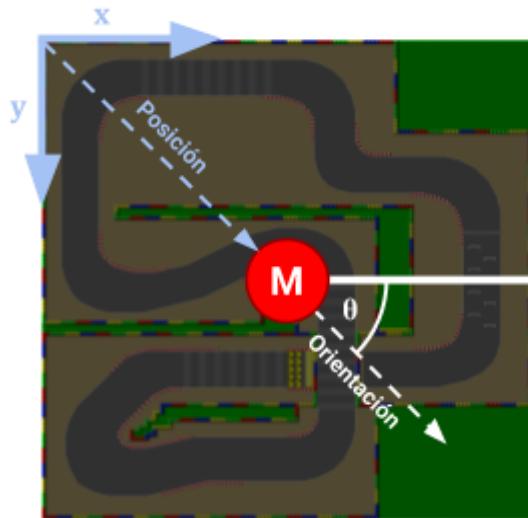
- **Aceleración:** en función de la potencia que pueda desarrollar el motor del vehículo simulado, la aceleración lineal podrá ser mayor o menor.
- **Manejo o aceleración de giro:** para regular el control que el jugador tiene sobre el vehículo (manejo), determina el valor de la aceleración angular. Este parámetro también limitará el máximo de la velocidad angular del vehículo.
- **Velocidad máxima:** hace referencia al mayor valor de velocidad lineal que un conductor podrá alcanzar en función de su vehículo. Aunque el jugador continúe acelerando, nunca se sobrepasará este valor en condiciones normales. El límite podrá ser superado como consecuencia de un Power-Up que lo permita.
- **Peso:** además de afectar indirectamente a la aceleración de su vehículo, tiene consecuencias durante las colisiones, como se verá en el apartado correspondiente sobre [Colisiones con elementos móviles](#).

Los valores asignados a cada grupo de personajes, sin unidades conocidas, son los mostrados en la Figura 40.

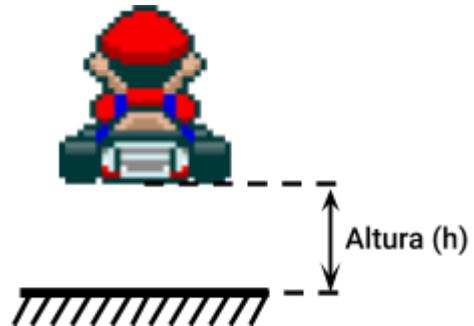
	BALANCEADOS	POTENTES	ÁGILES	PESADOS
<b>Aceleración</b>	0.11	0.15	0.12	0.1
<b>Manejo</b>	0.14	0.13	0.15	0.13
<b>Velocidad</b>	0.10200	0.10175	0.10150	0.10250
<b>Peso</b>	1.0	1.0	0.5	2.0

Figura 40. Valores asignados a los parámetros físicos de los vehículos.

Durante la carrera cada corredor tiene cuatro grados de libertad en su movimiento, los cuales quedan representados en la Figura 41.



(a) Posición (X, Y) y orientación ( $\theta$ )



(b) Altura (h) sobre el nivel del suelo

Figura 41. Grados de libertad de los corredores.

Los valores de la Figura 41 se almacenan como:

- **Posición (x, y):** un vector de dos componentes, coordenadas en el eje X y en el eje Y desde el origen, situado en la esquina superior izquierda de los circuitos. En  $[0.0, 1.0]$ .
- **Orientación ( $\theta$ ):** ángulo de rotación respecto a la horizontal del eje X. En  $[0, 2\pi]$ .
- **Altura (h):** distancia corredor-suelo en el eje Z del plano del circuito. En  $[0, \infty)$ .

Estos valores son calculados en cada actualización del estado, teniendo en cuenta los parámetros descritos anteriormente y sus derivados:

- **Velocidad hacia delante (v):** un corredor tendrá asociada en todo momento una cierta velocidad lineal. Según la Figura 42, esta velocidad se modifica de varias maneras.



Figura 42. Factores que influyen en la velocidad hacia delante.

- **Aceleración ( $a_m$ ):** cuando el vehículo acelera aumenta su velocidad a razón de:

$$v_f = v_o + a_m \cdot t$$

Sin embargo, la aceleración no es constante. En su lugar se pretende simular las funciones de aceleración mostradas en la Figura 43. De este modo, el jugador alcanza rápidamente una velocidad adecuada si su vehículo se encuentra inicialmente detenido, pero necesita más tiempo para lograr su velocidad máxima.

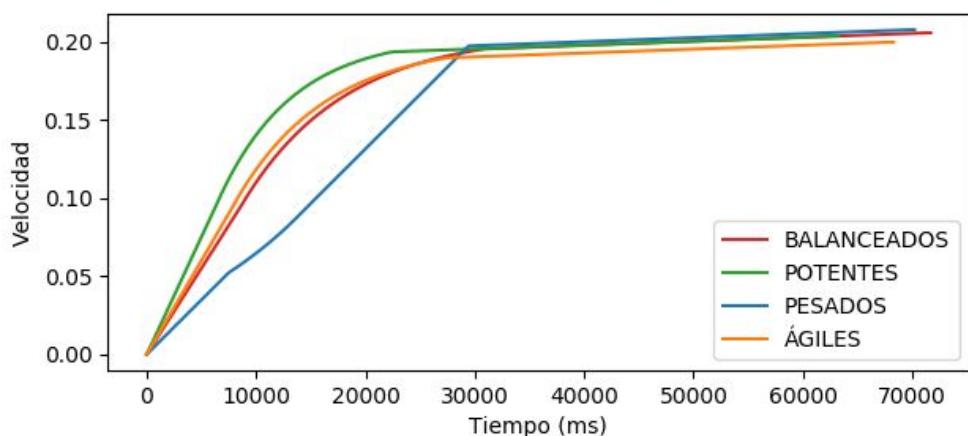


Figura 43. Gráfica de simulación de aceleración según grupo de personajes.

- **Frenado ( $a_f$ ):** los corredores pueden hacer uso de sus frenos, aplicándose una aceleración negativa al movimiento del vehículo que conducen.
- **Rozamiento ( $a_r$ ):** si se deja de acelerar, el vehículo se detendrá lentamente como consecuencia del rozamiento. Terrenos lento, como la hierba o la arena, aplican un mayor rozamiento cuando la velocidad supera cierto umbral. El objetivo es ralentizar a quienes circulan por fuera de los límites de la pista.
- **Otros modificadores:** los choques con otros jugadores o con elementos del circuito modificarán igualmente la velocidad lineal. Algunos Power-Ups tienen también efectos positivos o negativos sobre este valor. Finalmente, durante el giro, los vehículos ven reducida su velocidad lineal.
- **Velocidad de giro ( $\omega$ ):** esta variable física representa la derivada frente al tiempo del giro que efectúa un corredor. Según el convenio establecido, si observamos el mapa cenitalmente como en la Figura 41a, valores positivos de la velocidad de giro corresponden con giros a derechas (sentido horario). Por tanto, cuando el corredor gire a izquierdas (sentido anti horario) acumulará velocidad de giro negativa.
  - **Aceleración angular ( $\alpha$ ):** por debajo del 40% de la máxima velocidad, un vehículo aumentará su velocidad de giro de manera lineal con un coeficiente reductor ( $k = 0.15$ ) aplicado a la aceleración angular.

$$\omega_f = \omega_o + k \cdot \alpha \cdot t$$

- **Derrapaje:** cuando se supera el umbral del 40% se inicia el derrape y la velocidad angular aumenta sin coeficiente reductor (i. e.  $k = 1$ ) hasta alcanzar el máximo valor permitido por los parámetros del vehículo ( $\omega_{max} = 10 \cdot \alpha$ ). También es posible forzar el derrapaje de modo manual, mediante la correspondiente tecla («drift»).
- **Velocidad hacia arriba ( $v_h$ ):** en ciertas situaciones un corredor puede recibir un impulso que le haga ganar velocidad en el eje Z del plano y, en consecuencia, elevarse sobre el nivel del suelo. En este caso, surgirá una fuerza constante y hacia abajo (pseudo-gravedad, g) que le hará regresar al suelo.

$$v_h' = v_h + g \cdot t$$

Quedan explicadas todas las variables contempladas en el motor de físicas de los vehículos. Las fórmulas aplicadas para la actualización del estado de cada conductor, tras un lapso de tiempo de  $\Delta t$  segundos, son las siguientes:

