

Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

|  |
| --- |
| **Trabajo Fin de Grado** |
| **Remake del videojuego Mario Bros** |

|  |
| --- |
| **Alumno** |
| **Alejandro Ramos Gallego** |
| **Tutor** |
| **D.Carlos Javier Ogayar Anguita** |
| (Departamento de Informática) |

**[mes], [año]**



Don D.Carlos Javier Ogayar Anguita, tutor del Trabajo Fin de Grado titulado: ‘**Remake del videojuego Mario Bros**’, que presenta Don Alejandro Ramos Gallego, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, [mes] de [año]

|  |  |
| --- | --- |
| El alumno: | El tutor: |
|  |  |
| Alejandro Ramos Gallego | D.Carlos Javier Ogayar Anguita |

**Agradecimientos**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna.

Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci.

|  |  |
| --- | --- |
| Ficha del Trabajo Fin de Título | |
| Titulación | Elija un elemento. |
| Modalidad | Elija un elemento. |
| Especialidad (solo TFG) | Elija un elemento. |
| Mención (solo TFG) | Elija un elemento. |
| Idioma | Español |
| Tipo | Elija un elemento. |
| TFT en equipo | No |
| Autor/a | Alejandro Ramos Gallego |
| Fecha de asignación | Haga clic aquí o pulse para escribir una fecha. |
| Descripción corta |  |

|  |  |
| --- | --- |
| normas aplicadas en este documento | |
| Locales | |
| TFT-UJA:2017 | Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén  (Normativa marco UJA aprobada en Consejo de Gobierno) |
| TFT-EPSJ:2017 | Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén  (Normativa EPSJ aprobada en Junta de Escuela) |
| TFT-EPSJ | Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén |
| Nacionales e Internacionales | |
| ISO 2145:1978 | Documentación - Numeración de divisiones y subdivisiones en documentos escritos |
| UNE 50132:1994 | Traducción de la ISO 2145 |
| APA 6ª edición | Estilo de referencias y citas de APA (American Psychological Association) |

|  |  |
| --- | --- |
| normas utilizadas como base o referencia | |
| Nacionales | |
| UNE 157001:2014 | Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico |
| UNE 157801:2007 | Criterios generales para la elaboración de proyectos de sistemas de información |
| *Estas normas se han utilizado como base o referencia para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como proyecto la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.* | |

Contenido

[1 Introducción 13](#_Toc166511272)

[1.1 Propósito del Proyecto 13](#_Toc166511273)

[1.2 Alcance del Proyecto 14](#_Toc166511274)

[2 Herramientas y tecnologías 15](#_Toc166511275)

[2.1 Unity - Motor de Desarrollo 15](#_Toc166511276)

[2.2 Herramientas Adicionales 16](#_Toc166511277)

[3 Diseño y planificación 17](#_Toc166511278)

[3.1 Metodología de Desarrollo y Planificación temporal 17](#_Toc166511279)

[3.2 Estimaciones de recursos y costes 18](#_Toc166511280)

[4 Diseño inicial 20](#_Toc166511281)

[4.1 Visión General del Juego 20](#_Toc166511282)

[4.2 Mecánicas del Juego 21](#_Toc166511283)

[4.3 Personajes 23](#_Toc166511284)

[4.4 Interfaz de Usuario 24](#_Toc166511285)

[5 Desarrollo del juego 24](#_Toc166511286)

[5.1 Planificación y Configuración Inicial 24](#_Toc166511287)

[5.1.1 Configuración del Proyecto en Unity 24](#_Toc166511288)

[5.2 Desarrollo Incremental de Características 24](#_Toc166511289)

[5.2.1 Creación y Manipulación de Sprites 24](#_Toc166511290)

[5.2.2 Implementación de Físicas y Colisiones 24](#_Toc166511291)

[5.2.3 Animación de Personajes y Objetos 24](#_Toc166511292)

[5.2.4 Creación y Gestión de Enemigos 24](#_Toc166511293)

[5.2.5 Implementación de Transformaciones y Power-ups 24](#_Toc166511294)

[5.2.6 Gestión de Items y Poderes Especiales 24](#_Toc166511295)

[5.2.7 Integración de Audio y Efectos Sonoros. 24](#_Toc166511296)

[5.2.8 Desarrollo de la Interfaz de Usuario y HUD 24](#_Toc166511297)

[5.2.9 Implementación del Sistema de Puntuación y Temporizador. 25](#_Toc166511298)

[5.3 Enriquecimiento y Expansión de Juego 25](#_Toc166511299)

[5.3.1 Construcción de Niveles 25](#_Toc166511300)

[5.3.2 Configuración de la Cámara y Ajustes Visuales 25](#_Toc166511301)

[5.4 Evaluación y Adaptación 25](#_Toc166511302)

[6 Pruebas y validación 26](#_Toc166511303)

[6.1 Pruebas de Funcionalidad 26](#_Toc166511304)

[6.2 Pruebas de Interacción y Usabilidad 26](#_Toc166511305)

[7 finalización y transiciones 26](#_Toc166511306)

[7.1 Ajustes Finales y Preparación de Transiciones de Nivel 26](#_Toc166511307)

[7.2 Implementación del Menú Principal y Opciones de Juego. 26](#_Toc166511308)

[8 Apéndices 26](#_Toc166511309)

[8.1 Guía original del Trabajo Fin de Título 26](#_Toc166511310)

[8.2 Manuales de usuario 26](#_Toc166511311)

[9 Bibliografía 27](#_Toc166511312)

Índice de ilustraciones

[Ilustración 10.1 30](#_Toc1631674)

Índice de tablas

[Tabla 10.1 30](#_Toc1631675)

# Introducción

## Propósito del Proyecto

El propósito de este Trabajo de Fin de Grado es desarrollar un remake del icónico videojuego Super Mario Bros en 2D, utilizando el motor de juego Unity. Este proyecto tiene como objetivo principal aplicar y demostrar las competencias adquiridas en el campo de la ingeniería Informática, específicamente en el desarrollo de software, diseño gráfico y programación orientada a objetos. Además, se busca explorar las capacidades de Unity como herramienta de desarrollo de videojuegos, enfocándose en la reproducción fiel de las mecánicas, estética y niveles del juego original, al tiempo que se incorpora mejoras de diseño que enriquezcan la experiencia del usuario.

**Objetivos específicos del proyecto:**

* **Recrear la Jugabilidad Clásica:** Implementar las mecánicas fundamentales de Mario Bros, asegurando que el juego no solo se sienta como una réplica fiel del original en términos de controles y respuesta, sino que también capture la esencia y el desafío que definieron el juego.
* **Modernizar Elementos Gráficos:** Aunque el juego mantendrá un estilo visual que respete el original, se utilizarán las herramientas modernas de Unity para mejorar los gráficos, animaciones y efectos visuales, adaptándolos a las resoluciones actuales sin perder el encanto retro.
* **Documentación Completa del Proceso de Desarrollo:** Detallar cada fase del desarrollo, desde la planificación y diseño hasta la implementación y pruebas, proporcionando una base documental sólida.

## Alcance del Proyecto

El alcance de este Trabajo de Fin de Grado se centra en desarrollar un remake del videojuego Super Mario Bros, limitando el contenido al primer mundo, que comprende los cuatro primeros niveles herramientas y tecnologías.

El alcance de este proyecto incluye:

1. **Desarrollo de Niveles:**

**Recreación de los Cuatro Primeros Niveles:** El proyecto incluirá el diseño y desarrollo de los primeros cuatros niveles del mundo 1 de Super Mario Bros, cada uno con suscaracterísticas únicas y desafios, incluyendo el nivel subterráneo y el nivel del castillo final .

1. **Mecánicas de Juego**

**Implementación de Mecánicas clásicas:** Se implementarán las mecánicas básicas de movieminto de Mario, incluyendo correr, saltar y agacharse, así como las iterraciones con enemigos, objetos y el entorno de juego.

