

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](#_Toc170286593)

[1.1 Propósito del Proyecto 13](#_Toc170286594)

[1.2 Alcance del Proyecto 14](#_Toc170286595)

[2 Herramientas y tecnologías 16](#_Toc170286596)

[2.1 Unity - Motor de Desarrollo 16](#_Toc170286597)

[2.2 Herramientas Adicionales 17](#_Toc170286598)

[3 Diseño y planificación 19](#_Toc170286599)

[3.1 Metodología de Desarrollo y Planificación temporal 19](#_Toc170286600)

[3.2 Estimaciones de recursos y costes 20](#_Toc170286601)

[4 Diseño inicial 22](#_Toc170286602)

[4.1 Visión General del Juego 22](#_Toc170286603)

[4.2 Mecánicas del Juego 23](#_Toc170286604)

[4.3 Personajes 25](#_Toc170286605)

[4.4 Interfaz de Usuario 27](#_Toc170286606)

[5 Desarrollo del juego 29](#_Toc170286607)

[5.1 Configuración Inicial 29](#_Toc170286608)

[5.1.1 Configuración del Proyecto en Unity 29](#_Toc170286609)

[5.2 Desarrollo de Características 30](#_Toc170286610)

[5.2.1 Creación y Manipulación de Sprites 30](#_Toc170286611)

[5.2.2 Implementación de Físicas y Colisiones 33](#_Toc170286612)

[5.2.3 Animación de Personajes y Objetos 37](#_Toc170286613)

[5.2.4 Creación y Gestión de Enemigos 41](#_Toc170286614)

[5.2.5 Implementación de Transformaciones y Power-ups 45](#_Toc170286615)

[5.2.6 Gestión de Items, Poderes Especiales y Bloques 47](#_Toc170286616)

[5.2.7 Implementación del Sistema de Puntuación y Audio. 55](#_Toc170286617)

[5.2.8 Desarrollo de la Interfaz de Usuario y Temporizador 65](#_Toc170286618)

[5.3 Enriquecimiento y Expansión de Juego 69](#_Toc170286619)

[5.3.1 Construcción de Niveles 69](#_Toc170286620)

[5.3.2 Configuración de la Cámara y Ajustes Visuales 93](#_Toc170286621)

[6 Apéndices 101](#_Toc170286622)

[6.1 Guía original del Trabajo Fin de Título 101](#_Toc170286623)

[6.2 Manuales de usuario 101](#_Toc170286624)

[7 Bibliografía 103](#_Toc170286625)

Índice de ilustraciones

[Ilustración 1. Herramientas Adicionales 18](#_Toc168394977)

[Ilustración 5.1. Goomba 26](#_Toc168394978)

[Ilustración 5.1. Koopa Troopa 26](#_Toc168394979)

[Ilustración 5.1. Planta Piraña 26](#_Toc168394980)

[Ilustración 5.1. HUD 27](#_Toc168394981)

[Ilustración 5.1. Alertas Visuales 28](#_Toc168394982)

[Ilustración 5.1. Versión Unity 29](#_Toc168394983)

[Ilustración 5.2. Estructura del Proyecto 30](#_Toc168394984)

[Ilustración 5.3. Texture Type 30](#_Toc168394985)

[Ilustración 5.4. Sprite Editor 31](#_Toc168394986)

[Ilustración 5.5. Filter Mode 32](#_Toc168394987)

[Ilustración 5.5. Sprite Renderer 32](#_Toc168394988)

[Ilustración 5.6. Rigidbody 2D 33](#_Toc168394989)

[Ilustración 5.7. Tipo de Rigidbody 2D 33](#_Toc168394990)

[Ilustración 5.8. Physics Material 2D 35](#_Toc168394991)

[Ilustración 5.9. Control de Movimiento 35](#_Toc168394992)

[Ilustración 5.10. Mecánica de Salto 36](#_Toc168394993)

[Ilustración 5.11. Detección del suelo 36](#_Toc168394994)

[Ilustración 5.12 38](#_Toc168394995)

[Ilustración 5.13 38](#_Toc168394996)

[Ilustración 5.14 38](#_Toc168394997)

[Ilustración 5.15 41](#_Toc168394998)

[Ilustración 5.16 43](#_Toc168394999)

[Ilustración 5.17 45](#_Toc168395000)

[Ilustración 5.18 **¡Error! Marcador no definido.**](#_Toc168395001)

[Ilustración 5.19 48](#_Toc168395002)

[Ilustración 5.20 **¡Error! Marcador no definido.**](#_Toc168395003)

[Ilustración 5.21 50](#_Toc168395004)

[Ilustración 5.22 53](#_Toc168395005)

[Ilustración 5.23 55](#_Toc168395006)

[Ilustración 5.24 55](#_Toc168395007)

[Ilustración 5.25 56](#_Toc168395008)

[Ilustración 5.26 58](#_Toc168395009)

[Ilustración 5.27 59](#_Toc168395010)

[Ilustración 5.28 61](#_Toc168395011)

[Ilustración 5.29 62](#_Toc168395012)

[Ilustración 5.30 63](#_Toc168395013)

[Ilustración 5.31 63](#_Toc168395014)

[Ilustración 5.32 64](#_Toc168395015)

[Ilustración 5.33 65](#_Toc168395016)

[Ilustración 5.34 65](#_Toc168395017)

[Ilustración 5.35 66](#_Toc168395018)

[Ilustración 5.36 66](#_Toc168395019)

[Ilustración 5.37 67](#_Toc168395020)

[Ilustración 5.38 68](#_Toc168395021)

[Ilustración 5.39 68](#_Toc168395022)

[Ilustración 5.40 69](#_Toc168395023)

[Ilustración 5.41 70](#_Toc168395024)

Índice de tablas

[Tabla 10.1 Costes Recursos Humanos 20](#_Toc168310764)

[Tabla 10.1 Costes Recursos Materiales y Software 21](#_Toc168310765)

[Tabla 10.1 Coste total del proyecto 21](#_Toc168310766)

[Tabla 10.1 58](#_Toc168310767)

# 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.

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 sus características 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]**.

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 y GIMP para edición más detallada de gráficos.

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 **[2]**.

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 **[3].**

1. **GIMP**

**Edición Gráfica Avanzada:** Para la edición más detallada y refinada de sprites, he utilizado GIMP, un programa de manipulación de imágenes gratuito y de código abierto.

**Herramientas de Diseño Profesional:** GIMP proporciona herramientas avanzadas para la edición de imágenes, como capas, máscaras, filtro y pinceles, facilitando la modificación de los sprites para el juego **[4].**



Ilustración 1. Herramientas Adicionales

# Diseño y planificación

## Metodología de Desarrollo y Planificación temporal

\*\*\*\*MODIFICAR

Para el desarrollo del remake de Super Mario Bros, he optado por una combinación de la metodología **Scrum**, que es una forma de metodología ágil, con un enfoque estructurado en el desarrollo incremental e iterativo. Scrum se caracteriza por su flexibilidad y capacidad para adaptarse a cambios rápidos y continuos en los requisitos del proyecto [5].

**Implementación de la Metodología Scrum:**

Scrum se implementa a través de ciclos de desarrollo cortos denominados ‘sprints’. Cada sprint incluye fases de planificación, desarrollo, revisión y retrospectiva. A continuación, se detalla cómo se ha aplicado Scrum en este proyecto:

1. **Planificación y Configuración Inicial:**

**Configuración del Proyecto en Unity:** Configuración inicial del entorno de desarrollo, definición de parámetros de proyecto y establecimiento de las herramientas esenciales.

**Definición de Alcance y Requisitos:** Identificación de los requisitos funcionales, mecánicas del juego, niveles y elementos gráficos a desarrollar.

1. **Desarrollo de Características:**

**Creación y Manipulación de Sprites:** Diseño y ajuste de los sprites del personaje principal, enemigos y elementos del escenario utilizando herramientas como Paint y GIMP.

**Implementación de Mecánicas de Juego:** Programación de mecánicas fundamentales como el movimiento de Mario, saltos y colisiones utilizando el sistema de físicas de Unity.

**Animación de Personajes y Objetos:** Utilización de las herramientas de animación de Unity para crear animaciones detalladas y dinámicas.

**Integración de Sistemas de Puntuación y Progreso:** Desarrollo de la lógica para el manejo de puntuaciones, vidas y progreso del jugador a través de los niveles.

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

**Desarrollo de Niveles Específicos:** Utilización de Tilemap y otros recursos de Unity para construir y gestionar los niveles, incluyendo la configuración de obstáculos y elementos interactivos.

**Implementación de Audio y Efectos Visuales:** Integración de efectos sonoros y música, mejorando la experiencia del juego.