1. Para la actualización de la posición, se calcula la distancia lineal recorrida,

$$\Delta s = v_o \cdot \Delta t + \frac{1}{2} \cdot (a_m + a_r + a_f) \cdot \Delta t^2$$

a continuación se actualizan las coordenadas aplicando las razones trigonométricas del ángulo de orientación del corredor,

$$x_f = x_o + \Delta s \cdot \sin \theta_o$$

$$y_f = y_o + \Delta s \cdot \cos \theta_o$$

2. Para la actualización de la orientación, se aplica la simplificación:

$$\theta_f = \theta_o + \omega_f \cdot \Delta t$$

3. Finalmente, para la actualización de la altura, se toma:

$$h_f = h_o + v_h \cdot \Delta t + \frac{1}{2} \cdot g \cdot \Delta t^2$$

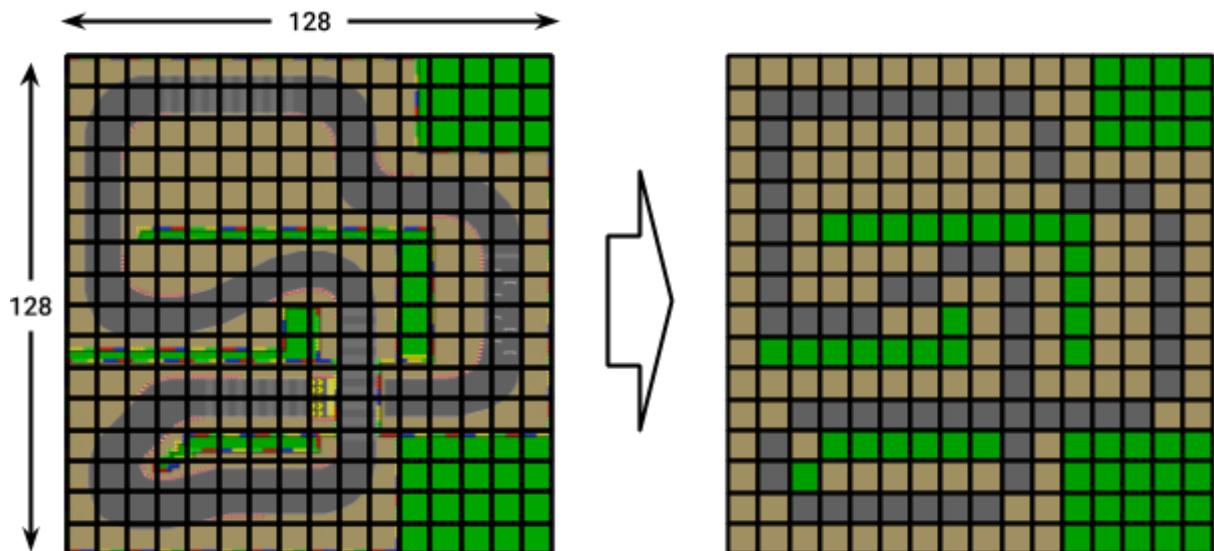
En casos especiales (p. ej. choque, obstaculización o acción de *Power-Ups*) los valores se modifican siguiendo las lógicas específicas de cada evento. Consulte los detalles en los apartados relativos a dichos sucesos.

## 5.2. Mapa de terrenos

Todos los circuitos cuentan con dos tipos de terrenos por los que es posible circular. Por un lado existen zonas de terreno normal o pista de carreras, por las cuales los corredores pueden circular a máxima velocidad. Por otro lado, las zonas de terreno lento, situadas en el exterior de la pista, ralentizan el avance de los corredores. Además, ciertas zonas de los mapas puede

ser lagos de agua o de lava. Tanto estas zonas como las caídas al vacío provocarán que el jugador sea transportado hasta una posición segura (véase [Recolocación de corredores](#)).

Para gestionar la lógica de los diferentes terrenos se hace uso de una matriz de 128x128 celdas en las que se almacena el tipo y naturaleza de la zona. Cada mapa se divide entonces en casillas del mismo tamaño, como se muestra en la Figura 44.



*Figura 44. Ejemplo simplificado de la matriz de terrenos.*

A diferencia de la simplificación de la Figura 44, la matriz real es capaz de definir los terrenos del mapa con una precisión mucho mayor. Nótese que los assets utilizados para la carga de los circuitos tienen 1024x1024 píxeles. Es decir, cada una de las celdas de la matriz de terreno representa un cuadrado de 8x8 píxeles del mapa.

A partir de esta estructura de datos, el sistema de [Físicas de vehículos](#), explicado en el apartado anterior, es capaz de determinar el tipo de terreno por el que circula cada corredor. Dada la posición (x, y) de un jugador, se indexa la matriz de terrenos en base a:

$$\text{MapaTerrenos}[\text{floor}(y \cdot 128)][\text{floor}(x \cdot 128)]$$

ya que el rango de las coordenadas de posición se encuentra en [0.0, 1.0].

### 5.3. Colisiones

El último componente del motor de físicas es el encargado de gestionar los choques que se producen en el mundo simulado como parte del videojuego. La naturaleza de las colisiones depende del tipo de cuerpos que intervienen en ellas.

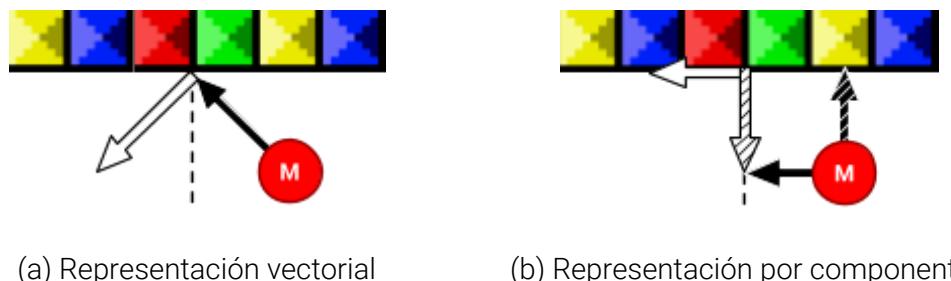
#### 5.3.1. Con elementos horizontales

Los objetos horizontales tienen la peculiaridad de estar colocados sobre el plano que sirve de suelo al circuito. Además, en relación al asset del mapa, todos los elementos tienen una extensión múltiple de 8 píxeles. Para aprovechar estas características, las colisiones con

estos elementos se realizarán a través de la estructura de datos descrita en el apartado anterior para el [Mapa de terrenos](#).

Existen siete tipos de elementos horizontales en cuanto a su comportamiento. Cada uno de ellos constituye una clase diferente en el diagrama de la Figura 14, exceptuando los muros.

- **Muros (■■■■)**: se registran en el mapa de terrenos como un nuevo tipo , «BLOCK», de modo que si un corredor va a moverse hacia una casilla de esta categoría se calcula su rebote. La dirección y módulo de la fuerza de reacción se estima de forma diferente para paredes longitudinales que para paredes transversales.



*Figura 45. Efecto de rebote contra una pared transversal.*

En la Figura 45, se ilustra el rebote ocasionado por el choque con una pared transversal, que se extendía a lo largo del eje X del mapa. En este caso, atendiendo a la representación vectorial, la velocidad de incidencia (vectores en negro) se traduce en una velocidad de rebote (vectores en blanco). Esto se consigue cambiando de sentido la componente perpendicular a la pared (marcada en la Figura 45b). En el caso de las paredes longitudinales (eje Y) se realiza la transformación sobre la componente correspondiente. Manteniéndose inalterada la componente opuesta.

- **Charco de aceite (●), rampa transversal (▬), longitudinal (▬) y zipper (▲)**: todos estos elementos horizontales tienen un efecto directo sobre los corredores que los pisan y nunca desaparecen del mapa. Cada uno de ellos se registra en el mapa de terrenos con un identificador que referencia su tipo, «OIL\_SLICK», «RAMP\_HORIZONTAL», «RAMP\_VERTICAL» y «ZIPPER», respectivamente. Cuando el motor de físicas detecta estas categorías de terreno ejecuta las acciones asociadas.
- **Monedas (🟡) y cajas de objetos (🟠)**: a diferencia de los anteriores, estos elementos desaparecen o cambian cuando son activados por un corredor. Por ello es necesario mantener una referencia a la instancia exacta de cada *FloorObject*. En el mapa de terrenos aparecen reflejados como «OTHER». Ante una casilla de este tipo, el motor de físicas comprueba con qué entidad concreta ha interactuado el jugador y, en caso necesario, aplica las actualizaciones oportunas al estado de los elementos del juego. Una vez detectada la colisión, la identificación de la entidad se realiza comparando la posición del corredor con la posición y dimensiones de los *FloorObjects* especiales.

### 5.3.2. Con elementos verticales

Como se ha visto en el apartado de [Elementos verticales](#), cada uno de estos está definido por un punto. Este punto supone el centro de un círculo (*hitbox*) con un radio que depende del tipo de elemento. A partir de esto, el problema de encontrar colisiones entre dos objetos se puede reformular a encontrar pares de círculos que intersecan entre sí. Sin embargo, si hay una gran cantidad de elementos en el mapa, encontrar los pares de círculos mediante fuerza bruta no es la mejor opción en cuanto a eficiencia.

Una alternativa para resolver este problema es extender la matriz de colisiones con elementos horizontales, de forma que las colisiones se realicen en dos partes: “registro” de los objetos y posterior “consulta” para varios elementos. De esta forma, la estructura auxiliar es optimizada para búsquedas más rápidas y se evita la realización de consultas “todos con todos”. Algunos objetos cambian su posición con frecuencia, lo que fuerza a actualizar la estructura auxiliar.