**Sistemas de Puntuación y Progreso:** Se reproducirán los sistemas de puntuación basados en la recolección de monedas, eliminación de enemigos y finalización de niveles, junto con la gestión de vidas y el progreso del jugador a través de los niveles.

**Iteracciones con Enemigos y Objetivos:** Se incluirán iterraciones clásicas con enemigos como los Goombas y Koopas, así como con objetos como bloques de interrogación, bloques de ladrillos, y power-ups(hongos y glores de fuego)

1. **Elementos Audiovisuales:**

**Gráficos y Animaciones Modernizadas:** Aunque el estilo visual mantendrá un homenaje al arte pixelado original , se utilizar las herramientas de Unity para mejorar las animaciones y los gráficos, asgeurnado que sean adecuados para dispositivos de alta resolución.

**Efectos de Sonido y Música:** Se recrearán los efectos de sonido icónicos y la música de fondo característica de cada nivel, optimizados para una reproducción de alta calidad.

1. **Documentación y Pruebas:**

**Pruebas de Juego:** Se realizarán pruebas para asegurar que el juego funciona correctmanete en términos de mecánica, rendimiento y bugs.

# Herramientas y tecnologías

## Unity - Motor de Desarrollo

Unity es un motor de desarrollo de videojuegos ampliamente conocido y utilizado en la industria por su versatilidad y robustez, lo que lo hace una elección ideal para la creación de juegos en 2D. Para este Trabajo Fin De Grado, he seleccionado Unity debido a varias de sus características clave que son particularmente adecuadas para el este proyecto:

1. **Facilidad de Uso:**

**Interfaz Intuitiva:** Unity ofrece una interfaz de usuario gráfica que es intuitiva y amigable para desarrollares de todos los niveles, lo que facilita la creación, prueba y depuración de juegos.

**Amplia Documentación:** Unity cuenta con una extensa documentación en línea, incluyendo tutoriales y foros de discusión, lo que proporciona un excelente soporte durante el aprendizaje y desarrollo.

1. **Herramientas Integradas**

**Sistema de Animación Avanzado:** Las herramientas de animación de Unity permiten crear animaciones complejas y fluidas, lo que mejora la calidad y el dinamismo visual del juego.

**Sistema de Físicas Incorporado:** Unity incluye un motor de físicas que facilita la simulación de movimientos realistas y colisiones, esencial para replicar y mejorar las mecánicas de juego de Mario Bros.

1. **Desarrollo de Scripts**

**Soporte para C#:** Unity utiliza C# para scripting, un lenguaje de programación moderno y potente, que permite escribir scripts limpios y eficientes para controlar la lógica del juego, eventos, y comportamiento de los personajes.

1. **Recursos y Assets:**

Unity proporciona acceso a una extensa biblioteca de recursos en su Asset Store, donde se pueden encontrar desde scripts hasta modelos y texturas que pueden ser utilizados para acelerar el proceso de desarrollo.

## Herramientas Adicionales

Además de Unity, para el desarrollo del videojuego he utilizado otras herramientas para diversas tareas del proceso de creación del juego. Estas herramientas adicionales incluyen Visual Studio para scripting y Paint para diseño gráfico.

1. **Visual Studio:**

**Integración con Unity:** Visual Studio se integra perfectamente con Unity, proporcionando un entorno robusto para escribir, depurar y gestionar scripts en C#.

**Herramienta de Desarrollo Avanzada:** Visual Studio es conocido por su potente editor de código, capacidad de depuración y características de autocompletado inteligente, lo que lo convierte en una herramienta indispensable para el desarrollo de scripts complejos necesarios para las mecánicas y la lógica del juego.

1. **Paint:**

**Edición gráfica sencilla:** Paint lo he utilizado en este proyecto para tareas rápidas y simples de edición gráfica, para modificar y crear sprites y texturas de manera sencilla.

**Accesibilidad y Facilidad de Uso:** Paint es una herramienta accesible que forma parte de los sistemas operativos de Windows, conocido por su interfaz intuitiva y fácil manejo, lo que lo hace ideal para los ajustes rápidos y tareas de diseño gráfico sin complicaciones.

# Diseño y planificación

## Metodología de Desarrollo y Planificación temporal

Para el desarrollo del remake de Super Mario Bros, he optado por una Metodología de Desarrollo Ágil, que se caracteriza por su flexibilidad y capacidad para adaptarse a cambios rápidos y continuos en los requisitos del proyecto. Este enfoque se centra en iteraciones cortas y frecuentes de desarrollo que permiten una integración y prueba constantes de nuevas funcionalidades y mejoras.

**Implementación de la Metodología Ágil:**

La metodología Ágil se implementa a través de un ciclo de desarrollo iterativo y evolutivo, estructurado en varias fases clave:

1. **Planificación y Configuración Inicial:**

**Configuración del entorno de Unity:** Se establecen las bases técnicas del proyecto, configurando Unity para soportar tanto los gráficos como las interacciones del juego.

1. **Desarrollo Incremental de Características:**

**Gestión de Sprites y Animaciones:** Implementación y ajuste de los sprites del personaje y de los enemigos, seguido de la animación de estos elementos para dar vida al juego.

**Integración de Mecánicas Físicas:** Incorporación de físicas y colisiones para añadir realismo y desafío a las interacciones dentro del juego.

**Implementación de Sistemas de Audio y UI:** Integración de efectos sonoros y música, junto con el desarrollo de la interfaz de usuario para mejorar la navegación y la jugabilidad.

1. **Enriquecimiento y Expansión del Juego:**

**Desarrollo de Niveles Específicos:** Construcción detallada de los primeros cuatro niveles del juego.

1. **Evaluación y Adaptación:**

Basado en la retroalimentación y los resultados de las pruebas, se realizará ajustes en el diseño y la funcionabilidad del juego para mejorar la experiencia del usuario.

## Estimaciones de recursos y costes

A continuación, se presenta una estimación detallada de los recursos y costes asociados al desarrollo del remake de Super Mario Bros utilizando Unity. Esta estimación incluye tanto los recursos humanos como materiales.

**Recursos Humanos:**

|  |  |  |
| --- | --- | --- |
| Descripción | Rol | Coste |
| Desarrollador | Estudiante de Ingeniería | 0 EUR (proyecto académico) |
| Asesor Académico | Profesor/Tutor | 0 EUR (proyecto académico) |

Tabla 10.1 Costes Recursos Humanos

**Recursos Materiales y Software:**

|  |  |  |
| --- | --- | --- |
| Recurso | Descripción | Coste |
| Unity Personal | Student Plan | 0 EUR |
| Visual Studio Community | Entorno de desarrollo integrado gratuito | 0 EUR |
| Paint | Herramienta de diseño básico | 0 EUR |
| Ordenador Personal | Necesario para la realización del proyecto | 800 EUR |
| Microsoft Office Word | Para documentación | 69 EUR (Suscripción anual) |

Tabla 10.1 Costes Recursos Materiales y Software

**Resumen de Costes Estimados:**

|  |  |  |
| --- | --- | --- |
| Categoría | | Coste Estimado Total |
| Recursos Humanos | 0 EUR | |
| Recursos Materiales y Software | 869 EUR | |
| Total Estimado | **868 EUR** | |

Tabla 10.1 Coste total del proyecto

La mayor parte del proyecto se puede completar sin costes directos significativos, debido al uso de software gratuito y recursos universitarios.

# Diseño inicial

## Visión General del Juego

**Título del Juego**

* **Super Mario Bros Remake**

**Género**

* **Plataforma 2D:** El juego mantiene el género de plataformas en dos dimensaiones, característico del original, enfocado en el control del personaje principal, Mario, que debe superar diversos obstáculos y enemigos para salvar a la Princesa Peach.