## 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.

\*\*\*\*\*MODIFICAR

**Recursos Humanos:**

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

Tabla 3.1 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 |
| GIMP | Programa de edición de imágenes avanzada | 0 EUR |
| Ordenador Personal | Necesario para la realización del proyecto | 800 EUR |
| Microsoft Office Word | Para documentación | 69 EUR (Suscripción anual) |

Tabla 3.2 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 3.3 Coste total del proyecto

La estimación de costes destaca que la mayor parte del proyecto se puede completar sin incurrir en costes directos significativos, gracias al uso de software gratuito y recursos académicos.

# 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

Para este remake de Super Mario Bros, he conservado numerosas mecánicas clásicas mientras he añadido varias innovaciones que enriquecen la experiencia de juego. El objetivo ha sido mantener la autenticidad del juego original al tiempo que se introduce una dimensión moderna a través de mejoras técnicas y funcionales.

**Controles Básicos**

* **Movimiento:** Mario puede moverse hacia la izquierda o derecha usando los controles direccionales, facilitando la exploración e interración con el entorno y los enemigos.
* **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 velocidad y la distancia de sus saltos. Correr también es crucial para superar ciertos obstáculos y enemigos más rápidamente.
* **Agacharse:** Mario puede agacharse para evitar ataques o para entrar en tuberías.
* **Disparo de Bolas de Fuego:** Al obtener la Flor de Fuego, Mario puede lanzar bolas de fuego, lo que permite atacar enemigos desde una distancia.

**Romper Bloques desde Arriba:** Cuando Mario es Super Mario o posee la capacidad de fuego, puede romper ciertos bloques golpeándolos desde abajo.

**Interraciones con el Entorno**

* **Bloques Interactivos:** Golpearlos puede revelar monedas, power-ups o secretos, siendo parte fundamental de la exploracion y progresión.
* **Tuberías:** Sirven como acceso a áreas secretas y como obstáculos estratégicos.

**Power-ups y Objetos**

* **Super Champión:** Aumenta el tamaño a Mario, permitiéndole soportar un golpe extra y romper bloques.
* **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:** Ofrece un periodo breve de invulnerabilidad donde Mario puede derrotar enemigos al contacto.
* **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.

**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.



Ilustración 4.1. Goomba

* **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.

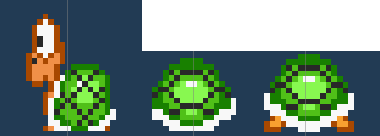


Ilustración 4.2. Koopa Troopa

* **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.



Ilustración 4.3. Planta Piraña

## Interfaz de Usuario

El diseño de la UI en este remake de Super Mario Bros está centrado en la claridad y funcionalidad, asegurando que los jugadores tengan acceso fácil a la información esencial mientras juegan.

1. **HUD**

**Indicador de Puntos:** Muestra la puntuación actual del jugador, actualizando en tiempo real conforme Mario recolecta monedas, derrota enemigos o alcanza hitos importantes dentro del nivel, como capturar la bandera.

**Contador de Monedas:** Informa al jugador sobre el número de monedas recolectadas, lo que es crucial para el seguimiento de progreso hacia obtener vidas extras.

**Indicador del Nivel:** Muestra el nivel actual en el que se encuentra Mario, proporcionando a los jugadores una referencia clara de su progreso en el juego

**Temporizador de Nivel:** Un reloj regresivo que añade urgencia y desafío al juego, empujando a los jugadores a completar cada nivel antes de que se agote el tiempo.



Ilustración 4.4. HUD

1. **Menús y Pantallas**

**Pantalla - Inicio:** Una introducción visual al juego, con opciones para iniciar o continuar un juego guardado.

**Pantalla - Nivel Superado**

**Pantalla - Perder Vida:** Permite a los jugadores detener temporalmente el juego y acceder a opciones sin perder el progreso.

**Pantalla - G*ame Over*:** Se muestra cuando Mario pierde todas sus vidas, ofreciendo opciones para reiniciar el nivel o salir del juego.

1. **Feedback Visual y Auditivo**

**Alertas Visuales:** Información como la obtención de un nuevo power-ups o la pérdida de una vida se destacan mediante animaciones y cambios visuales en la HUD. Esto incluye la visualización de la puntuación al obtener un ítem, eliminar a un enemigo, y captura la bandera, así como efectos visuales al obtener un power-up o recibir un golpe.



Ilustración 4.5. Alertas Visuales

**Efectos Sonoros:** Cada acción importante, como saltar, recoger monedas, o activar un power-up, está acompañada de efectos de sonido distintivos que mejoran la retroalimentación inmediata al jugador.

# 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 5.1. Versión Unity

**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, sprites, fuentes de texto, animaciones, prefabs, archivos de audio y demás.
* **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 5.2. Estructura del Proyecto

## Desarrollo 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 **[6]:**

**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 “Sprites” dentro de “Assets” del proyecto en Unity.

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



Ilustración 5.3. Texture Type

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 spritesheets 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 5.4. Sprite Editor

* **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 vean menos pixelados y más suaves al ser escalados.

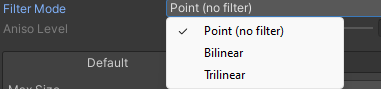


Ilustración 5.5. Filter Mode

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 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.

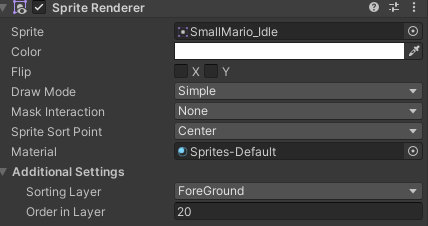


Ilustración 5.6. Sprite Renderer

### 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 **[7]**.

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 5.7. Rigidbody 2D

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 5.8. Tipo de Rigidbody 2D

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 5.2. Layer Collision Matrix

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 5.9. Physics Material 2D

1. **Script de Movimiento (Mover.cs)**

**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 5.10. Control de Movimiento

**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 5.11. Mecánica de Salto

1. **Colisiones (Colisiones.cs)**

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 **[8].**



Ilustración 5.12. Detección del suelo

**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 **[9].**
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 **[10].**
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 **[11].**



Ilustración 5.13. Animation

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



Ilustración 5.14. Animator

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 5.15. Configuración Animación

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.

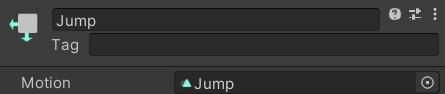


Ilustración 5.16. Integración con el Sistema de Animación

* **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.

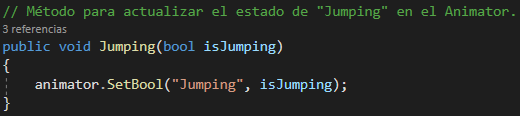


Ilustración 5.17. Control Dinámico de Animaciones

* **Sincronización con la Física:** Las animaciones se sincronizan con los estados físicos del personaje, verificando condiciones como si no se está en el suelo y saltando a través del script de colisiones.



Ilustración 5.18. Transición de Estados - Animator

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.

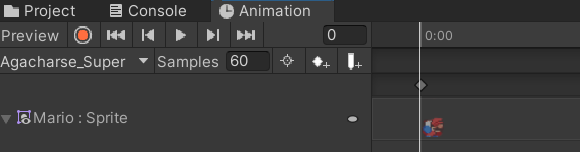


Ilustración 5.19. Animación Agachado

* **Restricciones de movieminto:** Mientras está agachado, Mario no puede moverse.

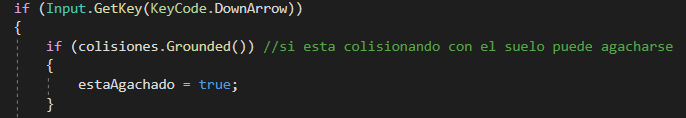


Ilustración 5.20. Animación Agachado

* **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.

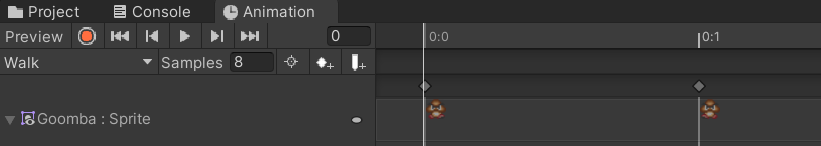


Ilustración 5.21. Animation Enemigo (Goomba)

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

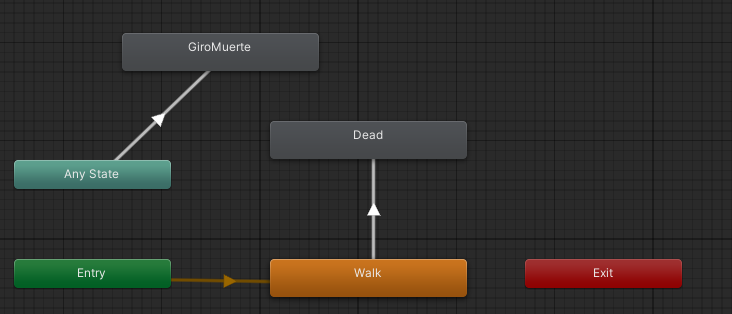


Ilustración 5.22. Animator Enemigo (Goomba)

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.