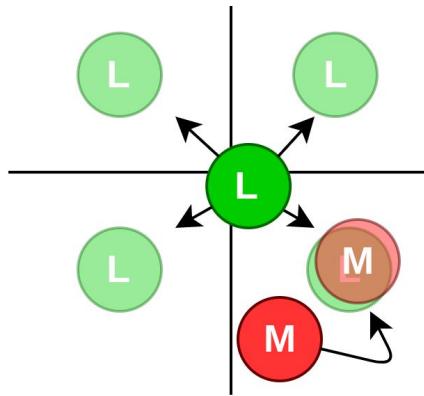


Figura 46. Reparto de los jugadores en contenedores. Algunos pueden pertenecer a diferentes contenedores.

La estructura auxiliar propuesta es separada a la matriz de colisiones, y se basa en *hashing espacial* para encontrar círculos cercanos rápidamente [2]. Existen varios contenedores, y cada posición puede ser rápidamente relacionada con uno de ellos (Figura 46).

Ya que una entidad puede estar en más de un contenedor a la vez, es posible que se registre en múltiples de ellos. Finalmente, para detectar las colisiones de un elemento solo hace falta mirar en los contenedores que lo contendrían, reduciendo el número de combinaciones al comprobar.

Posteriormente, para calcular la colisión entre dos elementos se emplean tres variables:

- **Momento:** Vector, encargado de mover al jugador en una dirección según el choque.
- **Multiplicador de ralentización:** Reduce la velocidad del jugador en un cierto porcentaje (algunos choques frenan al jugador completamente, como los caparazones rojos, otros permiten que el jugador tenga algo de velocidad).
- **Efectos especiales:** Son opcionales, y permiten aplicar cambios a la [Gestión de estados del personaje](#) o al componente encargado de animar al [Jugador](#). Por ejemplo, el caparazón rojo hace al jugador girar durante un breve periodo, o el aplastamiento por *thwomp* aplana al jugador durante un tiempo.

Todas las colisiones con elementos del entorno o Power-Ups envían al jugador en la dirección que conecta sus dos posiciones. En el caso general, dicho vector es escalado según la velocidad del usuario, para simular su rebote. El caso de colisión entre dos jugadores es más

complicado: el momento depende de los pesos y velocidades de ambos corredores, intentando dar ventaja a los coches más pesados.

#### 5.4. Gestión de estados del personaje

Además de todas las variables físicas descritas en el apartado sobre los [Vehículos](#), el estado de los corredores puede verse alterado por efectos derivados de los distintos *Power-Ups* disponibles en el juego. Cada uno de ellos tiene asociado uno o varios estados:

Estado	Causas posibles	Duración
SPEED_UP	//	1,5 segundos
MORE_SPEED_UP	//  +	0,75 segundos
STAR		10 segundos
SPEED_DOWN		10 segundos
UNCONTROLLED	//  //  //	1 segundo

Figura 47. Causas de los estados especiales de un corredor.

En la Figura 47 se presentan los estados en los que se puede encontrar un corredor a lo largo de una carrera. Los estados no son excluyentes, de modo que un mismo jugador puede encontrarse con dos o más activados simultáneamente. Siempre que uno de ellos está activo, se aplican sus efectos positivos y/o negativos:

- **SPEED\_UP:** el corredor goza de un límite de velocidad mayor que el de su vehículo.
- **MORE\_SPEED\_UP:** combinando dos estados «SPEED\_UP» se consigue un mayor beneficio en la velocidad punta del vehículo.
- **STAR:** el jugador es invencible frente a choques y terrenos lentos.
- **SPEED\_DOWN:** la velocidad máxima del vehículo se reduce temporalmente.
- **UNCONTROLLED:** el conductor pierde el control del vehículo.

Algunos de estos estados se puede desactivar antes de que expire su duración a causa de otros eventos que fuerzan la pérdida de las ventajas o desventajas otorgadas.

Para implementar la gestión de estos estados se ha optado por tratar cada uno de ellos como una máscara de bits. Todas las posiciones del vector valen cero excepto los que representan estados activos.

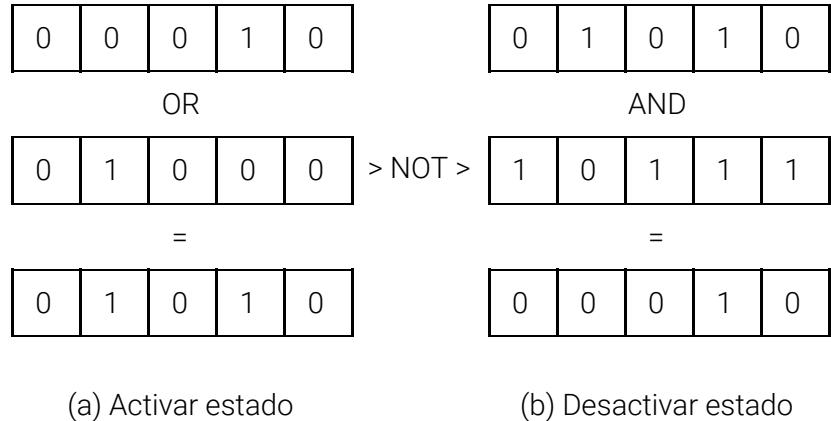


Figura 48. Activación y desactivación de estados mediante máscaras de bits.

En la Figura 48 se presentan dos ejemplos de uso de máscaras de bit para editar el estado de un corredor. Si se desea activar un efecto, se aplica la operación de «OR» bit a bit entre la máscara del estado actual y la máscara del nuevo estado. En caso de desechar desactivarlo, se debe calcular el «AND» del estado actual y el negado del estado a desactivar. Para las operaciones de consulta, se comprueba el resultado del «AND» entre ambos estados.

Complementariamente, existe otra estructura de datos que almacena el momento en el que se ha de desactivar cada uno de los estados. Al comienzo de cada actualización del corredor se comprueba si existe algún estado especial listo para ser desactivado y se limpia dicho bit de la máscara.

## 6. Aspectos de audio

### 6.1. Gestor de sonido

Para facilitar la abstracción durante la gestión del audio del juego se ha diseñado e implementado una clase, en forma de *Singleton*, que almacena referencias a todos los sonidos que se pueden reproducir o que están sonando en un momento determinado. La interfaz ofrecida permite comenzar, pausar o detener la reproducción tanto de música de fondo como de efectos de sonido (SFX).

### 6.2. Música

La música está presente durante todo el juego, ya sea en menús, o durante las carreras. Cada circuito tiene asociado un tema diferente. Además, según la clasificación en la que se acabe

un circuito, la música reproducida al jugador será triste o alegre, conforme la situación lo requiera. El listado de audios no relacionadas con los circuitos son las siguientes:

- **Menú principal:** Suena durante todos los menús disponibles antes de comenzar una partida (configuración, controles, selección de dificultad, circuito, etc.)
- **Selección de personajes:** Se reproduce durante la selección de personajes tras elegir el circuito y/o dificultad, antes de comenzar la carrera.
- **Presentación del circuito:** Durante la cinemática de «Preparación de carrera» se reproduce esta melodía introductoria.
- **Ganador de circuito:** En caso de quedar entre la primera y cuarta posición, al acabar la carrera durante la cinemática de «Fin de carrera» suena esta música de victoria.
- **Perdedor de circuito:** En caso de quedar entre la quinta y octava posición, al acabar la carrera durante la cinemática de «Fin de carrera» suena esta música de derrota.

### 6.3. Efectos de sonido o SFX

Los efectos de sonido son audios de pequeña duración que ayudan a completar la sensación de inmersión del jugador. La lista de SFX incorporados en el juego es extensa y puede ser consultada a continuación.

- **«Ding» de Nintendo:** suena al cargar el juego, cuando se muestra el logotipo.
- **Menús:** al cambiar la selección de un menú, confirmar una selección o retroceder dentro del árbol de menús.
- **Lakitu:** indicadores de inicio de carrera (semáforo), aviso de última vuelta, de dirección errónea y de fin de carrera.
- **Colisiones:** colisión con paredes o con objetos verticales.
- **Motores:** merece mención aparte.
- **Sonidos de victoria y derrota:** si el jugador termina la carrera entre las tres primeras posiciones o en posiciones posteriores.
- **Derrape:** cuando el jugador supera el 40% de su velocidad angular máxima.
- **Frenado:** al presionar la tecla de frenado.
- **Salto y aterrizaje:** al elevarse y volver al suelo.
- **Caídas:** al vacío, a un lago de agua o de lava.
- **Aplastamiento:** al ser alcanzado por un Thwomp en estado de bajada.
- **Empequeñecimiento y recuperación de tamaño:** a causa de un rayo.

- **Monedas:** ganadas o perdidas.
- **Objetos especiales:** obteniendo un objeto aleatorio, cuando se obtuvo el objeto, al lanzar el objeto o al depositarlo sobre el terreno. También al utilizar un champiñón o un zipper para ganar velocidad. En el momento que algún jugador hace uso de un rayo, o cuando el jugador es perseguido por un caparazón rojo.

### 6.3.1. Sonido de motores

El caso de los efectos de sonido relacionados con los motores de los jugadores ha de ser detallado. Se trata del único sonido dinámico, que varía en función de la velocidad del corredor. Así, el ruido del motor será grave cuando el coche se encuentre en ralentí, pero su tono aumentará conforme el corredor acelere. Además, es el único audio que se ha modelado en un espacio 3D, pudiéndose distinguir la procedencia de cada fuente de sonido.