**Estilo Visual**

* **Pixel Art Mejorado**: Aunque el juego orginal es conocido por su estilo de pixel art, con este remake propongo una versión mejorada con sprites más detallados y escerios más ricos visualmente, sin perder el encanto del arte pixelado original.

**Temática**

* **Aventura y Rescate**: La temática central gira en torno a la aventura de Mario a través de diferentes mundos, cada uno con su propia temática y desafios, para rescatar a la Princesa Peach de las garras del villano Bowser.

**Audiencia Objetivo**

* **Jugadores de Todas las Edades**: Si bien el juego apela a la nostagia de los jugadores que crecieron con el original en los años 80 y 90, también esta diseñado para ser accesible y disfrutable para los jóvenes jugadores. La sencillez de las mecánicas y la universalidad de su historia hacen de este remake un juego adecuado para toda la familia.

**Objetivo del Juego**

* **Progresión Lineal a través de Niveles Desafiantes**: Los jugadores deben guiar a Mario a través de varios nieveles, cada uno ofreciendo un incremento en dificultad, con el objetivo final de derrotar a Browser y rescatar a la Princesa. Cada nivel está diseñado para poner a prueba las habilidades de salto, timing y estrategia del jugador, con varios secretos y atajos por descubrir.

## Mecánicas del Juego

En este remake, he conservado muchas de las mecánicas clásicas mientras he introducido algunas innovaciones para enriquecer la experiencia de juego.

**Controles Básicos**

* **Movimiento:** Mario puede moverse hacia la inzquierda o derecha usando los controles direccionales.

**\*\*\*\*\***

* **Salto:** El salto es la mecánica central en este viodejuego. Mario puede realizar saltos de diferentes alturas dependiendo de cuánto tiempo el jugador presione el botón de salto. Esto permite una variedad de maniobras y estrategias al enfrentar obstáculos o enemigos.
* **Correr:** Manteniendo presionado, Mario puede correr, lo que aumenta su velocidadd y la distancia de sus saltos. Correr también es crucial para superar ciertos obstáculos y enemigos más rápidamente.

**Interraciones con el Entorno**

* **Bloques Interactivos:** Algunos bloques pueden ser golpeados desde abajo para revelar monedas o power-ups.
* **Tuberías:** Algunas tuberías sirven como atajos o llevan a zonas secretas. Mario puede agacharse para entran en ellas cuando es posible.

**Power-ups y Objetos**

* **Super Champión:** Transforma a Mario en Super Mario, haciéndolo más grande y permitiéndole romper bloques de ladrillo.
* **Flor de Fuego:** Otorga a Mario la habilidad de lanzar bolas de fuego, lo que le permite derrotar enemigos desde una distancia segura.
* **Estrella de Invencibilidad:** Confiere a Mario invulnerabilidad temporal y la capacidad de derrotar enemigos con solo tocarlos.
* **Monedas:** Recolectar 100 monedas otorga una vida extra a Mario, incentivando la exploración del entorno.

**Mécanicas de Enemigos**

* **Patrones de Movimiento:** Cada tipo de enemigo tiene patrones de movieminto distintos. Por ejemplo, los Goombas caminan hacia adelante hasta que chocan con un obstáculo y se voltean, mientras que los Koopas se esconden en sus caparazones cuando son golpeados.
* **Interraciones:** Los enemigos pueden ser derrotados saltando sobre ellos o usando power-ups como las bolas de fuego.

**\*\*\*\*\***

**Innovaciones**

* **Modos de juego Adicionales:** Se introducen modos de juego que ofrecen desafíos temporales o misiones específicas, como completar un nivel sin recolectar monedas.
* **Mecánicas de Puntuación Mejoradas:** Aparte de ganar puntos por recolectar monedas o derrotar enemigos, se añaden bonificaciones por completar niveles rápidamente o con ciertas condiciones.

## Personajes

Cada personaje, tanto protagonistas como antagonistas, ha sido cuidadosamente diseñado para mantener la esencia del original mientras le he añadido detalle visual. Aquí se detalla la información sobre los persones principales y los enemigos que los jugadores encontrarán.

**Personaje Principal:**

* **Mario:** Mario es el protagonista del juego. Su misión es rescatar a la Princesa Peach, superando una serie de obstáculos y enemigos a lo largo de varios niveles. Mario puede transformarse y obtener nuevas habilidades mediante el uso de power-ups.

\*\*\*\*\*

**Personajes Secundarios:**

* **Princesa Peach:** La princesa del Reino Champiñon que ha sido secuestrada por Bowser. Ella aparece principalmente en las cinemáticas y al final de la saga del castillo.
* **Luigi:** Aunque no jugable en la versión original del primer Super Mario Bros, en este remake, Luigi aparece en ciertos modos de juego cooperativos, actuando como el segundo jugador.

**Antagonistas**

* **Bowser:** El principal antagonista del juego, Bowser es el rey de los Koopas y ha secuestrado a la Princesa Peach. Es el jefe final que Mario debe enfrentar en el último nivel de cada mundo.
* **Goombas:** Pequeñas criaturas con forma de seta que son los enemigos más comunes en los niveles. Se mueven lentamente hacia Mario y pueden ser derrotados fácilmente saltando sobre ellos.
* **Koopa Troopas:** Tortugas que caminan de un lado a otro. Al ser golpeados, se retraen dentro de sus caparazones, que luego pueden ser utilizados por Mario como proyectiles contra otros enemigos.
* **Planta Piraña:** Plantas carnívoras que salen de las tuberías intentando morder a Mario. No pueden ser derrotadas fácilmente y requieren que Mario las esquive hábilmente.

## Interfaz de Usuario

# Desarrollo del juego

## Configuración Inicial

### Configuración del Proyecto en Unity

Para el desarrollo del remake, he seleccionado la versión 2021.3.38f1, conocida por su estabilidad y soporte extenso.



Ilustración 10.1

**Selección del Template:**

He optado por el template 2D de Unity, que proporciona una configuración óptima del entorno para desarrollar juegos en dos dimensiones. Este template preconfigura el editor con las herramientas y paneles adecuados para el diseño en 2D, asegurando que los recursos como sprites y animaciones sean fácilmente gestionables y optimizados para este tipo de proyectos.

**Estructura del proyecto:**

El proyecto está organizado en una estructura de carpteas estándar que Unity recomienda y utiliza por defecto, lo que facilita la gestión y el acceso a los diversos recursos y configuraciones del proyecto.

* **Assets:** en esta carpeta se guardan todos los recursos del juego, scripts, texturas y archivos de audio.
* **Packages:** esta carptea contiene la lista de paquetes agregados al proyecto.
* **ProjectSettings:** esta carpeta contiene los archivos con las configuraciones de distintas partes del proyecto como la física, capas, etiquetas y configuraciones de tiempo.



Ilustración 10.1

## Desarrollo Incremental de Características

### Creación y Manipulación de Sprites

Para cargar y manipular un Sprite en Unity hay que seguir una serie de pasos específicos que permiten utilizar estos componentes gráficos de manera efectiva en el proyecto. Antes de nada, voy a definir los conceptos básicos:

**Sprite:** Es una imagen en dos dimensiones que se utiliza para representar objetos gráficos como personajes, enemigos, y elementos del entorno en un videojuego. Los Sprites son importados en formatos PNG o JPG, y se transforman en componentes manejables dentro del motor a través de la configuración de su Sprite Renderer.

**Sprite Renderer:** Componente de Unity que se añade a un GameObject para poder mostrar el Sprite en la escena. Controla cómo se visualiza el Sprite, incluyendo la transparencia, el color y la capa.