Ilustración 5.23. Componentes Físicos del Enemigo (Goomba)

1. **Script de Movimiento:** Utilizo una estructura de clases con una clase base ‘Enemigo’ que define propiedades y métodos comunes. De esta clase derivan 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.

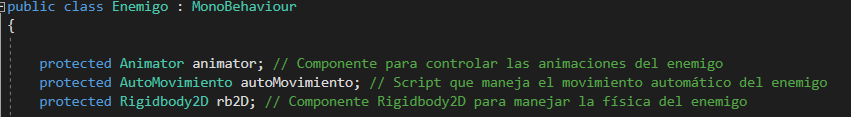


Ilustración 5.24. Clase ‘Enemigo’

**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.

1. **Gestión de Colisiones:** Configuro las colisiones para que el enemigo pueda interactuar de manera adecuada con elementos del juego.

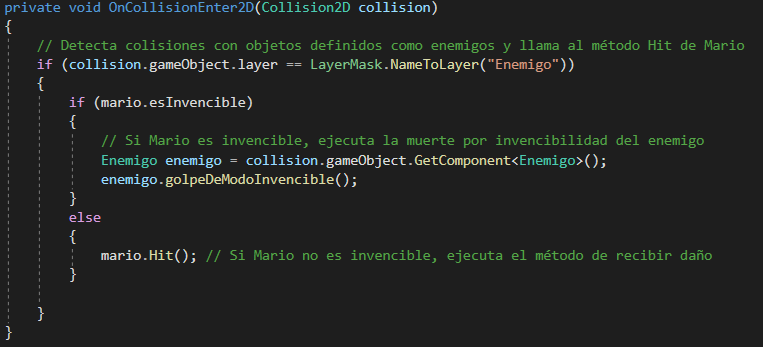


Ilustración 5.25. Gestión de Colisiones Enemigo

1. **Interacciones con el Jugador:** Implemento lógicas específicas para que cada tipo de enemigo responda de manera única cuando interactúa con el jugador.

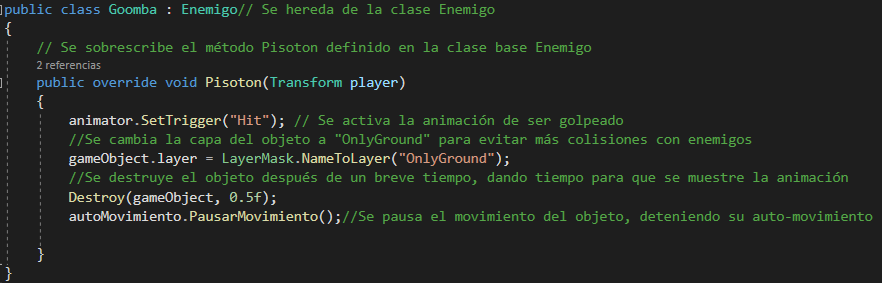


Ilustración 5.26. Clase Derivada de Enemigo (Goomba)

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**

**Mostar y esconderse:** 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.

1. **Substates para Organizar Animaciones**

Utilizo substates dentro de ‘Animator’ para agrupar animaciones similares (Small, Super, Fuego), 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.

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



Ilustración 5.27. Animator Mario

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.

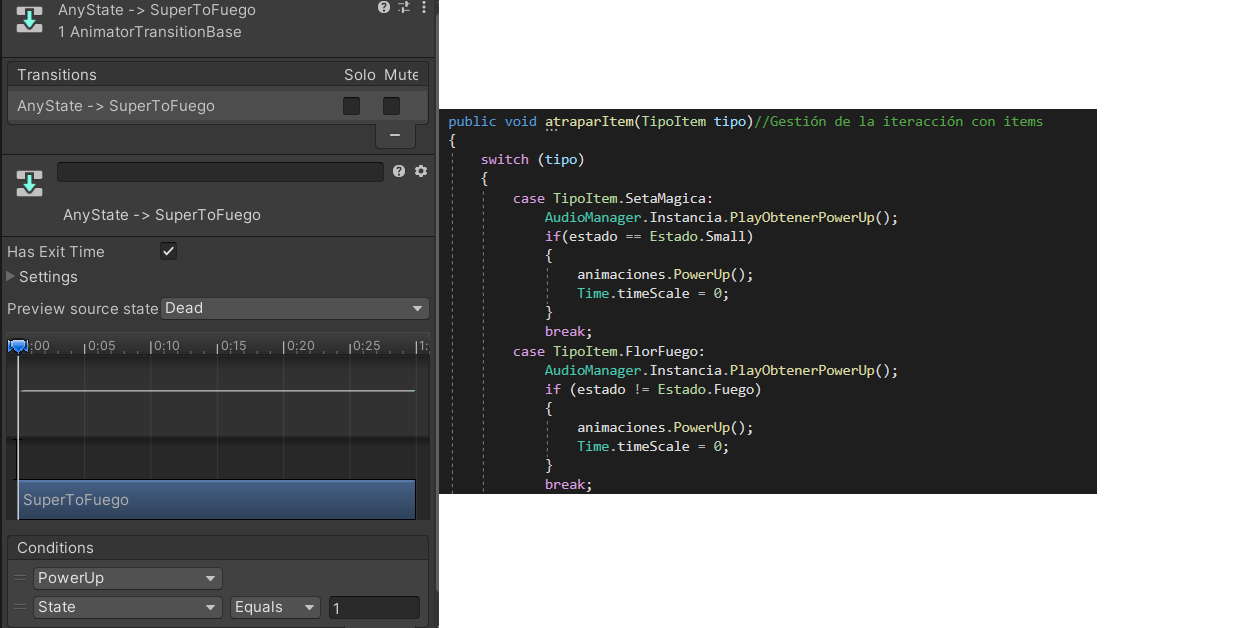


Ilustración 5.28. Gestión Power-Ups

**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 **[12].**

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

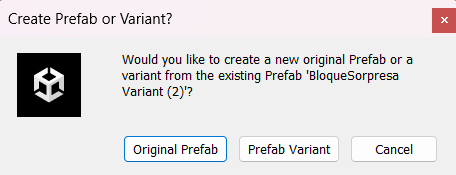


Ilustración 5.29. Prefab Variant

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.

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.

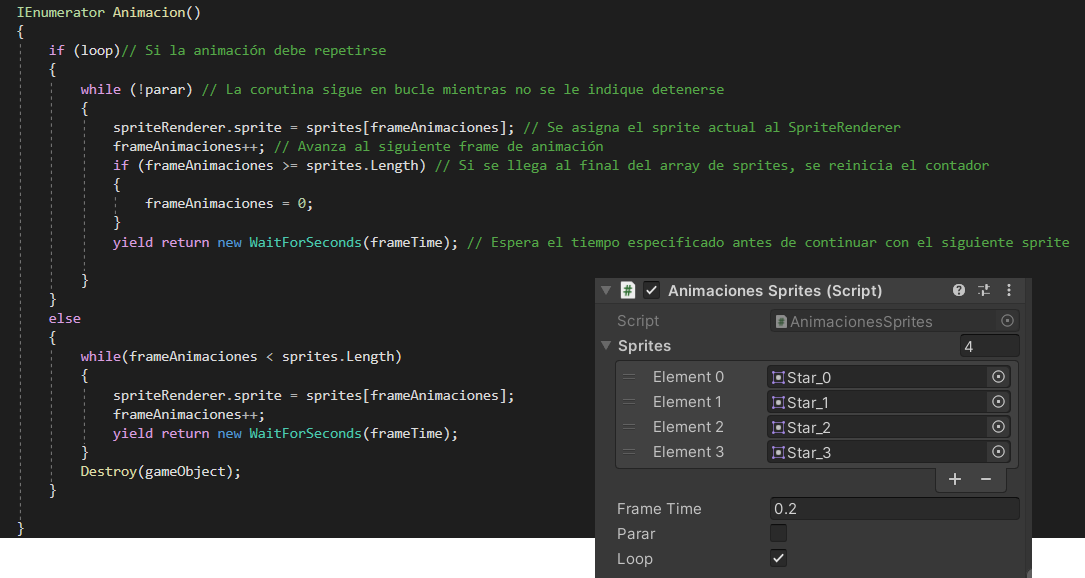


Ilustración 5.30

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 agotados los recursos.