La primera característica se ha implementado con una relación lineal entre el tono del SFX y la velocidad de cada corredor. Estos parámetros se actualizan en cada uno de los bucles de refresco del estado del juego.

En cuanto al segundo aspecto, todas las fuentes de audio de este tipo son omnidireccionales y tienen asociada la posición en el espacio del vehículo que las genera. Teniendo en cuenta una atenuación prefijada de la onda de sonido en función de la distancia, se calcula la dirección y el recorrido del efecto de motor desde su origen hasta el oyente, es decir, el jugador. De este modo, el usuario escuchará de forma continuada el ruido de su propio motor y podrá distinguir, además, por dónde se acercan el resto de competidores.

La localización del oyente queda determinada por su posición en el espacio 3D ( $x, y, z$ ) y dos vectores perpendiculares para fijar la rotación.

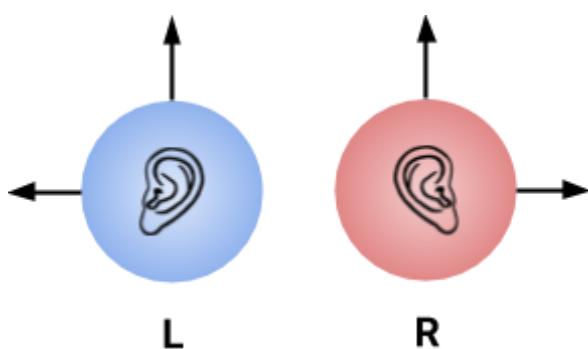


Figura 49. Esquema de oyente.

En la Figura 49, se representa un oyente cuya posición se sitúa en el centro de la esfera, y se define el vector “mirando hacia delante” y el vector “apuntando hacia arriba”. Con todo ello, quedan completamente especificados los dos canales de audio, izquierdo (L) y derecho (R).

## 7. Jugabilidad

El juego ofrece una buena jugabilidad basada en unos gráficos amigables con una interfaz intuitiva, gran capacidad de personalización de controles, una inteligencia artificial adecuada para ofrecer un reto al jugador sin que este se vea incapaz de superarla y un desafío por conseguir la mejor puntuación.

Como ya se ha expuesto en apartados anteriores, la inteligencia artificial que ofrece este videojuego se adapta a tres niveles de dificultad (fácil, normal y difícil) modificando la velocidad de los conductores, probabilidades de usar objetos especiales y capacidad de obstruir el paso de los conductores (no dejar adelantar) por parte de la IA. Esto hace que el jugador si no es capaz de superar a los contrincantes (no jugadores) en una dificultad alta pueda probar una más baja para practicar y mejorar hasta superar la dificultad más alta consiguiendo así el mayor logro del juego.

Además, este videojuego tiene una gran capacidad de rejugabilidad ya que ninguna partida es como la anterior debido a que en cada partida los parámetros de la inteligencia artificial (probabilidades de usar objetos, obstruir, temeridad) son elegidos aleatoriamente para cada conductor no humano.

### 7.1. Controles

Los controles de juego por defecto son muy cómodos para cualquier jugador ya que por un lado la mano izquierda se encarga de controlar la potencia del coche acelerando, frenando o derrapaje; mientras que la mano derecha se encarga de controlar la dirección y, en caso de tener un objeto especial, utilizarlo.

Si esta configuración no es la que más se ajusta al jugador, es libre de cambiarla por otra y dicho cambio será persistente entre ejecuciones al igual que con las demás opciones del juego (sonido, resolución, etc.)

Tecla	Acción
X	Acelerar
Z	Frenar
C	Derrapar (salto corto)
<b>Flecha Izquierda</b>	Girar a la izquierda
<b>Flecha Derecha</b>	Girar a la derecha
<b>Flecha Arriba</b>	Lanzar objeto especial hacia delante
<b>Flecha Abajo</b>	Lanzar objeto especial hacia atrás
<b>Escape</b>	Pausa / Cancelar
<b>Enter</b>	Aceptar / Pasar

Figura 50. Tabla de controles tecla acción.

### 7.1.1. Opciones y configuración

Además de los controles, como ya se ha indicado, es posible cambiar algunas configuraciones del juego en el menú "Settings" (Figura 51), estas son:

- **El audio:** tanto la música de fondo o «Music» como los sonidos o «SFX». Ambas opciones permiten establecer el volumen entre 0 y 100 puntos.
- **La resolución:** tiene 7 opciones fijas: 256x224, 512x448, 768x672 (por defecto), 1024x896 y, en las pantallas que lo permitan, 1280x1120, 1536x1344 y 2048x1792.

Todas estas configuraciones son persistentes entre diferentes ejecuciones del juego.

## 7.2. Diseño y generación de niveles

Se han mantenido los diseños originales del juego Super Mario Kart desarrollado por Nintendo para la videoconsola SNES. De cada uno de los ellos sólo se dispone de un asset 2D representando el plano del mapa visto desde el cenit. A partir de este fichero de imagen se ha desarrollado un *script* capaz de interpretar la información visual disponible y generar instrucciones textuales sobre la naturaleza del nivel.

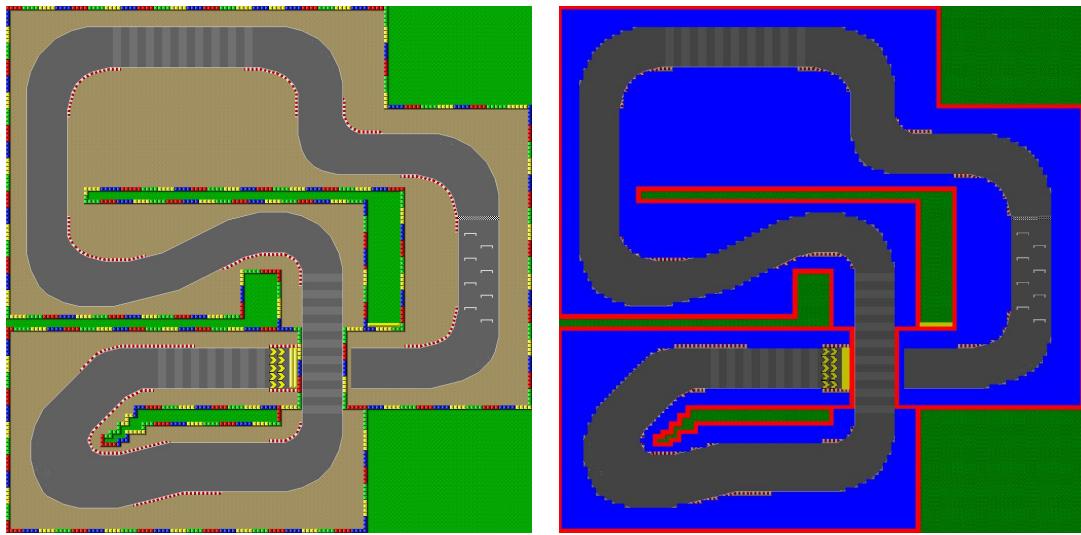
Todos los mapas se puede ver como rejillas 128x128 celdas, en las que cada compartimento engloba 8x8 píxeles de la imagen. De este modo, la generación de un nuevo nivel consiste en recorrer secuencialmente todas las celdas de la imagen. Para cada trozo procesado se aplican dos acciones:

- **Detección de terrenos:** a partir de un asset del mapa sin objetos horizontales, se calcula la mediana del color de los 64 píxeles de cada celda. El color resultante se compara con una base de datos preconfigurada, para detectar zonas de arena, hierba, vacío o lava. El resto de casillas, que no correspondan a ninguna de esas categorías, son tratadas como pista principal del circuito. La matriz resultante es volcada a un fichero que posteriormente será cargado por la aplicación del juego para poblar el [Mapa de terrenos](#).

En la Figura 52 se muestra como el *script* es capaz de detectar las zonas de arena y catalogarlas como zonas de terreno lento (■ azul). Además deja inalterada la zona de la pista (■ gris), donde los corredores podrán conducir con normalidad. Finalmente reconoce los límites del mapa (■ rojo) mediante un procedimiento simplificado, pero similar al que se explica en siguiente punto.



Figura 51. Menú de configuración («Settings»).



(a) Base del circuito

(b) Terrenos detectados

*Figura 52. Ejemplo de detección de terrenos en Mario Circuit 2.*

- **Detección de elementos horizontales:** utilizando un asset del mapa que contenga los objetos horizontales, el *script* es capaz de determinar la localización (coordenadas), la orientación ( $0^\circ, 90^\circ, 180^\circ, 270^\circ$ ) y la dimensión (ancho, largo) de los objetos. Recuérdese que la imagen del mapa se recorre secuencialmente, en este caso, de izquierda a derecha y de arriba a abajo. Por ello es posible decidir las variables de cada elemento comparando fragmentos de  $8 \times 8$  píxeles iterativamente.