**Pasos para Cargar y Manipular Sprites**

1. **Importar la Imagen**

**Paso 1:** Arrastrar y soltar la imagen en la carpeta “Sprite” dentro de “Assets” del proyecto en Unity.

**Paso 2:** En el panel “Inspector”, cambiar el “Texture Type” a “Sprite (2D and UI)”.



Ilustración 10.1

1. **Configurar el Sprite**

**Paso 3:** Ajustar las propiedades del Sprite en el “Inspector”

Las opciones incluyen:

* **Sprite Mode:** Define si la imagen se maneja como un “Simple” Sprite Individual o “Multiple”, que es útil para spitesheets donde múltiples sprites están contenidos en una sola imagen. En el modo “Multiple”, se necesitá usar el Sprite Editor para definir los límites de cada Sprite individual dentro de la hoja.



Ilustración 10.1

* **Pixel per Unit:** Esta propiedad establece cuántos píxeles de la imagen corresponden a una unidad en Unity.
* **Pivot**: El pivot es el punto alrededor del cual se realizarán todas las rotaciones y escalas del Sprite.
* **Filter Mode:** Esta propiedad define cómo Unity interpola los píxeles del Sprite cuando es escalado o transformado:
  + Point: No hay interpolación; los píxeles permanecen bloqueados y nítidos, ideal para un estilo visual pixelado.
  + Bilinear: Los pixeles se suavizan, lo que hace que los Sprites se cean menos pixelados y más suaves al ser escalados.

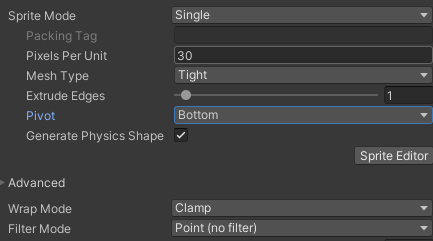


Ilustración 10.1

1. **Añadir Sprite a la Escena**

**Paso 4**: Se crea un nuevo GameObject en la escena y se añade el componente “Sprite Renderer”

En “Sprite Renderer” existe la gran opción de “Sorting Layers”, que permite controlar el orden de los objetos en la escena, es decir esta propiedad sirve para que algunos objetos se dibujen sobre otros.

**Paso 5:** Se asigna el Sprite al componente “Sprite Renderer” arrastrando el Sprite desde el panel “Project” al campo “Sprite” en el Inspector del GameObject.

### Implementación de Físicas y Colisiones

Para implementar las Físicas y las Colisiones a los objetos he seguido los siguientes pasos, lo que garantiza que los objetos interactúen de manera realista con su entorno y otros objetos del juego.

1. **Añadir un Rigidbody al GameObject:**

**Selección del GameObject:** primero se escoge el objeto dentro de la jerarquía del proyecto que se desea aplicar las físicas.

**Agregar el componente Rigidbody:** esto permite que el objeto pueda responder a la gravedad y a otras fuerzas, como impulsos y colisiones.



Ilustración 10.1

1. **Configuración del Rigidbody:**

**Tipo de Rigidbody:** Se elige entre “**Dynamic**”, para objetos que reaccionarán a fuerzas y colisiones; “**Kinematic**”, que no reaccionan a la gravedad, pero pueden ser movidos mediante scripts; o “**Static**”, para objetos que no se mueven, pero pueden ser colisionados.



Ilustración 10.1

1. **Añadir Colliders:**

**Selección del tipo de Collider:** Se tiene que escoger el Collider que se ajuste a la forma del objeto. Esto puede ser un ‘**BoxCollider’, ‘SphereCollider’,** o **‘MeshCollider’.**

**Ajuste del Collider:** Además se debe modificar las dimensiones y la posición del Collider para asegurar de que cubre el objeto de manera precisa, proporcionando una representación exacta del espacio que ocupa el objeto para la detección de colisiones.

1. **Configuración de las Colisiones:**

**Uso de Capas (Layers):** Hay que organizar los objetos en capas para controlar con qué otros objetos pueden interactuar.



Ilustración 10.1

Aquí se puede ver como se configura las capas, y en este caso, se muestra que solo va a existir colisión entre la capa “Player” y la capa “Ground”.

1. **Implementación de Materiales Físicos:**

Además, se puede crear y ajustar propiedades como la fricción y el rebote para influir en cómo los objetos rebotan o resbalan al colisionar. Esto se aplica al Collider para cambiar la manera en que el objeto interactúa al entrar en contacto con otros objetos.



Ilustración 10.1

1. **Movimiento y Salto**

**Control de Movimiento:** He implementado un sistema que permite a Mario moverse hacia izquierda o derecha y ajustar su velocidad basándose en la aceleración y la fricción, simulando un movimiento realista dentro del entorno del juego.



Ilustración 10.1

**Mecánica de Salto:** Mario puede iniciar un salto solo si está en contacto con el suelo, verificado por el script de colisiones. Durante el salto, la gravedad de Mario se ajusta para ofrecer una respuesta dinámica basada en la duración que el jugador mantiene presionado el botón de salto.



Ilustración 10.1

1. **Colisiones**

Para la detección del suelo, utilizo un script de colisiones que verifica constantemente si Mario está en contacto con el suelo mediante técnicas de raycasting.



Ilustración 10.1

**Uso del ‘Time.deltaTime’**: es un valor crucial que representa el tiempo en segundo que ha transcurrido desde que se dibujó el último frame. Este valor es esencial para garantizar que la mecánica de juego se ejecuta de manera suave y consistente en diferentes dispositivos. Es tan importante porque sin el uso de este, un juego correría más rápido en sistemas más poderosos y más lento en sistemas con menos recursos. Al multiplicar las velocidades y fuerzas aplicadas por ‘Time.deltaTime’, se normaliza el movimiento a una tasa consistente, independiente del hardware.

### Animación de Personajes y Objetos

En este apartado, describo el proceso de animación, el cual permite a los objetos y personajes del juego mostrar un comportamiento dinámico y atractivo visualmente. Los pasos para implementar las animaciones son los siguientes:

1. **Concepto de Animaciones:** Las animaciones son secuencias de imágenes estáticas (sprites) que, al ser reproducidas a una velocidad adecuada, crean la ilusión de movimiento.
2. **Uso de Unity Animator:** Unity emplea un sistema llamado “Animator Controller”, que funciona como una máquina de estados finitos. Cada estado representa una animación diferente, y el sistema puede cambiar de un estado a otro bajo ciertas condiciones, permitiendo una transición entre animaciones.
3. **Creación de Clips de Animaciones:** Se inicia seleccionando los sprites que formarán parte del clip de animación y arrastrándolos directamente a la escena de Unity. Esto automáticamente crea un GameObject con un componente Sprite Renderer y, si se trata de múltiples sprites, añade un Animator y un clip de animación.



Ilustración 10.1

1. **Configuración del Animator Controller:** Dentro del “Animator Controller”, se configuran los diferentes estados de animación y las transiciones entre ellos.



Ilustración 10.1

1. **Animaciones en Bucle y No Bucle:** En esencial determinar si una animación debe reproducirse en bucle (como correr) o si debe ejecutarse una sola vez (como un salto).



Ilustración 10.1

Para implementar correctamente las animaciones y garantizar que se sincronizan con las acciones y movimientos del personaje en el juego, se realizan ajustes que aseguran que las animaciones se activen en los momentos precisos, de acuerdo con la lógica del Juego:

* **Integración con el Sistema de Animación:** Se añade una referencia al componente “Animator”. Esto permite activar o cambiar animaciones basadas en el estado y acciones del personaje, como saltar, correr o detenerse.
* **Control Dinámico de Animaciones:** Se utiliza los métodos “animator.SetBool” y animator.SetFloat” para ajustar las animaciones basadas en el estado de movimiento del personaje, por ejemplo, ajustar la animación de salto cuando el personaje está en el aire y cambiar a la animcación de correr o detenerse según la velocidad.
* **Sincronización con la Física:** Las animaciones se sincronizan con los estados físicos del personaje, verificando condiciones como si está en el suelo o en el aire a través del script de colisiones.