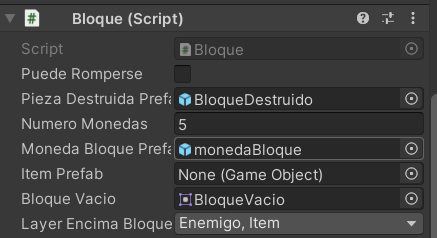


Ilustración 5.31. Parámetros Bloque

1. **MonedaBloque:** Gestiona las monedas que aparecen al golpear ciertos bloques, animando su trayectoria hacia arriba y su desaparición.
2. **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.
3. **AutoMovmiento:** Estos se aplican a los ítems, que se mueven independientemente tras ser libreados de un bloque.

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.



Ilustración 5.32. Parámetros Bola de Fuego

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 la bola de fuego desintegrándose.

El comportamiento de los enemigos frente a la bola de fuego 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 5.33. Prefabs Bola de Fuego

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 arsprites 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 cambian 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.

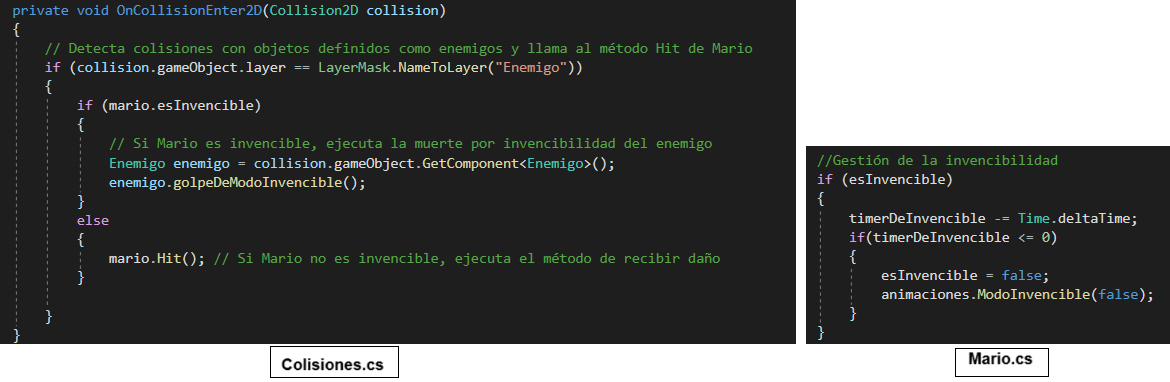


Ilustración 5.34. Modo Invencible

1. **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.

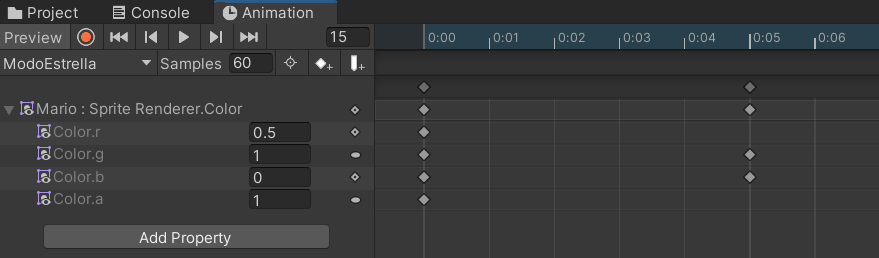


Ilustración 5.35. Animación de Invencibilidad

1. **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.
2. **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 5.36. Detección de Enemigos – Items encima del Bloque

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 está 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.

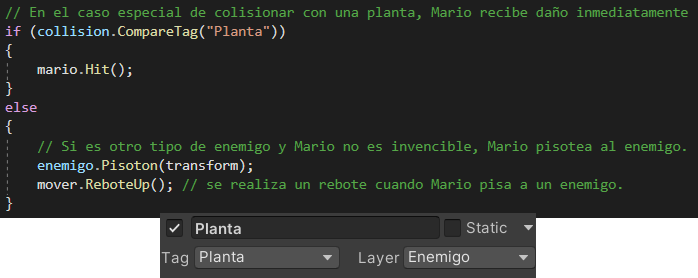


Ilustración 5.37. Interacción con la Planta Piraña

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.

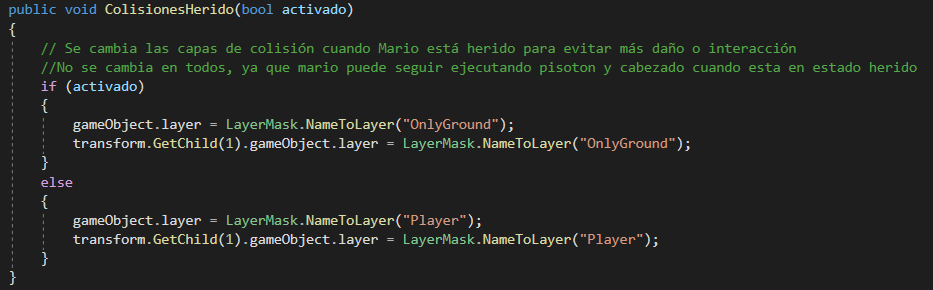


Ilustración 5.38. Colisiones cuando Mario es golpeado

### Implementación del Sistema de Puntuación y Audio.

Antes de detallar los puntos mencionados en este apartado, voy a explicar cómo 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 5.39. Sprite de la bandera

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 5.40. Animator - Escalar

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



Ilustración 5.41. Condicion de la animación Escalar

**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. Esto se gestiona en el script de VictoriaBandera.

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

* **Termina 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.

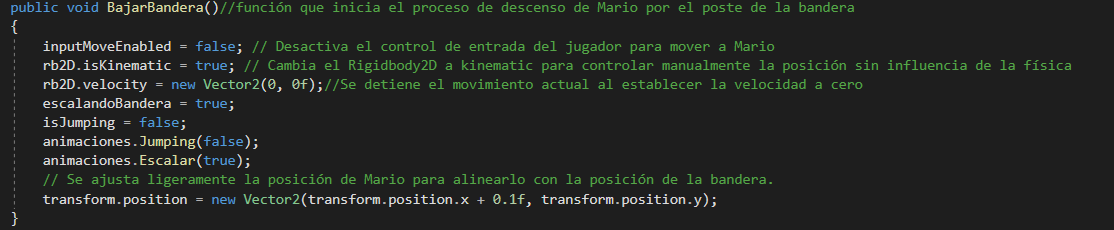


Ilustración 5.42. Función para Bajar de la bandera

* **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.

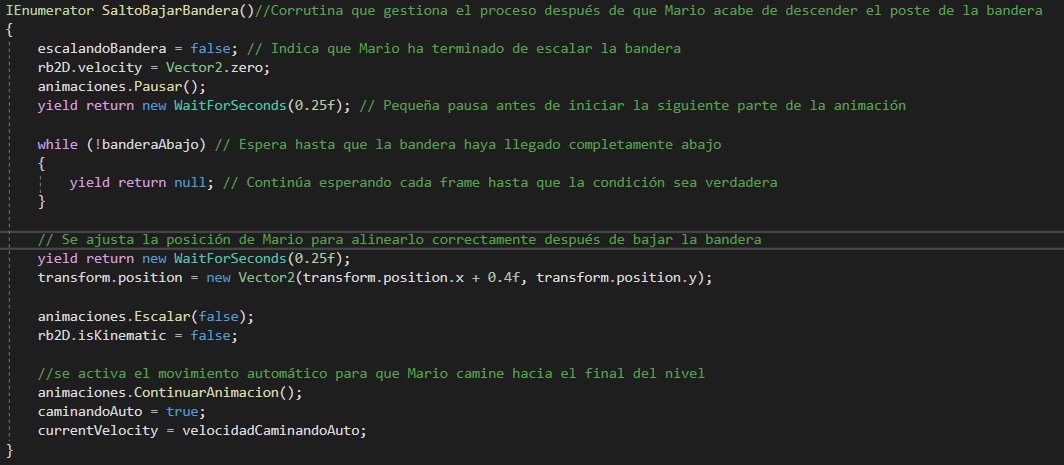


Ilustración 5.43. Función para Saltar de la bandera

**SISTEMA DE PUNTUACIÓN**

Para la gestión del sistema de puntos he utilizado un patrón de diseño llamado ***‘Singleton’* [13]**. 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 5.44. Score Manager

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 5.1. Puntos para cada interacción específica

Al final de cada nivel, cuando Mario baja 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 5.45. Alturas a las que Mario puede tocar la bandera

* 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: 200 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 altura 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 acesible globalmente y persista entre diferentes escenas y niveles.