En los ejemplos de la Figura 53 se muestra el proceso iterativo seguido por el algoritmo para establecer el tipo y la posición de varios objetos de distintas dimensiones. Cuando el *script* empareja uno de los fragmentos de  $8 \times 8$  píxeles comienza una búsqueda exhaustiva expandiendo el contorno de la zona procesada hacia la derecha y/o hacia abajo. La información de salida se vuelca a un fichero para ser leída por el programa principal del juego.

Tipo	Se detecta...	Se busca...	Corresponde a..	Salida
Zipper		✓ ✓ ✓		Orientación: arriba  Tamaño: 2x2
		✗ <hr/> ✓ ✓ ✓		Orientación: izquierda  Tamaño: 2x2
Moneda				Tamaño: 1x1
Rampa		✓ ✗ <hr/> ✓		Orientación: longitudinal  Tamaño: 3x1
		✗ <hr/> ✓		Orientación: longitudinal  Tamaño: 2x1
		✗ <hr/> ✓		Orientación: transversal  Tamaño: 1x4

Figura 53. Ejemplos de detección de elementos horizontales

### 7.3. Niveles de dificultad

El juego dispone de gran cantidad de mecanismos para permitir al usuario elegir el tipo de reto al que se quiere enfrentar, y posteriormente ajustar ese reto a las habilidades concretas del jugador. En concreto, dependiendo del nivel de dificultad elegido (50/100/150cc) se modifican los siguientes aspectos:

- Las propiedades de los [Vehículos](#) (aceleración, giro, etc.) son escaladas: los jugadores avanzan y manejan su coche con mayor capacidad.
- Los NPCs tienen mayor probabilidad de realizar sus [Acciones especiales](#), como bloquear otros corredores o esquivar obstáculos.
- Se incluye un cierto nivel de variación en el concepto de *farvision*, presentado en el apartado de [Inteligencia Artificial](#). Al variar este valor, los NPCs tomarán las curvas más pronto o más tarde, no de forma óptima.
- Finalmente se introduce el concepto de **dificultad adaptable**. El nivel de los NPCs cambia dinámicamente dependiendo del rendimiento del jugador. Si el jugador es más rápido que su competencia, esta empezará a conducir más rápido. En el caso contrario, su competencia se ralentizará ligeramente para permitir al jugador alcanzarla con mayor facilidad. Este concepto también cambia según la dificultad elegida, aumentando las ventajas para los NPCs y reduciendo sus ralentizaciones.

## 8. Inteligencia Artificial

Una serie de técnicas son aplicadas para dotar de cierta inteligencia a los NPCs del juego, así como a ciertos objetos animados.

### 8.1. Direcciones y aceleración

Este primer apartado hace referencia al movimiento de los *Non-Player Characters* (NPCs) del juego, corredores que compiten contra el jugador. Estos deben de ser capaces de completar las cinco vueltas. En el diseño de la *Inteligencia Artificial* o IA se han tenido en cuenta los siguientes aspectos:

- Los coches de todos los corredores, tanto el *Player Character* como NPCs, deben seguir las mismas reglas. Los NPCs deben crear órdenes de "acelerar" o "girar" de la misma forma que lo haría un jugador por teclado.
- Los NPCs deben poder determinar qué órdenes aplicar de una forma muy rápida, permitiéndoles reaccionar a cambios en el entorno a tiempo.
- Estos también deberán poder determinar el camino desde cualquier punto del circuito, de una forma óptima, teniendo en cuenta las zonas lentas, de caída, saltos, etc.

- Estos cálculos deben funcionar para cualquier circuito, con los mínimos cambios posibles por parte de los desarrolladores. De esta forma, con poco trabajo se pueden introducir, si se desea, varios circuitos más en el juego, evitando cambiar el algoritmo para ajustarlo a las características de cada uno.

A partir de estos cuatro puntos se ha desarrollado el sistema de IA, basado en una división del circuito en celdas similar a la Figura 44. A cada celda del circuito le es asignada una **puntuación** dependiendo de su distancia a la meta (siguiendo el sentido del circuito) y los tipos de terreno (ir por terreno lento es peor que ir por terreno normal, pero puede ser interesante acortar por una zona de terreno lento si se recorre un menor espacio). La puntuación es mayor cuanto más esfuerzo se requiera para llegar a la meta. En las casillas de la meta, la puntuación es cero. La Figura 54 muestra una representación de este concepto.

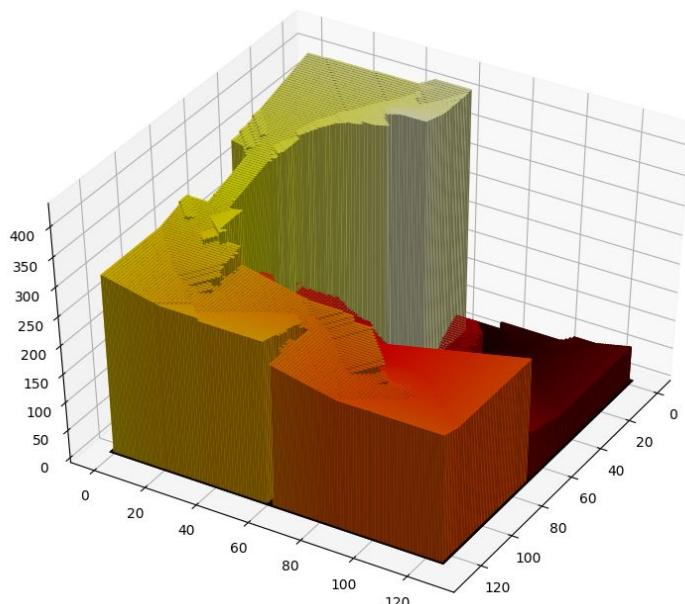


Figura 54. Representación visual de la matriz empleada para ordenar a los corredores. NOTA: esta matriz es diferente a la matriz de puntuaciones.

Las casillas con mayor valor representan puntos alejados de la meta. La idea desarrollada es que un corredor, para moverse, observa cual es la dirección de descenso en esta matriz.

Se puede observar que la meta es un caso especial, donde se ha de forzar a los corredores a seguir en la dirección de la meta (siempre hacia el norte).

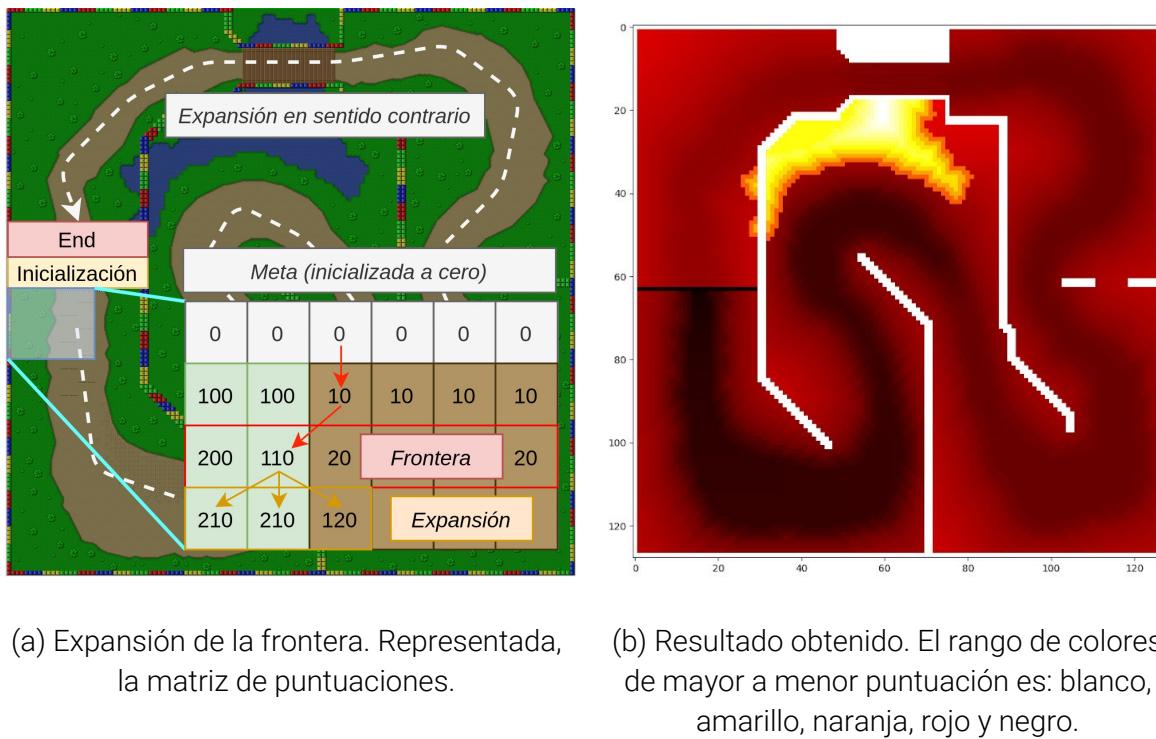
Como se verá a continuación, existen dos matrices de este tipo: la **matriz de puntuaciones** y la **matriz de posiciones**. La Figura 54 representa esta última, ya que es más fácil de visualizar.