Además, se ha añadido la capacidad de agacharse a Mario, limitada a sus formas Super Mario y Mario Fuego. La implementación involucra:

* **Animaciones específicas:** Se han creado animaciones dedicadas para Mario agachado tanto en su forma Super como en la forma Fuego.
* **Restricciones de movieminto:** Mientras está agachado, Mario no puede moverse.
* **Bloqueo de acciones:** En forma de Fuego, no puede lanzar bolas de fuego mientras está agachado.
* **NUEVA FUNCIONALIDAD – Romper bloques con un Pisitón** Al estar agachado, Mario puede romper bloques presionando la tecla ‘z’.

### Creación y Gestión de Enemigos

En esta sección, describo la implementación de enemigos en el juego,

enfocándome en cómo interactúan con el jugador y el entorno. La creación de enemigos sigue un proceso detallado que asegura su comportamiento dinámico y realista dentro del juego.

1. **Preparación de Sprites:**

**Organización:** Los sprites de los enemigos lo he organizado en carpteas específicas dentro del proyecto para fácil acceso y modificación.

**Configuración:** Ajusto las propiedades de tamaño y el punto de pivote de los sprites para asegurar animaciones precisas.

1. **Animación:** Utilizo el “Animator Controller” para crear las animaciones específicas de cada enemigo.

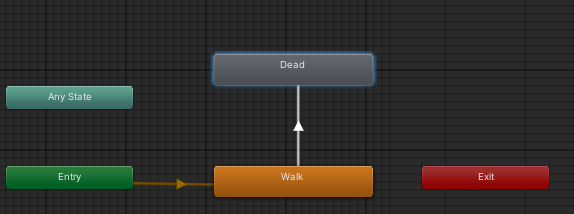


Ilustración 10.1

1. **Componentes Físicos:** Añado un “Rigidbody2D” para permitir que el enemigo interactúe con la física del juego y un “Collider” para definir el área de interacción física.
2. **Script de Movimiento:** Utilizo una estructura de clases con una case base ‘Enemigo’ que define propiedades y métodos comunes. De esta clase deruvan otras como ‘Goomba’, ‘Koopa’ y ‘Planta Piraña’, que personalizan el movimiento y otros comportamientos.

**Clase Base: Enemigo:** Incluye componentes básicos como ‘Animator’, ‘Automovimiento’, y ‘Rigidbody2D’. Este método facilita la reutilización y expansión de funcionalidades.

**Movimiento Autónomo:** ‘AutoMovimiento’ gestiona el movimiento automático y la dirección de los enemigos, adaptándose a obstáculos mediante cambio de dirección y detención.

**Clases Derivadas:** Cada clase derivada ajusta y expande el comportamiento de la clase base.

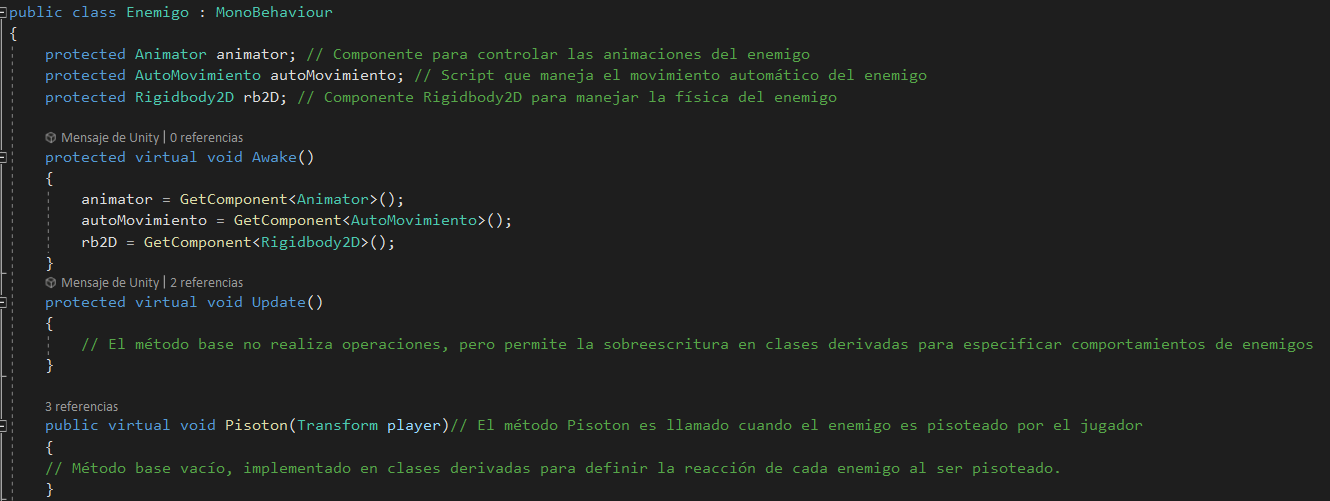


Ilustración 10.1

1. **Gestión de Colisiones:** Configuro las colisiones para que el enemigo pueda interactuar de manera adecuada con elementos del juego.
2. **Interacciones con el Jugador:** Implemento lógicas específicas para que cada tipo de enemigo responda de manera única cuando interactúa con el jugador.

Para cada tipo de enemigo, se ha desarrollado animaciones específicas que reflejan sus comportamientos únicos:

1. **Goomba**

**Andar:** Animación básica de movimiento.

**Muerte:** Al ser pisoteado por el jugador, se aplasta y desaparece.

1. **Koopa**

**Andar:** Movimiento habitual caminando

**Esconderse en el caparazón:** Cuando el jugador salta encima, se esconde en su caparazón.

**Movimiento en el caparazón:** Puede ser empujado para atacar a otros enemigos o dañar a Mario.

**Salida del caparazón:** Después de un tiempo o cuando el jugador salta de nuevo sobre él mientras está en su caparazón.

1. **Planta Piraña**

**Emerger y retirarse:** La planta sale de la tubería y se retira alternativamente.

**Muerte por bola de fuego:** Cuando es atacada por una bola de fuego, desaparece sin animación específica.

### Implementación de Transformaciones y Power-ups

Para la implementación de Transformaciones y Power-ups, he utilizado el componente ‘Animator’, que permite gestionar las animaciones mediante un sistema de estados y transiciones. Los estados de Mario los he configurado con animaciones específicas que se activan bajo ciertas circunstancias, como obtener un objeto o interactuar con enemigos.

1. **Animaciones y Estados en Animator:**

Se duplican las animaciones existentes de Mario pequeño para los estados de Super Mario y Mario Fuego, ajustando los sprites correspondientes.

Se configura el componente ‘Animator’ para manejar estos estados y transiciones.

1. **Substates para Organizar Animaciones**

Utilizo substates dentro de ‘Animator’ para agrupar animaciones similares, facilitando la gestión y la visualización.

1. **Gestión de Transiciones y Condiciones**

Ajusto las transiciones entre los estados de animación en función de las acciones del jugador, como saltar o recibir un power-ups.

Se establece condiciones en el ‘Animator’ para controlar cuándo debe pasar de un estado a otro.



Ilustración 10.1

1. **Implementación de Power-Ups**

Se añaden eventos a las animaciones para manejar los cambios de estado cuando Mario obtiene power-ups, es decir, añadiendo triggers y variables en el ‘Animator’ para activar las transformaciones adecuadas.

**Optimización y Efectos Adiciones**

Además, he añadido efectos visuales como la transparencia durante las transformaciones para mejorar la experiencia visual, y además se maneja la escala de tiempo del juego para pausar otros elementos mientras Mario se transforma, asegurando que las animaciones de transformación se ejecuten sin interrupciones.

### Gestión de Items, Poderes Especiales y Bloques

Para poder gestionar los Items como power-ups y monedas, he utilizado la funcionalidad de prefabs, que es un asset que encapsula un GameObject con sus componentes y propiedades, permitiendo su uso repetido, lo que permite una implementación eficiente y uniforme a lo largo de los diferentes niveles del juego. Al realizar cambios en un prefab, estos se aplican a todas las instancias de ese prefab en el juego.