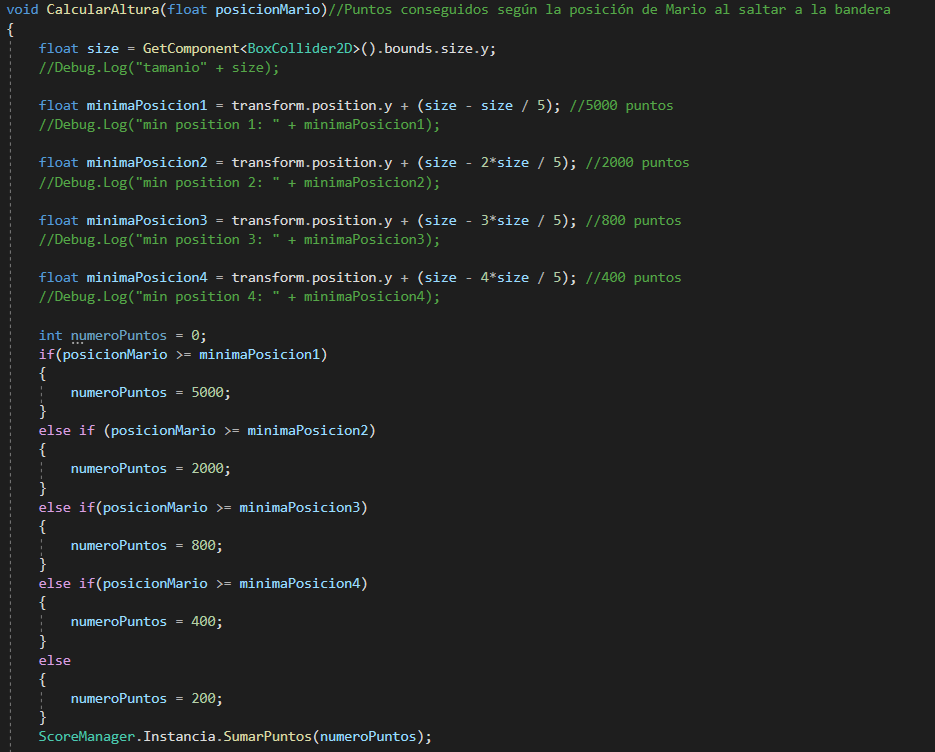


Ilustración 5.46. Calculo de la altura y asignación de puntos de la bandera

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.



Ilustración 5.47. Efecto 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 **[14][15].**

**Configuración de Audio:**

El audio del juego se ha configurado con diferentes métodos de carga para optimizar el rendimiento y la calidad:

* **Descomprimir al Cargar**: Esta técnica se ha utilizado para efectos de sonido cortos y frecuentes, como los sonidos de saltos y recolección de monedas, proporcionando una reproducción sin retardos.
* **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 pistas de música o diálogos, monimizando el uso de la memoria RAM.



Ilustración 5.48. Tipos de Carga de Audio

**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 5.49. Tipos de compresión de Audio

**Integración de Audio**

La integración y gestión del audio se han centralizado a través del componente ‘AudioManager’, implementado siguiendo el patrón Singleton. Esto asegura que solo existe una instancia de este gestor en todo el juego, facilitando el acceso y control de los sonidos desde cualquier parte del código.

* **AudioSource y AudioClips**: Se han utilizado múltiples AudioSource para separar los sonidos de efectos, música y ambiente.

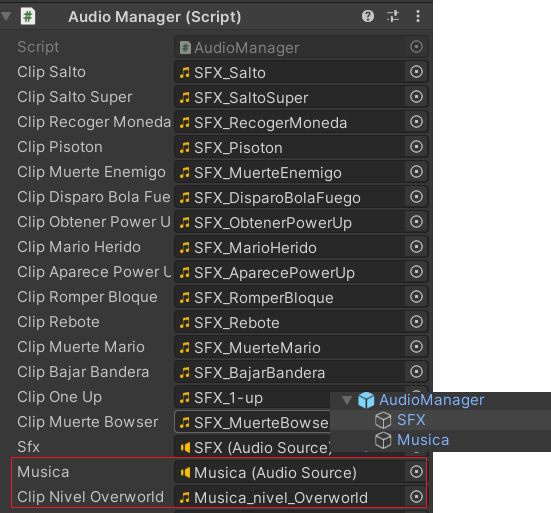


Ilustración 5.50. Audio Source

El AudioManager no solo gestiona la reproducción de sonidos efectivos, sino que también maneja la transición y adaptación de la música ambiental a través de diferentes escenarios del juego:

* **Transiciones Musicales:** Se manejan cambios dinámicos en la música de fondo según el contexto del juego utilizando métodos específicos que ajustan la música a la atmósfera deseada.

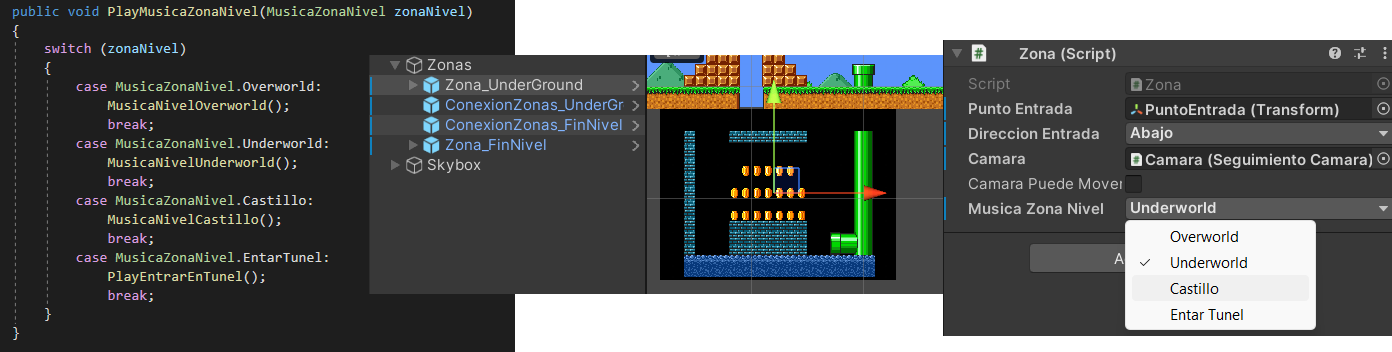


Ilustración 5.51. Audio Manager

* **Modo Invencible:** Durante este modo, se reproduce una pista especial, y el sistema está diseñado para volver automáticamente a la música de fondo del nivel una vez que el modo invencible termina.

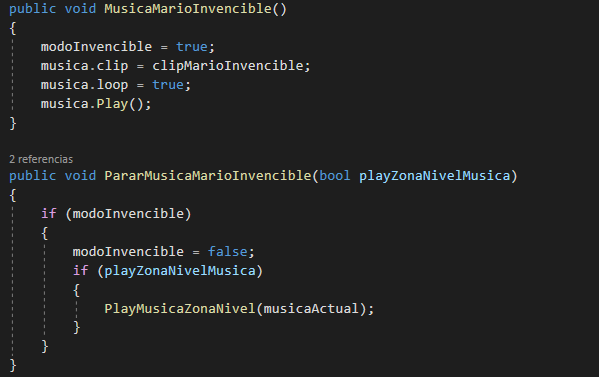


Ilustración 5.52. Audio Manager

### 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 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 contenedor para todos los componentes de la UI. Se utiliza para organizar y renderizar visualmente todos los elementos de la interfaz en la pantalla **[16].**

**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 **[17].**

**Render Mode:** 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 **[18].**



Ilustración 5.53. Render Mode - Canvas

1. **Adaptabilidad y Escalabilidad:**

Se ha utilizado el componente ‘Canvas Scaler’ para asegurar que nuestra UI se vea consistentemente en diferentes dispositivos y resoluciones **[19].**

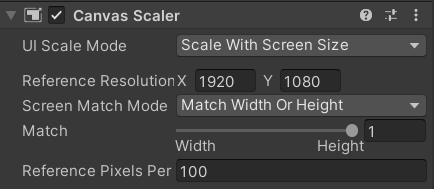


Ilustración 5.54. Canvas Scaler

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 **[20].**

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 5.55. Anchors Presets

Para la **Creación de la UI,** primero 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 5.56. Vista Previa de la UI

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 5.57. Vista previa monedas - UI

**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 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 5.58. Función Actualizar Tiempo en el HUD

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 5.59. Función para convertir cada segundo restante en puntos

## Enriquecimiento y Expansión de Juego

### Construcción de Niveles

Para gestionar la construcción de niveles en el juego, se ha utilizado la herramienta Tilemap de Unity, que proporciona un método eficiente y flexible para diseñar entornos de juego de manera rápida y a escala. La principal ventaja de Tilemap es su capacidad para trabajar con cuadrículas, donde cada celda puede ser llenada con "tiles" que son sprites individuales. Esto permite una colocación precisa y fácil de texturas o elementos de juego **[21].**

El proceso comienza con la creación de un objeto "Grid" que sirve como contenedor para uno o más objetos "Tilemap". Este objeto grid gestiona propiedades comunes como el tamaño de la celda y la separación entre ellas, proporcionando una base uniforme para el diseño del nivel.