Para entender la diferencia entre estas dos matrices hay que entender cómo se calculan dichas puntuaciones. La idea consiste en inicializar un varios elementos de la matriz con puntuación cero, correspondientes a la meta, y expandir el recorrido en dirección contraria teniendo en cuenta el tipo de terreno. Según el tipo de celda, la puntuación añadida es:

Tipo de terreno atravesado	Puntuación
Bueno: <i>zippers</i> , cajas de objetos, monedas	1 punto
Normal: asfalto	10 punto
Evitable: zonas lentas, <i>oil slicks</i>	100 puntos
Malo: agua, lava y vacío	500.000 puntos

Figura 55. Puntuación otorgada al atravesar diferentes tipos de terreno.

La Figura 56 muestra el proceso para expandir estos elementos de la matriz (la “frontera”) hasta llegar a la meta por el otro lado. Los elementos de la frontera se expanden de la siguiente forma: si el valor de una de sus celdas adyacentes es mayor que el valor de la frontera sumado a su valor de puntuación, se añade el nuevo elemento a la frontera y se sustituye su puntuación por el nuevo mínimo encontrado. Esta expansión de la frontera evita las paredes, actuando estas como bloqueo para evitar cruzar a otras zonas. Las zonas fuera de pista o lentes no siguen esta lógica, pero el algoritmo evita ir por ellas por su elevado peso.



*Figura 56. Creación de la matriz de puntuaciones para el mapa Donut Plains 1. La matriz de posiciones se encuentra en la Figura 54.*

La matriz de posiciones se emplea para el cálculo de posiciones relativas entre dos jugadores, es decir, un jugador A está delante de B si y sólo si

```

numero_vuetas[A] > numero_vuetas[B], o
numero_vuetas[A] = numero_vuetas[B] y además
    matriz_posiciones[A] < matriz_posiciones[B].
  
```

Ya que la matriz de puntuaciones otorga pesos desorbitados al entrar en terrenos malos o evitables, se ha creado otra matriz diferente para este caso. La matriz de posiciones se crea y va actualizando a la vez que la de puntuaciones, asumiendo que todas las casillas tienen peso unitario.

Para “enseñar” a un NPC a conducir, se introduce el concepto de **farvision**. La idea principal es buscar el elemento de la matriz de puntuaciones con menor valor y moverse en esa dirección, girando o acelerando si procede. Sin embargo, para tomar las curvas con suficiente

antelación se debe explorar la zona que precede la corredor. El *farvision* es el número de casillas que se avanza al buscar un punto objetivo. Un *farvision* mayor hace tomar los giros más pronto. Este valor es ajustado por cada circuito, incluyendo cierta variación según el nivel de dificultad.

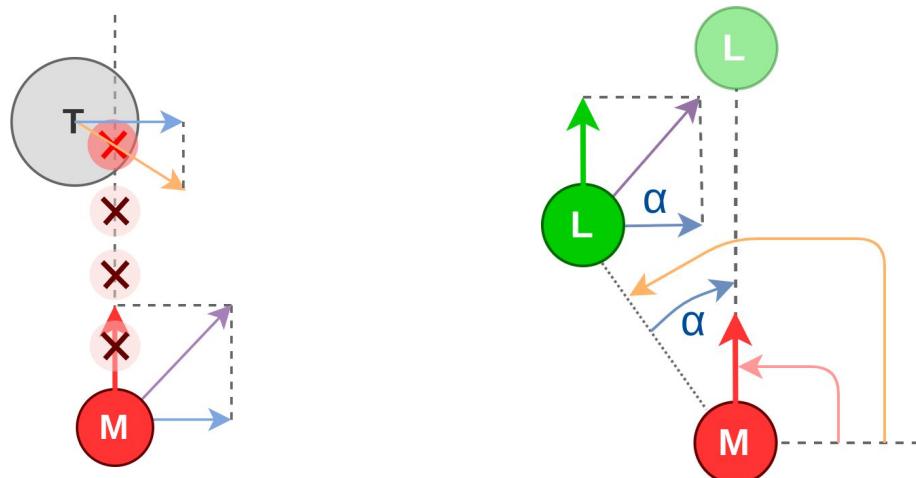
Estas dos matrices se pueden calcular para cualquier circuito, aprovechando el sistema de detección de terrenos demostrado en la Figura 52. En conclusión, con la matriz de posiciones es posible dar un orden total entre los ocho corredores, y la matriz de puntuaciones permite a los corredores moverse por el circuito sin importar su posición.

### 8.1.1. Acciones especiales

Por ahora el corredor solo sabe interactuar con el circuito, pero no con el resto de entidades que existen en él. Por ello, se han implementado las siguientes funcionalidades, las cuales son representadas en la Figura 57.

- **Esquivar otras entidades:** El corredor es capaz de evitar elementos peligrosos como plátanos, *thwomps* u otros corredores que se encuentren en la carretera. Al encontrar un obstáculo inminente se añaden órdenes de girar hacia la izquierda o la derecha, dependiendo de cuál sea la mejor opción. Esto se realiza mediante diferentes comprobaciones a distancias pequeñas frente al jugador.
- **Bloquear a otros corredores que les intentan adelantar:** Siguiendo el esquema mostrado, la IA se coloca en delante de los corredores que vienen por detrás con el objetivo de bloquear su camino.

Ambos comportamientos son configurables según la dificultad. En este último caso, la IA se desplaza hacia el jugador con mayor o menor rapidez y probabilidad.



(a) Evasión de un *thwomp* al encontrarse en frente de un corredor (Mario). Giro a partir del vector azul (componente del naranja a 90°). (b) Bloqueo de Luigi a Mario a partir de la diferencia de ángulos  $\alpha$ , que se pretende minimizar a cero.

Figura 57. Acciones especiales de la Inteligencia Artificial.

## 8.2. Relocalización de corredores

Cuando un corredor sale fuera de los límites ha de ser reposicionado en una zona segura para que pueda continuar la carrera. El jugador recibe una penalización en forma de pérdida de monedas. Además será retenido durante un tiempo sin poder moverse.

Para encontrar una posición justa a la que transportar al corredor se mantiene un historial de las últimas posiciones de cada jugador. Cuando es necesario relocalizar un vehículo se retrocede un número de posiciones hasta una zona segura. Finalmente, se utiliza la dirección decreciente del gradiente para situar al jugador centrado en la pista y mirando hacia delante.

Se deben tener en cuenta varias condiciones para ejecutar la relocalización:

1. Mantener coherente el número de vueltas completadas por el corredor. Si existe la posibilidad de que el jugador atravesase la meta como consecuencia de la relocalización, se debe impedir que esto suceda o actualizar el contador acordemente.
2. No depositar al corredor en una zona insegura. Si el jugador reaparece sobre una zona de agua, lava o vacío, volverá a ser penalizado por una acción que él no ha cometido.

## 8.3. Power-Ups

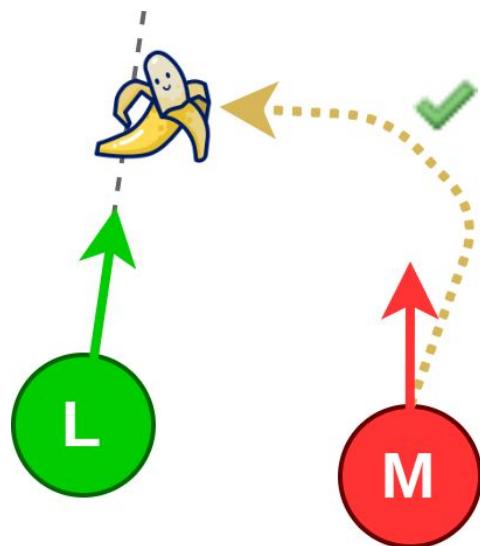


Figura 58. Lanzamiento de plátano.

Por último, se ha implementado un conjunto de reglas que indican a los NPCs cuándo es un buen momento para utilizar cada uno de los Power-Ups. La idea principal es obtener una probabilidad de uso en cada una de las actualizaciones.

Si la probabilidad es alta, lo usará en poco tiempo. Si no es un buen momento para usar el objeto se deja una probabilidad baja, de forma que de media lo siga utilizando antes de coger la siguiente caja de objetos. En orden de simplicidad, las reglas empleadas para cada objeto son las siguientes:

- **Moneda:** Alta probabilidad si el usuario tiene menos de nueve monedas (hay un límite de diez y no se aprovecharía del todo).
- **Trueno:** La probabilidad aumenta si el usuario está en las últimas posiciones, ya que este Power-Up afectará a más jugadores.

- **Champiñón y estrella:** La probabilidad aumenta si el usuario va muy por debajo de su velocidad máxima o si se encuentran en una línea recta (obtenido a partir de la matriz de puntuaciones).
- **Caparazón rojo:** Alta probabilidad si el usuario se encuentra lejos de la persona que va por delante de él (si se encuentra cerca, el usuario puede adelantarlo sin usar el Power-Up) o si el usuario se encuentra en la última vuelta.
- **Caparazón verde:** Alta probabilidad si el usuario se encuentra alineado con otro jugador (por delante o por detrás) a una distancia suficiente.
- **Plátano:** Alta probabilidad si el usuario se encuentra alineado con otro jugador (por detrás) o si al lanzar el plátano este caería en la trayectoria del oponente (Figura 58).