Para enemigos, ítems y bloques que requieren variaciones, como diferencias en comportamiento y apariencia, utilizo la funcionalidad de ‘Variant.prefab’. Esto permite que las variantes hereden propiedades del prefab base, pero también mantengan sus características únicas.

Al cambiar una propiedad en el prefab base, como la velocidad de un enemigo, todas las variantes que no hayan modificado esa propiedad específica se actualizarán automáticamente.

\*\*\*\*Actualizar\*\*\*

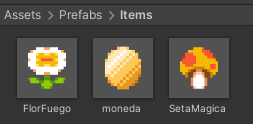


Ilustración 10.1

Para la animación de los ítems y bloques, he optado por utilizar scripts directos en lugar de los sistemas ‘Animator’ y ‘Animation. Este enfoque simplifica el proceso al evitar la sobrecarga que implica manejar numerosos controladores de animación, especialmente útil para los ítems que solo requieren animaciones básicas de cambio de Sprite.

En la implementación, cada ítem tiene asignado el script que gestiona un array de sprites. Mediante una corutina, el script cicla continuamente a través de este array, actualizando el Sprite visible en intervalos definidos.

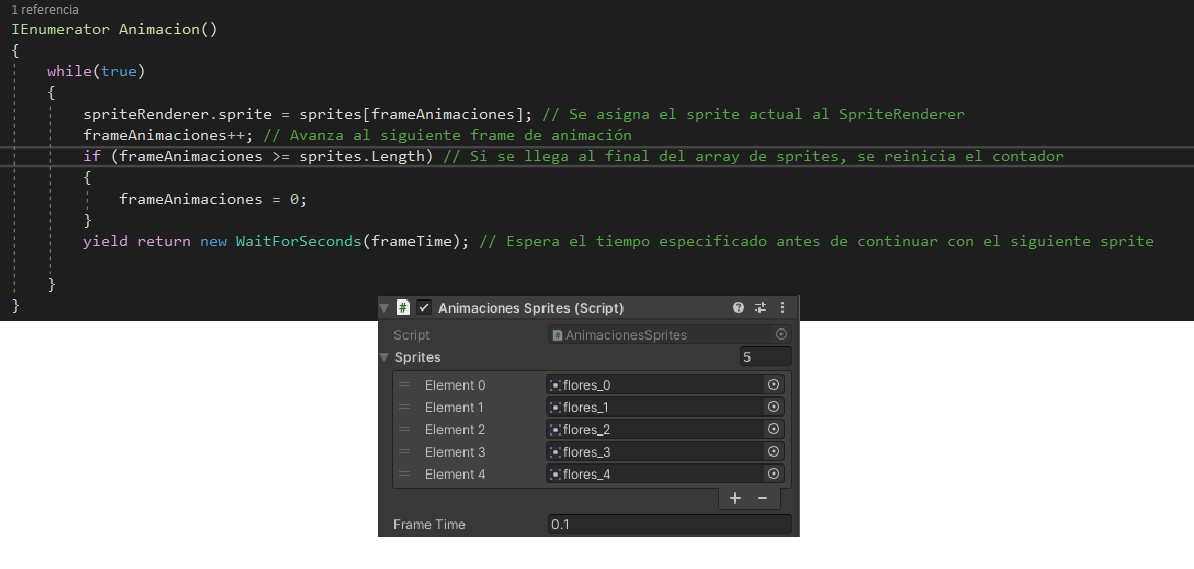


Ilustración 10.1

Además, he implementado un sistema de gestión de bloques similar al de ítems, que permite interacciones dinámicas con Mario. Este sistema se basa en varios scripts que permiten a los bloques responder a diferentes acciones de Mario.

1. **Bloque:** Controla las interacciones con los bloques, detectando golpes de Mario desde abajo. Dependiendo de su configuración, puede romperse emitir monedas o ítems, además de iniciar una animación de rebote y cambiar su Sprite a una versión vacía una vez agorados los recursos.
2. **MonedaBloque:** Gestiona las monedas que aparecen al golpear ciertos bloques, animando su trayectoria hacia arriba y su desaparición.
3. **AnimacionesSprite:** Utilizado en bloques que cambian visualmente, como pasaba en los ítems, este script anima secuencias de Sprite para visualizar cambios sin la necesidad de sistemas de animación más complejos.
4. **AutoMovmiento:** Estos se aplican a los ítems, que se mueven independientemente tras ser libreados de un bloque.

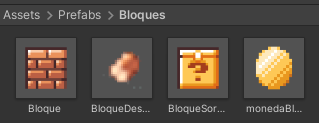


Ilustración 10.1

Para el **lanzamiento de Fuego,** he configurado un sistema que permite a Mario disparar Bolas de Fuego cuando se encuentra en el estado “Fuego”. Este proceso se inicia con la creación de un prefab de bola de fuego, que lo he configurado con propiedades específicas como la dirección, velocidad y la fuerza de rebote cuando la bola colisione con superficies horizontales.

El disparo se realiza cuando se presiona la tecla ‘Z’, que genera una bola de fuego que se desplaza en la dirección que Mario enfrenta. Esta dirección es determinada por la orientación actual de Mario.

Cuando la bola de fuego impacto con un enemigo, se activa una animación de rotación del enemigo, controlada por el ‘Animator’. Este impacto también desactiva el movimiento automático del enemigo y, después de ejecutar la animación, el enemigo es destruido.

Si la bola impacta en un objeto sólido, se genera una animación de explosión utilizando un prefab de explosión creado, que simula el efecto de l abola de fuego desintegrándose.

El comportamiento de los enemigos frente a la bola defuego está configurado para reaccionar de manera específica según el tipo de enemigo. La planta, simplemente se destruye, mientras que el Goomba y Koopa tienen una animación especial de rotación al ser golpeados.



Ilustración 10.1

Para la implementación del **efecto de invencibilidad de Mario**, he desarrollado un sistema que activa un estado especial cuando Mario obtiene la Estrella de Invencibilidad. Este estado permite a Mario ser invulnerable a los ataques y daños durante un período de tiempo determinado. El proceso ha incluido varios pasos clave:

1. **Creación del Prefab de la Estrella:** se ha desarrollado un prefab para la Estrella que incluye animaciones específicas para representar su estado activo. Esto implica una secuencia de sprites que reflejan el brillo y movimiento característico de la Estrella.
2. **Gestión de la Interacción:** Al recoger la Estrella, Mario activa un modo especial donde se cambias sus propiedades físicas y de interacción. Esto se ha hecho mediante scripts que modifican los componentes de movimiento y colisión, permitiendo que Mario no reciba daño.
3. **Animación de Invencibilidad:** Se implementó una capa adicional en el sistema de animación de Mario para cambiar los colores del Sprite durante el efecto de invencibilidad, creado una apariencia visual que refleja el estado temporal de invulnerabilidad.
4. **Control de Tiempo:** El efecto de la Estrella tiene una duración limitada. Se establece un temporizador que, una vez finalizado, revierte a Mario a su estado normal.
5. **Interacciones con Enemigos:** Durante el estado de invencibilidad, cualquier contacto de Mario con los enemigos resulta en la eliminación de estos, igual a cómo afectan las bolas de fuego

Además, he resuelto algunos puntos pendientes sobre el estado ‘Herido’ de Mario, otras muertes de los enemigos y varias interacciones y estados específicos del juego que voy a detallar a continuación:

1. **Detección de Enemigos Encima de Bloques:**

Para matar a los enemigos situados encima de un bloque cuando Mario golpea ese bloque desde abajo, se ha utilizado un área de colisión configurada justo encima del bloque. Esta caja se genera usando la función ‘Physics2D.OverlapBoxAll’, que detecta todos los colisionadores dentro de una caja definida en la ubicación específica. Si un enemigo está dentro de esta caja en el momento del golpe, se le aplica la acción de ser eliminado.