Para cada Tilemap, se utilizan "paletas" que son colecciones de tiles. Estas paletas permiten seleccionar y colocar tiles dentro de Tilemap con facilidad.

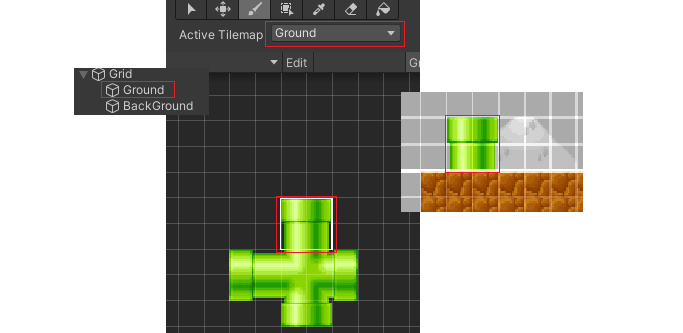


Ilustración 5.60. Utilización de las paletas del Tilemap

Como en el diseño de niveles es importante configurar adecuadamente los layers y colliders, Unity permite añadir automáticamente colliders a los tiles, facilitando la configuración de interacciones físicas sin necesidad de ajustar manualmente cada elemento. Para optimizar el rendimiento y evitar problemas en las interacciones de los colliders, he utilizado el componente Composite Collider, que combina múltiples colliders en uno solo, reduciendo la carga computacional y simplificando el manejo de las colisiones.

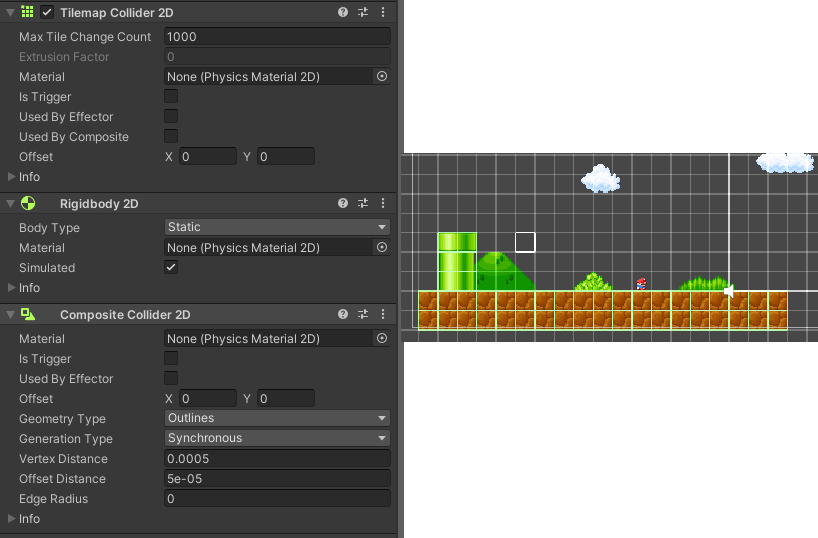


Ilustración 5.61. Utilización del Composite Collider 2D

En todos los niveles, he utilizado una imagen de referencia del juego original, permitiéndome dibujar los elementos del nivel directamente sobre una plantilla visual en Tilemap. Este método asegura que todos los componentes del nivel son colocados de manera adecuado y fiel al original, manteniendo la autenticidad del estilo y la jugabilidad.

**PRIMER NIVEL**

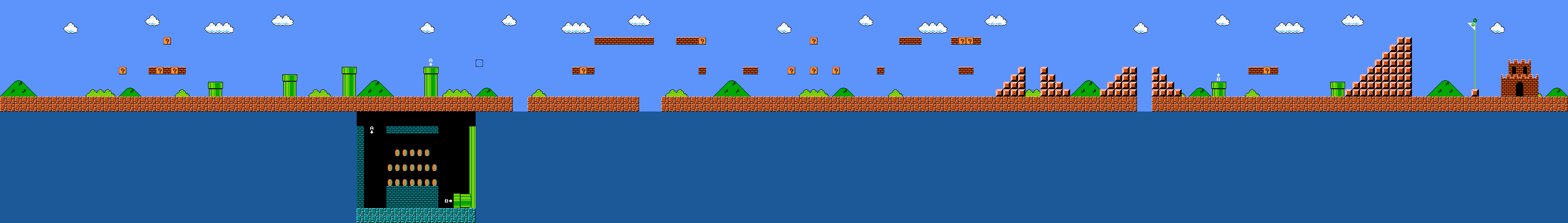


Ilustración 5.62. Utilización de las paletas del Tilemap

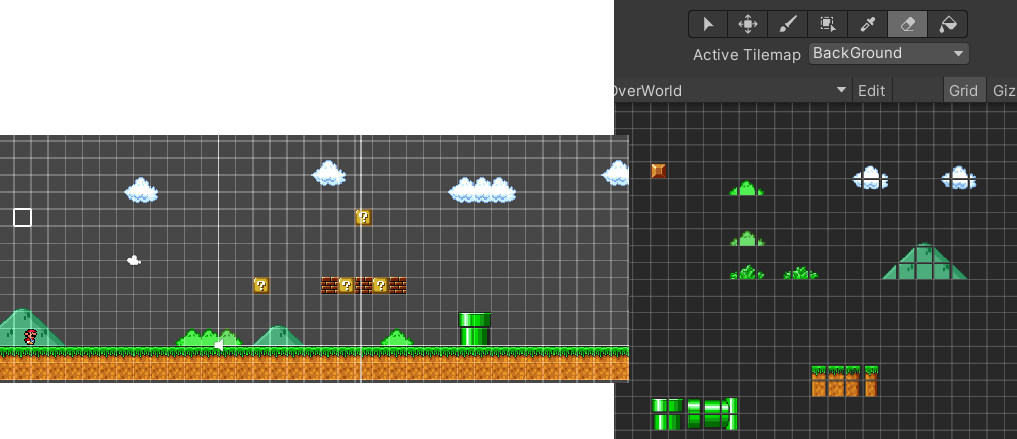


Ilustración 5.63. Utilización de las paletas del Tilemap

Además, en este primer nivel he ajustado lo siguiente:

* **Bloques invisibles**

He implementado bloques invisibles que se activan cuando Mario los golpea desde abajo. Estos bloques, inicialmente configurados para ser transparentes, mediante el script ‘Bloque’ cambia su propiedad de visibilidad y colisión en respuesta a la interacción del jugador.

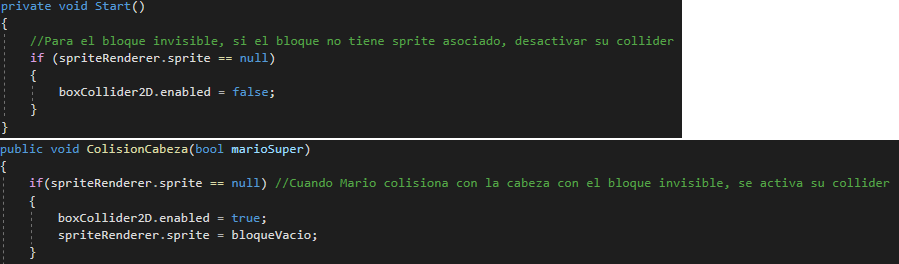


Ilustración 5.64. Utilización de las paletas del Tilemap

* **Interacción de Bloques de ítems**

Si Mario está en el estado ‘Super’ y golpea un bloque que liberaría una ‘Seta Mágica’, el bloque en cambio ofrecerá una flor de fuego.

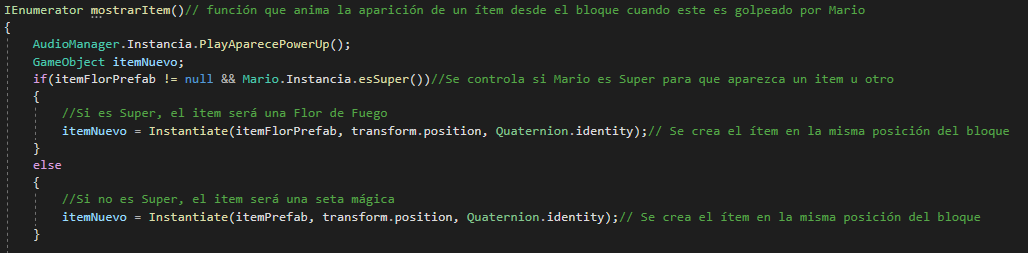


Ilustración 5.65. Utilización de las paletas del Tilemap

* **Uso del GameObject ‘Brush’ para colocación de Objetos**

Para la colocación de objetos como monedas y bloques dentro del nivel, se ha utilizado el GameObject ‘Brush’. Esta herramienta permite ‘pintar’ objetos directamente en el Tilemap, lo cual facilita la distribución uniforme y precisa de los elementos a lo largo de nivel.