## 9. Tecnologías y herramientas de implementación

### 9.1. Lenguajes y bibliotecas

Se ha decidido implementar el videojuego en C++, dada la robustez y eficiencia del lenguaje. Además, el equipo de desarrollo contaba con gran experiencia previa en este ecosistema. Para ello se ha aprovechado la biblioteca SFML [3] que abstrae tanto la gestión de ventanas y gráficos, como el manejo del audio y efecto de sonido 3D.

En el caso de la [Generación de niveles](#), los scripts han sido desarrollados en Python por su simplicidad en el manejo de imágenes y la potencia expresiva de sus primitivas. Además, se hace uso de la biblioteca Matplotlib [4] y Numpy para la gestión de datos y la creación de gráficos e ilustraciones.



*Figura 59. Lenguajes y principales bibliotecas utilizadas.*

## 9.2. Control de versiones y herramientas de edición digital

Todo el código y los recursos del proyecto se encuentran bajo control de versiones mediante un repositorio de Git, gestionado por la herramienta GitHub. De esta plataforma también se ha aprovechado la herramienta de gestión de incidencias para mantener el seguimiento de las fallos encontrados y mejoras propuestas. Las versiones de los documentos escritos para el diseño y memoria del proyecto se han gestionado con las herramientas ofrecidas por Google Drive y aplicaciones de terceros compatibles.

Para la edición individual y/o colaborativa del código, se han aprovechado las utilidades complementarias a Visual Studio Code. En la edición de los recursos visuales y de audio se han utilizado: GIMP, programa de edición de imágenes libre y gratuito; y Audacity, aplicación libre para grabación y edición de audio. Por último, las reuniones telemáticas para coordinación del proyecto han tenido lugar a través de la plataforma Google Meet.

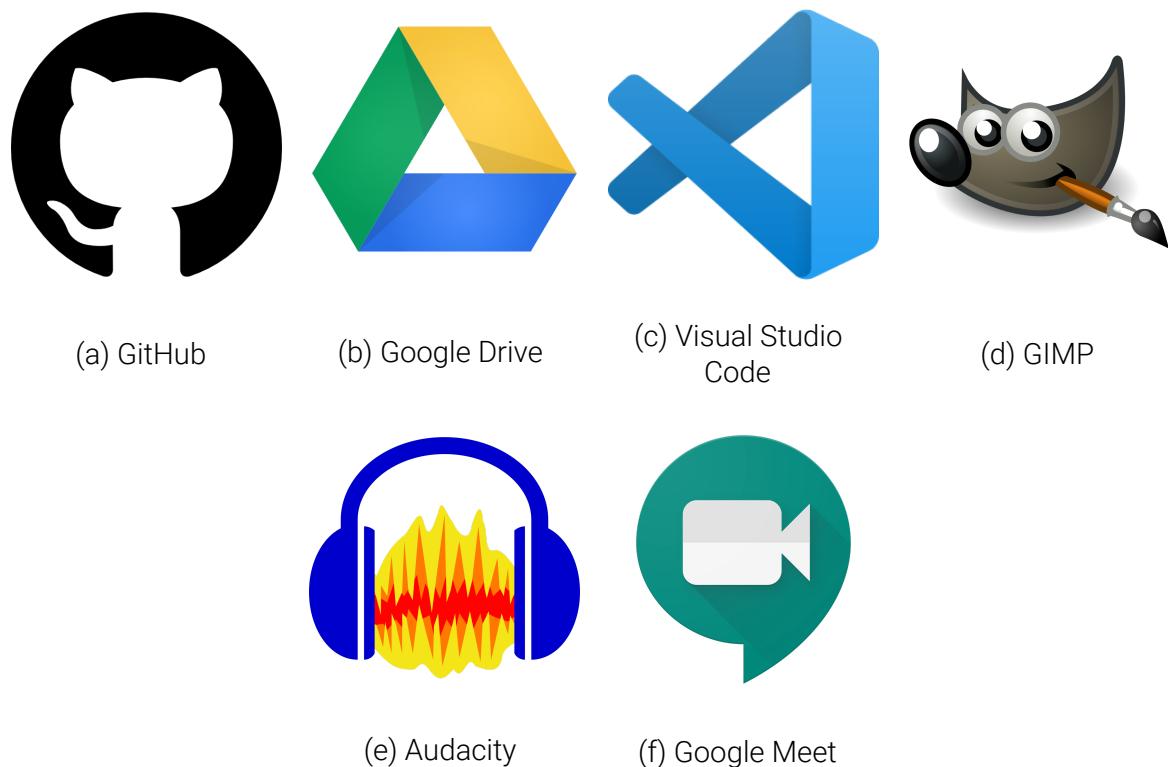


Figura 60. Principales herramientas y aplicaciones utilizadas.

## 10. Gestión del proyecto

### 10.1. Problemas encontrados

Durante el desarrollo del videojuego han surgido varios problemas a la par que sugerencias para mejorar el resultado final, estas son algunas de las más destacables que hemos resuelto/sugerido:

Título	Tipo	Estado
<a href="#">`Segmentation fault` en Bowser Castle</a>	Bug	Resuelto
<a href="#">Problemas con fixedUpdate y push/pop de states</a>	Bug	Resuelto
<a href="#">Añadir una sombra debajo de los objetos con height &gt; 0</a>	Mejora	Resuelto
<a href="#">Mejora del uso de objetos por parte de la IA</a>	Mejora	Resuelto

Figura 61. Principales problemas y mejoras que surgieron durante el desarrollo, más en el [repositorio](#) de GitHub

El listado de cambios tras la versión Beta, gracias a los comentarios recibidos es:

- Permitir saltar algunas cinemáticas.
- Agregar las opciones "cámara fija" y "seguir controlador" en el modo "demostración".
- Mejorar IA cargando información desde ficheros para optimizar recursos.
- Agregar aleatoriedad a los parámetros de IA (visión lejana, uso de elementos, impedimento, etc.) para mejorar la jugabilidad.
- Agregar un final de carrera forzada tarda más de 5 segundos en cruzar la meta tras hacerlo el último de los NPCs.
- Añadir a la IA capacidad de impedir que otro corredor le adelante, bloqueando su camino.
- Mejora visual del estado de la entrega de premios, con varias animaciones más.
- Agregar un sonido de advertencia al ser seguido por un caparazón rojo.
- Mejora los objetos de caparazón: ahora pueden volar al coger rampas.
- Mejorar la jugabilidad al equilibrar los factores de IA, las probabilidades de utilizar de objetos y al introducir dificultad adaptable.
- Solucionado el error al golpear bloques de pared con "impulso de colisión" (después de golpear a un jugador / golpe / tubería).
- Agregar animación de corredores a la pantalla de inicio.

## 10.2. Reparto de tareas y cronograma

El equipo de desarrollo está formado por tres integrantes. La repartición de responsabilidades se ha decidido de manera consensuada y equitativa.

Nombre de tarea	Encargado
1. Circuito en 2.5D	Diego
1.1. Visualización	Diego
<ul style="list-style-type: none"> <li>• Proyección circuito (Mode 7)</li> <li>• Proyección del fondo</li> <li>• Proyección de obstáculos (pipes, Thwomp...)</li> <li>• Minimap</li> </ul>	Diego
<ul style="list-style-type: none"> <li>• Proyección cajas de objetos (power-ups)</li> <li>• Proyección de monedas</li> </ul>	Victor
<ul style="list-style-type: none"> <li>• Paneles de juego y GUI (tiempo, posición, objeto...)</li> <li>• Lakitu</li> </ul>	Javier
1.2 Físicas	Víctor
<ul style="list-style-type: none"> <li>• Recorrido sin limitaciones</li> <li>• Limitaciones duras (off-limits)</li> <li>• Limitaciones blandas (slow-downs)</li> <li>• Salto de obstáculos (rampas)</li> <li>• Bandas de aceleración (speed-ups)</li> </ul>	Víctor
<ul style="list-style-type: none"> <li>• Detección de meta</li> </ul>	Javier
<ul style="list-style-type: none"> <li>• Detección de posiciones</li> </ul>	Javier
2. Personajes	Javier
2.1. Visualización	Javier
<ul style="list-style-type: none"> <li>• Superposición estática (PC)</li> <li>• Animación (PC)</li> </ul>	Javier
<ul style="list-style-type: none"> <li>• Proyección 2.5D (NPCs)</li> <li>• Animación (NPCs)</li> </ul>	Diego
2.2. Físicas	Diego
<ul style="list-style-type: none"> <li>• Colisiones PC - NPC</li> <li>• Colisiones NPC - NPC</li> <li>• Colisiones con objetos (power-ups)</li> <li>• Modelo físico parametrizado (speed, weight...)</li> </ul>	Diego
3. Inteligencia Artificial (IA)	Diego
3.1. Personajes no jugadores (NPCs)	Diego
<ul style="list-style-type: none"> <li>• Trayecto perfecto</li> <li>• Velocidad máxima</li> </ul>	Diego
<ul style="list-style-type: none"> <li>• Aleatoriedad</li> </ul>	Javier
<ul style="list-style-type: none"> <li>• Obstaculización</li> <li>• Espejo</li> </ul>	Victor
3.2. Objetos inteligentes	Diego
<ul style="list-style-type: none"> <li>• Caparazón rojo (red shell)</li> </ul>	Diego