Ilustración 10.1

1. **Capacidad de Romper Bloques:**

Solo ‘Super Mario’ puede romper bloques. Esto se maneja verificando el estado actual de Mario antes de permitir que el bloque se rompa. Si Mario no esta en su forma potenciada, el bloque no se romperá y solo se mostrará una animación de rebote.

1. **Cambiar la Dirección de los Ítem Encima de los Bloques:**

Cuando Mario da un ‘cabezado’ a un bloque y hay ítems encima de este, la dirección de estos ítems cambia. Esto se implementa detectando ítems sobre el bloque (usando la misma técnica a la detección de enemigos) y aplicando un cambio en su dirección de movimiento.

1. **Muerte por ‘caparazon’ de Koopa a Otros Enemigos:**

Cuando un ‘caparazón’ de Koopa es lanzado por Mario y colisiona con otros enemigos, estos son eliminados. Esto se gestiona detectando colisiones entre el caparazón en movimiento y cualquier enemigo, y luego aplicando la lógica de ‘muerte’ a esos enemigos. Si colisiona con otro caparazón estos cambiarán de dirección.

1. **Interacción con la Planta Piraña:**

A diferencia de otros enemigos, Mario no puede ‘pisotear’ a la Planta Piraña para matarla. Si intenta hacerlo, Mario es el que recibe daño. Esto se controla verificando si el colisionador de Mario entra en contacto con una Planta Piraña y, de ser así, aplicando daño a Mario en lugar de al enemigo.

1. **Invencibilidad Temporal de Mario tras ser Herido:**

Cuando ‘Super Mario’ o ‘Mario Fuego’ recibe un golpe, entra en un estado temporal de invencibilidad, donde parpadea y no puede ser dañado. Durante este tiempo, cualquier colisión con enemigos no resultará en más daño. Este estado se maneja con un temporizador que cuenta la duración de la invencibilidad y restablece la capacidad de Mario para recibir daño una vez que el temporizador expira.

Además, en el estado de invencibilidad, se ajusta las capas de colisión para que Mario pueda atravesar enemigos sin interactuar físicamente con ellos, evitando así que se acumulen daños adicionales durante este período vulnerable.

### Implementación del Sistema de Puntuación y Audio.

Antes de detallar los puntos mencionados en este apartado, voy a explicar como he realizado la implementación y gestión de la bandera en el juego, es decir, el proceso realizado para que Mario interactúe correctamente con este elemento al final de cada nivel:

1. **Creación de Elementos Gráficos:** Como en todos los apartados anteriores, inicio configurando los sprites necesarios para la bandera, sus movimientos y la asta de la bandera.



Ilustración 10.1

1. **Animaciones de Mario para Bajar la Bandera:** Se ha implementado una animación específica donde Mario se desliza hacia abajo por el poste.
2. **Interacción de Mario con la Bandera:**



Ilustración 10.1

**Inicio del Descenso:** Al interactuar con la bandera, se detecta la colisión de Mario con ella y activa el proceso de descenso.



Ilustración 10.1

**Control de Descenso**: Durante el descenso, Mario se mueve hacia abajo a una velocidad constante que está programada.

**Bajar la Bandera:** Simultáneamente con el descenso de Mario, la bandera misma, que inicialmente está en la parte superior del poste, también desciende. Esto se logra movimiento la bandera hacia abajo en el poste a la misma velocidad que Mario, creando la ilusión de que Mario la está bajando realmente.

**Acciones al Finalizar el Descenso:** Una vez que Mari llega al final del poste, se activan varias acciones:

* **Temrina la Animación de Descenso:** La animacion de descenso se detiene y se activa una nueva animación donde Mario se desmonta del poste, esto se hace transladando la posición de Mario un poco en el eje x.
* **Caminata Hacia el Castillo:** Después de bajarse del poste, Mario entra automáticamente en una animación de caminata que lo lleva hacia el sieguiente nivel, simbolizando la conclusión del nivel actual.

**SISTEMA DE PUNTUACIÓN**

Para la gestión del sistema de puntos he utilizado un patrón de diseño llamado ‘Singleton’. Este patrón asegura que una clase tiene una única instancia y proporciona un punto de acceso global a ella. Esta técnica es útil para manejar componentes como el Score Manager, que necesita ser accesible de manera global y persistente a través de diferentes escenas del juego.

Para implementar el patrón Singleton en el Score Manager, he seguido los siguientes pasos:

1. **Creación de la Instancia Singleton:** He definido una variable privada estática de la misma clase que se encarga de almacenar la instancia, además creo un método público que permite el acceso a esta instancia.



Ilustración 10.1

1. **Acceso y Gestión de Puntos:** Se puede modificar la puntuación desde cualquier parte del juego.

En el juego, la gestión de puntos para cada interacción específica como recoger ítems, destruir enemigos y la acción de la bandera al finalizar el nivel se maneja de la siguiente manera:

|  |  |
| --- | --- |
| Acción | Puntos |
| Recoger Moneda | 200 |
| Recoger Flor de Fuego | 1000 |
| Recoger Seta Mágica | 1000 |
| Recoger Estrella | 1000 |
| Destruir Goomba | 100 |
| Destruir Koopa | 100 |
| Destruir Planta Piraña | 200 |

Tabla 10.1

Al final de cada nivel, cuando Mario baha por el asta de la bandera, los puntos se otorgan según la altura a la que Mario toca la bandera. El asta de la bandera se divide en segmentos, y cada segmento otorga diferentes cantidades de puntos:



Ilustración 10.1

* Zona Superior del Asta: 5000 puntos.
* Segunda Zona desde arriba: 2000 puntos.
* Tercera Zona desde arriba: 800 puntos.
* Cuarta Zona desde arriba: 400 puntos.
* Base del Asta: 100 puntos.

El cálculo de la altura y la asignación de puntos se realizan mediante la posición de Mario en relación con el asta de la bandera al momento de colisionar con ella. Se verifica la posición de Mario en relación con dividir la altua total del asta en cinco segmentos, asignando puntos según la sección alcanzada.

Estos puntos se manejan a través del ‘ScoreManager’, que utiliza el patrón Singleton para asegurarse de que la gestión de puntos sea ccesible globalmente y persista entre diferentes escenas y niveles.

Para enriquecer la retroalimentación visual para los jugadores, he implementado efectos visuales que muestran los puntos obtenidos por diversas acciones dentro del juego, ya sea matando a los enemigos o recogiendo ítem.

1. Primero se ha diseñado y utilizado sprites individuales para diferentes cantidades de puntos (100,200,400,800,1000,2000,5000).
2. Para la implementación técnica, se ha creado un prefab que incluye estos sprites y mediante un script se controla la dinámica de aparición y ocultamiento de los sprites.

Estos sprites aparecen y se mueven ligeramente hacia arriba antes de desvanecerse, creado un efecto visual que simila el acto de ganar puntos.

**AUDIO**

En el desarrollo del juego, se ha dado especial atención a la implementación del sistema de audio. El tratamiento del audio en Unity se ha estructurado en tres categorías principales: música, efectos de sonido y sonidos de ambiente. Cada tipo de audio se ha gestionado de manera específica para optimizar tanto la calidad como el rendimiento del juego.

**Configuración de Audio:**

1. **Mono vs Estéreo:**  Se ha forzado la mayoría de los efectos de sonido a mono para evitar desajustes en la localización especial del sonido en un entorno 2D. Esto también ayuda a reducir el tamaño del archivo de audio.
2. **Tipo de Carga:** Dependiendo de la frecuencia y duración de cada sonido, se ha elegido entre las tres configuraciones de carga de Unity:

**Descomprimir al Cargar**: Utilizado para efectos de sonido cortos y repetitivos, como los pasos, permitiendo una respuesta instantánea sin retrasos en la reproducción.