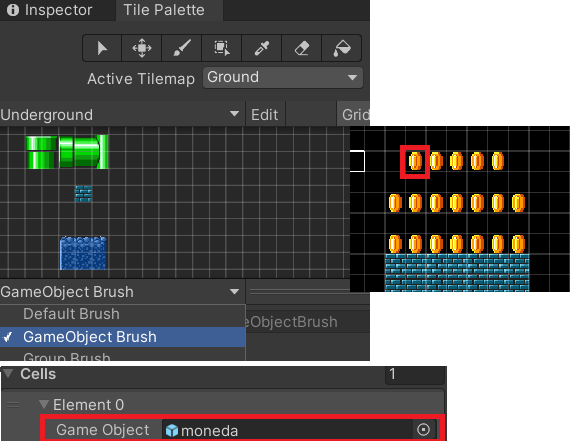


Ilustración 5.66. Utilización de las paletas del Tilemap

Al seleccionar el tipo de objeto, se puede fácilmente colocar estos elementos en la escena movimiento el cursor y haciendo clic en las ubicaciones deseadas dentro de la cuadrícula de Tilemap

* **Conexión de las distintas zonas del nivel mediante Tuberías**

Las tuberías son elementos que sirven como conexiones entre diferentes zonas del nivel. Para implementar la conexión entre las diferentes zonas del nivel he seguido el siguiente proceso:

1. **Configuración de Colliders**

Primero he creado dos prefabs:

**ConexionZonas:** Este prefab incluye un collider que detecta la presencia de Mario y activa el proceso de transición. El script ‘ConexionZonas’ está asociado con este prefab para gestionar la detección y la acción de transición.

**Zona:** Este prefab representa la nueva zona a la que Mario se moverá. Incluye el punto de entrada, que especifíca dónde Mario aparecerá en la nueva zona.



Ilustración 5.67. Utilización de las paletas del Tilemap

Además, en el prefab ‘ConexionZonas’ se configura un collider que funciona como trigger, el cual se activa cuando Mario colisiona, y el script ‘ConexionZonas’ prepara y ejecuta la transición.

1. **Script ‘ConexiónZonas’ para Detección y Transición**

Este script detecta cuando Mario está en la posición correcta para la transición, es decir, esta sobre la tubería y presionar la tecla adecuada.

Cuando se cumplen las condiciones, el script inicia la corrutina que maneja la transición de Mario a la nueva zona. Esto implica deshabilitar temporalmente el control del jugador, ajustar la cámara y mover a Mario hacia el punto de entrada a la nueva zona.

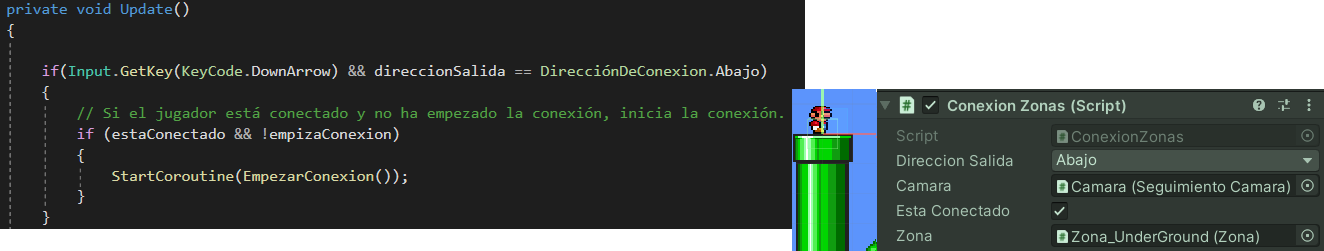


Ilustración 5.68. Utilización de las paletas del Tilemap

1. **Script ‘Zona’ para gestionar la llegada**

El script ‘Zona’ está configurado para recibir a Mario. Una vez que Mario es trasladado a la nueva zona, este script lo coloca en el punto de entrada y reactiva cualquier control o ajuste necesario, es decir, la cámara y los controles del jugador.

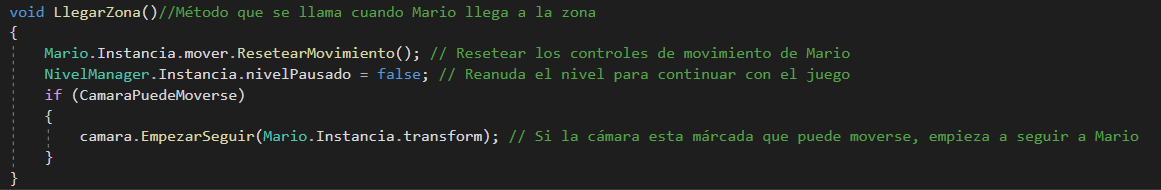


Ilustración 5.69. Utilización de las paletas del Tilemap

**TRANSICIÓN DE NIVELES**

Para implementar las transiciones de niveles, el proceso lo he estructurado en los siguientes pasos:

1. **Castillo**

Cuando Mario entra en colisión con el castillo, el script lo detecta y desplaza a Mario a una posición fuera de la pantalla para simular que ha entrado al castillo, y además mira si se ha completado la transición de segundos a puntos para pasar de nivel.



Ilustración 5.70. Utilización de las paletas del Tilemap

1. **Manejo de la Transición con ‘GameManager’**

Si se cumplen las condiciones adecuadas (Mario ha colisionado con el castillo y ha finalizado la transición de segundos a puntos), el ‘GameManager’ gestiona la transición al próximo nivel.

El ‘GameManager’ carga la escena de transición, esta escena se utiliza para mostrar información sobre el progreso del juego, número de vidas restantes, mundo y nivel a los que Mario se dirige.

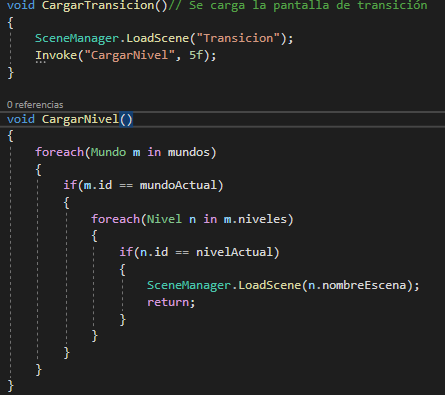


Ilustración 5.71. Utilización de las paletas del Tilemap

1. **Escena de Transición**

En la escena de transición, el script ‘Transición’ activa o desactiva paneles basado en el estado de juego. Si es ‘Game Over’, se muestra el panel correspondiente, y si el juego continúa, se muestra el panel con información del del próximo nivel.



Ilustración 5.72. Utilización de las paletas del Tilemap

1. **Carga del Nuevo Nivel**

Una vez que pasa 5 segundos en la escena de Transición, el ‘GameManager’ configura las variables para el nuevo mundo y nivel, basándose en la información almacenada en las estructuras de ‘Mundo’ y ‘Nivel’.

Finalmente, se carga el nuevo nivel utilizando ‘SceneManager’ para traer la escena correspondiente al mundo y nivel seleccionados.