4. Menús de juego	Diego
4.1. Menú de inicio	Diego
<ul style="list-style-type: none"> <li>● Selección de modalidad (50cc., 100cc.)</li> <li>● Selección de personaje</li> <li>● Selección de circuito</li> </ul>	Diego
4.2. Menú de configuración	Diego
<ul style="list-style-type: none"> <li>● Edición de controles</li> <li>● Edición de sonido</li> </ul>	Diego
4.3. Otros menús	Diego
<ul style="list-style-type: none"> <li>● Menú de fin</li> <li>● Contabilización de puntos</li> </ul>	Diego
<ul style="list-style-type: none"> <li>● Menú de pausa</li> </ul>	Víctor
5. Sonido	Víctor
<ul style="list-style-type: none"> <li>● Efectos especiales</li> <li>● Música de fondo</li> </ul>	Javier
6. Objetos (power-ups)	Diego
<ul style="list-style-type: none"> <li>● Cáscara de plátano</li> <li>● Caparazón verde</li> <li>● Caparazón rojo</li> </ul>	Diego
<ul style="list-style-type: none"> <li>● Champiñón</li> <li>● Estrella</li> <li>● Monedas</li> <li>● Rayo</li> </ul>	Javier
7. Diseño y generación de niveles	Víctor
<b>ENTREGABLES</b>	
Entrega Game Design Document (GDD) (2020/02/25)	Todos
Entrega videojuego (2020/05/12)	Todos
Feedback de otros grupos (2020/05/19)	Todos
Entrega final (2020/05/26)	Todos
- Memoria	Todos
- Slides	Javier
- Ejecutable	Víctor
- Video promocional	Diego

*Figura 62. Reparto de las principales tareas.*

En la Figura 62 aparecen recogidas las tareas más importantes llevadas a cabo durante el desarrollo del proyecto. La existencia de un encargado para cada parte no impide que cualquier miembro del equipo haya podido contribuir en algún detalle de su realización.

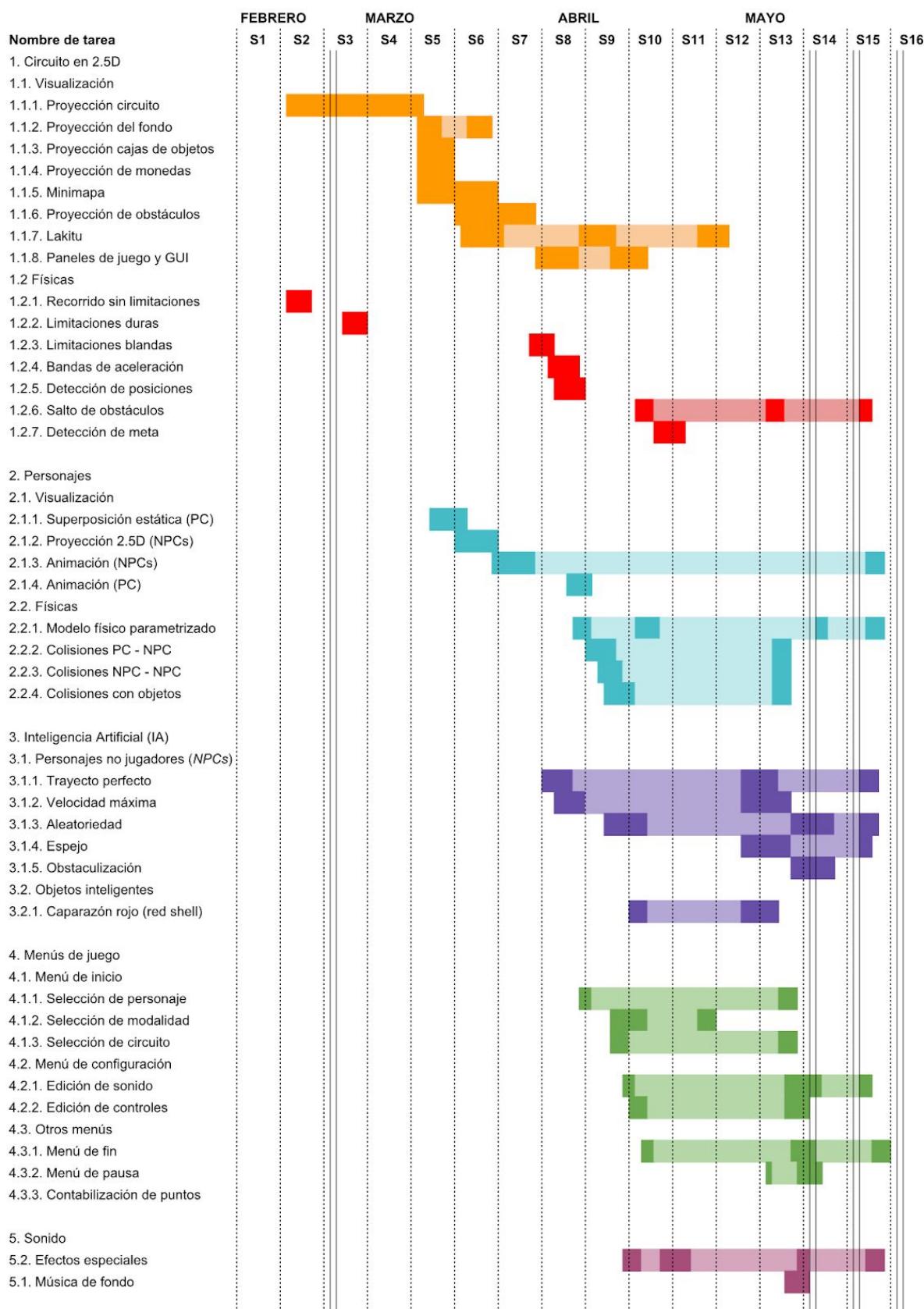


Figura 63a. Cronograma del proyecto. Parte 1.

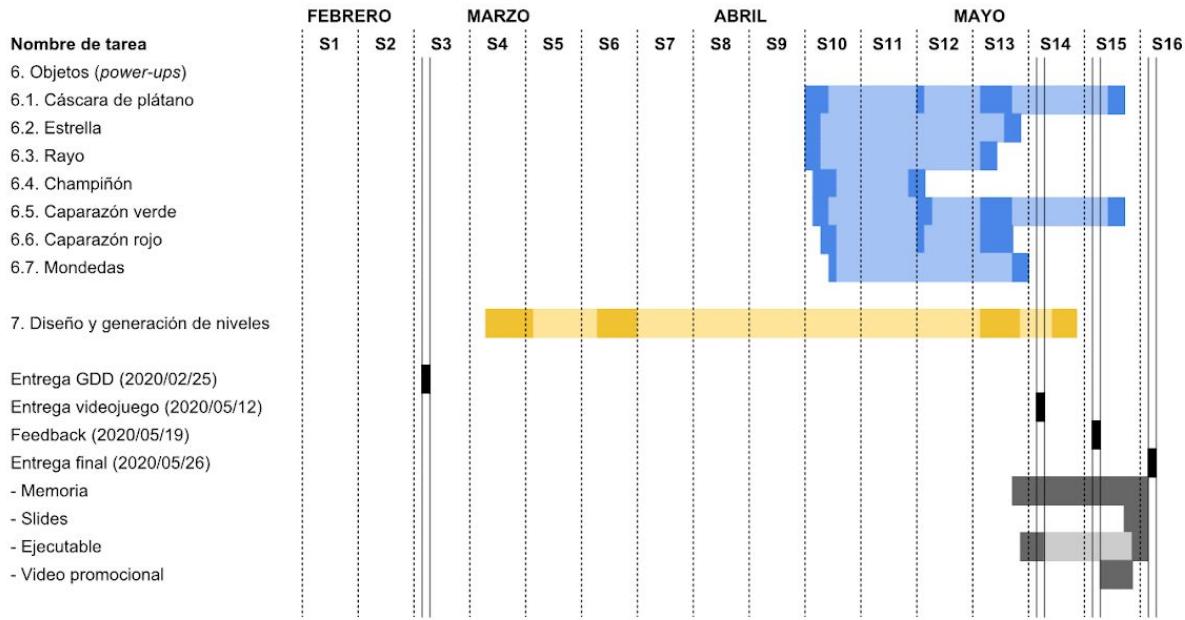


Figura 63b. Cronograma del proyecto. Parte 2.

En la Figura 63 se muestra el cronograma del proyecto, según la subdivisión de tareas resumida en la Figura 62. El eje temporal se subdivide en las 16 semanas de duración, a lo largo de las cuales se trabajó intensivamente (oscuro) o de manera esporádica (claro) en cada una de las partes que conforman el proyecto.

## 11. Referencias

- [1] Fix Your Timestep! (2004): [https://gafferongames.com/post/fix\\_your\\_timestep/](https://gafferongames.com/post/fix_your_timestep/)
- [2] Spatial hashing implementation for fast 2D collisions (2009):  
<https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/>
- [3] SFML for C++: <https://www.sfml-dev.org/>
- [4] Matplotlib for Python: <https://matplotlib.org/>