**Comprimido en Memoria**: Ideal para sonidos que ofrecen un balance entre tamaño y frecuencia de uso.

**Streaming:** Reservado para archivos de audio de gran tamaño y uso poco frecuente, como largas pisyas de música o diálogos, monimizando el uso de la memoria RAM.



Ilustración 10.1

1. **Compresión de Audio:** Se han seleccionado formatos de compresión adecuados para cada tipo de sonido, considerando la calidad necesaria y el impacto en el rendimiento. He utilizado PCM para efectos donde la inmediatez es crucial, y formatos comprimidos como Vorbis para otros sonidos donde la fidelidad es menos crítica pero deseable.



Ilustración 10.1

**Integración de Audio**

* **AudioSource y AudioClips:** Se utiliza el componente de AudioSource para reproducir sonidos. Los AudioClips correspondientes se asignaron a cada AudioSource según el tipo de sonidos (efectos, música, ambiente), configurando parámetros como loop, volumen y pitch (velocidad del sonido) de acuerdo con las necesidades de cada sonido.

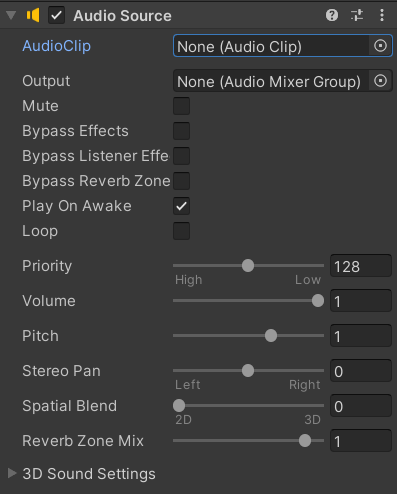


Ilustración 10.1

* **Gestión de Audio**: Se ha implementado un ‘AudioManager’ siguiendo el patron Singleton, centralizando la gestión de todos los sonidos del juego. Esto permite un acceso fácil y controlado a la reproducción de sonidos desde cualquier parte del código.



Ilustración 10.1

### Desarrollo de la Interfaz de Usuario y Temporizador

Para abordar la gestión e implementación de la interfaz de usuario en el desarrollo del juego, he creado un sistema que permite interactuar efectivamente con Mario a través de diversos elementos visuales y configuraciones que reflejan dinámicamente el estado del juego y las acciones del jugador.

1. **Componentes Básicos de la UI**

**Canvas:** Es el elemento principal que sirve como contener para todos los componentes de la UI. Se utiliza para organizar y renderizar visualmente todos los elementos de la interfaz en la pantalla.

**EventSystem**: Es el sistema que gestiona los eventos de la UI, como clics de botón o entradas táctiles, asegurando que los inputs del usuario sean procesados correctamente.

**Render Modes:** El canvas permite varias configuraciones de renderizado, pero he elegido la opción de ‘Screen Space – Overlay’, que dibuja la UI por encima de todo lo demás de la pantalla.



Ilustración 10.1

1. **Adaptabilidad y Escalabilidad:**

Se ha utilizado el componente ‘Canvas Scaler’ para asegurar que nuestra UI se vea consistentemente en diferentes dispositivos y resoluciones.

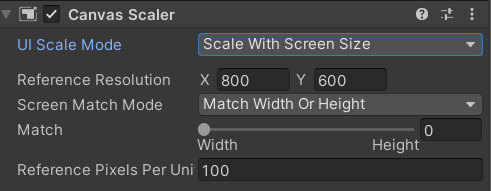


Ilustración 10.1

1. **Elementos Interactivos**

**Botones, Textos e Imágenes:** Estos son los elementos fundamentales para mostrar información como puntuación, niveles, estado del jugador y demás. Además, permite interacciones simples como presionar botones para navegar por el juego o cambiar configuraciones.

**Text Mesh Pro**: Se utiliza para una mejor calidad visual del texto, ofreciendo claridad independiente de la resolución, lo cual es esencial para interfaces como la puntuación y los diálogos dentro del juego.

1. **Anclajes y Ajustes Dinámicos**

Además, he implementado anlcajes dinámicos (‘Anchors’) para que los elementos de UI mantengan su posición relativa y tamaño proporiconal respecto al Canvas.



Ilustración 10.1

Para la **Creación de la UI, p**rimero se ha utilizado elementos de la UI de Unity como TextMeshPro para los elementos textuales que muestran la información de puntuación, monedas, nivel en el que nos encontramos y el tiempo. Estos elementos se organizan dentro de un canvas para mantener una jerarquía clara y manejable.



Ilustración 10.1

La implementación se ha llevado a cabo mediante los siguientes scripts principales que interactúan con la lógica del juego:

* ScoreManager: Gestiona todas las modificaciones de la puntuación.
* HUD: Controla la visualización de la información de puntuación, conteo de moneda, nivel actual y temporizador. Este script actualiza dinámicamente los datos en pantalla en respuesta a cambios notificados por el ‘ScoreManager’ y otros componentes del juego.
* NivelManager: Coordina el incremento de monedas y gestiona otros elementos del nivel, como el seguimiento del tiempo y la actualización del HUD correspondiente. A través de este manager, se comunica cualquier cambio necesario en la representación visual de la interfaz.

Además, para la animación de imágenes en la HUD, como la de la moneda, se ha utilizado el script ‘AnimacionImagen’, que cicla a través de un array de sprites, mostrándolos secuencialmente en un componente ‘Image’ para crear una animación fluida.



Ilustración 10.1

**TEMPORIZADOR**

Para gestionar el temporizador en el juego, se ha configurado el HUD para mostrar el tiempo restante, que se maneja mediante ‘NivelManager’. El temporizador decrementa cada segundo en tiempo de juego, que es más rápido que un segundo real, para mantener la sensación de desafío.

**Implementación del Temporizador**

1. **HUD:** Se ha modificado el HUD para incluir un display del tiempo, que se actualiza dinámicamente. Se emplea una conversión de los valores float del temporizador a enteros para que la visualización sea en números enteros sin decimales, manteniendo la estética clásica de los videojuegos.



Ilustración 10.1

1. **NivelManager:** Gestiona el temporizador de cada nivel. Al iniciar, se establece un tiempo incial que decrece cada segundo de juego. Se usa la función ‘Time.deltaTime’ para asegurarse que el descuento es independiente de la tasa de frames, lo que proporciona una experiencia uniforme independientemente del hardware.
2. **Finalización del Tiempo:** Si el temporizador llega a cero, causa la muerte de Mario si no se ha completado el nivel.

Cuando Mario completa un nivel antes de que el tiempo asignado expire, el tiempo restante se convierte en puntos adicionales para el jugador (cada segundo son 50 puntos).

Para impleméntalo, una vez que Mario alcanza el punto final del nivel, se invoca un método que detiene el temporizador y calcula los puntos finales a partir del tiempo restante. Estos puntos se añaden al puntaje total del jugador mediante el ‘ScoreManager’.



Ilustración 10.1

## Enriquecimiento y Expansión de Juego

### Construcción de Niveles

### Configuración de la Cámara y Ajustes Visuales

## Evaluación y Adaptación

# Pruebas y validación

## Pruebas de Funcionalidad

## Pruebas de Interacción y Usabilidad

# finalización y transiciones

## Ajustes Finales y Preparación de Transiciones de Nivel

## Implementación del Menú Principal y Opciones de Juego.

# Apéndices

## Guía original del Trabajo Fin de Título

**Contenido**: incluir la propuesta o **guía original del TFG/TFM** (publicada en la web de la EPS en el momento de la convocatoria), así como el histórico de modificaciones que haya podido tener dicha propuesta (título, objetivos, etc.).

## Manuales de usuario

**Contenido**: en caso de desarrollar algún software o sistema, es conveniente incluir unas instrucciones de uso que permitan probar todas las funcionalidades implementadas.

# Bibliografía

Pressman, R. (2010). *Ingeniería del Software.* McGraw-Hill.