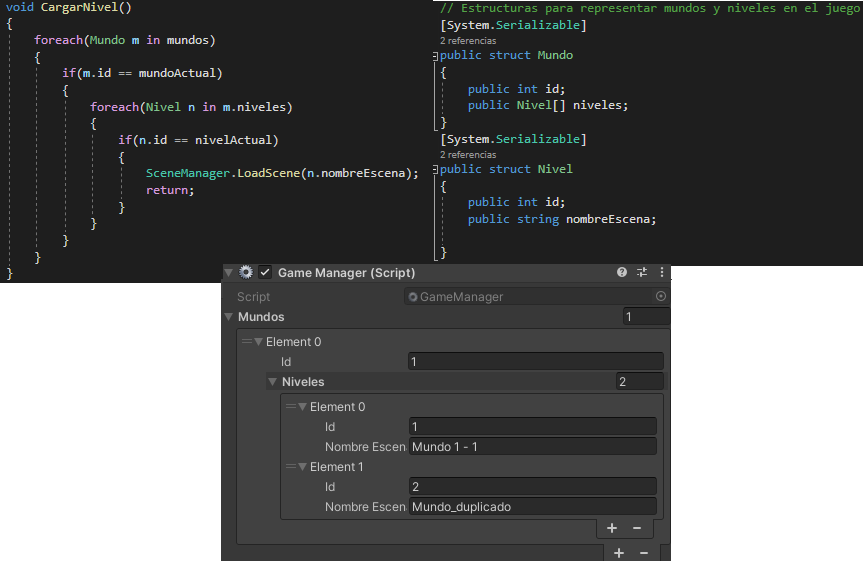


Ilustración 5.73. Utilización de las paletas del Tilemap

**SEGUNDO NIVEL**

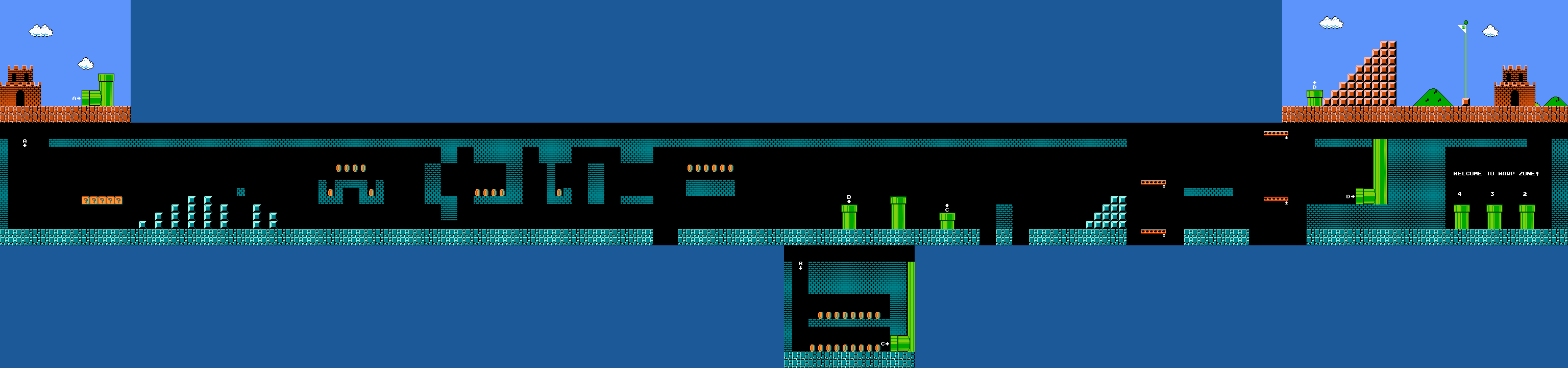


Ilustración 5.74. Utilización de las paletas del Tilemap

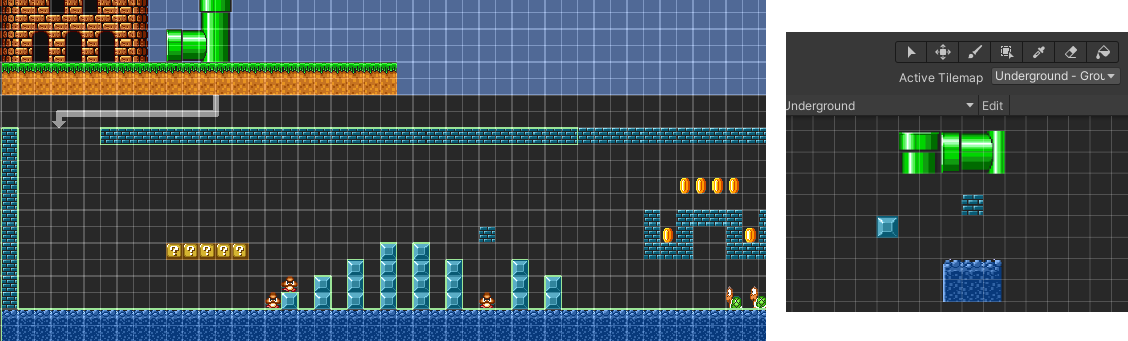


Ilustración 5.75. Utilización de las paletas del Tilemap

Además, en este segundo nivel he ajustado lo siguiente:

* **Plataformas**

Para implementar y gestionar las plataformas móviles, he seguido los siguientes pasos:

1. **Configuración de la Plataforma:** Como se ha hecho con todos los objetos, se ha añadido el Sprite de plataforma a la escena y se ha agregado un ‘BoxCollider2D’ para que Mario pueda interactuar físicamente con ella.
2. **Implementación del Comportamiento de Plataforma**

He utilizado el ‘PlatformEffector2D’, que permite crear un comportamiento de plataforma donde Mario puede saltar desde abajo a la plataforma.

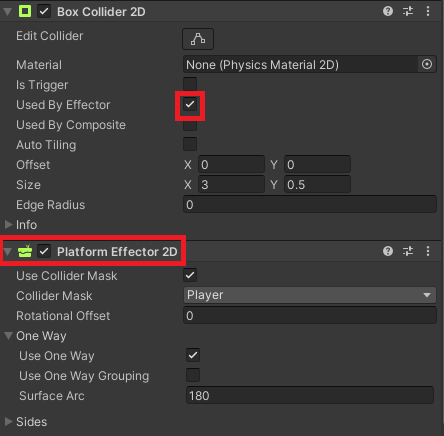


Ilustración 5.76. Utilización de las paletas del Tilemap

1. **Scripting del Movimiento:** He creado un script que maneja el movimiento de la plataforma entre dos puntos definidos (punto de inicio y punto final). Este script permite configurar la velocidad del movimiento, la dirección (horizontal, vertical o libre) y su el movimiento es continuo o de solo ida y vuelta.

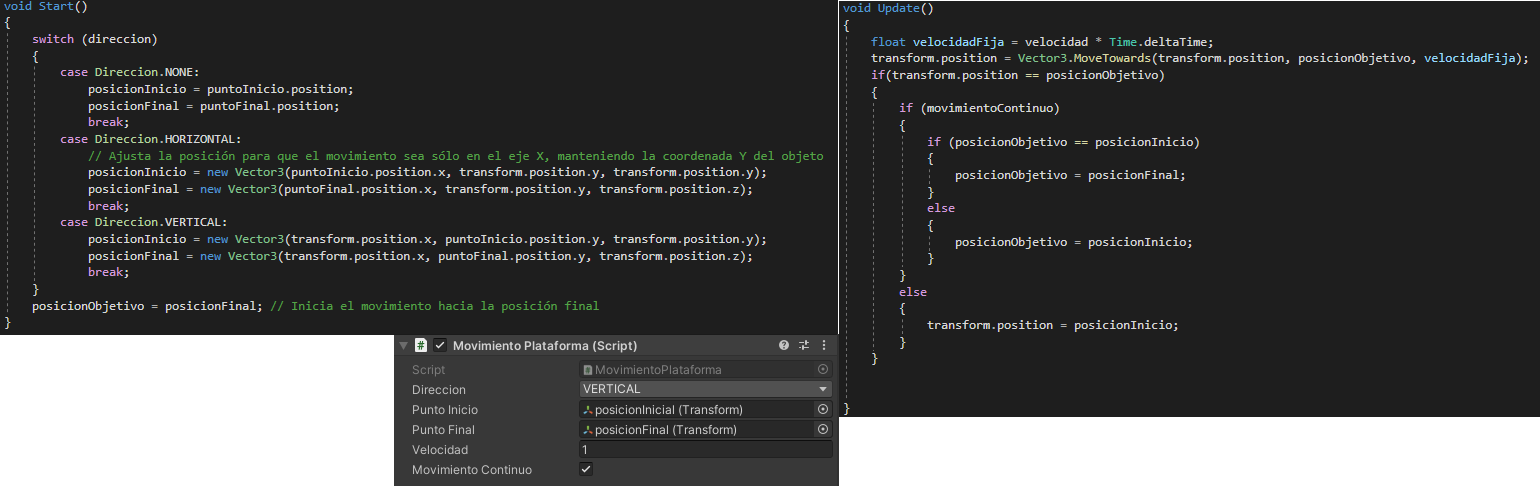


Ilustración 5.77. Utilización de las paletas del Tilemap

1. **Integración con Mario**

Para la correcta interacción con Mario, se asegura que cuando Mario salta sobre la plataforma en movimiento, él se mueve junto con ella. Esto se consigue haciendo que Mario sea hijo de la plataforma mientras está en contacto con la misma.

Cuando Mario deja la plataforma, se revierte el parentesco para que Mario vuelva a ser independiente del movimiento de la plataforma.

* **Mas Enemigos**

Además, para este nivel y niveles posteriores he añadido dos enemigos más, Koopa Rojo y Koopa con Alas

**Koopa Rojo:**

El Koopa Rojo se distingue de otros Koopas por su habilidad para no caer de las plataformas automáticamente, lo que implica una lógica especial de patrullaje:

* **Detección de Bordes:** Se implementó un sistema de detección de bordes para el Koopa Rojo usando raycast. Este sistema permite a Koopa cambiar de dirección al alcanzar el final de una plataforma, evitando que se caiga de ella.

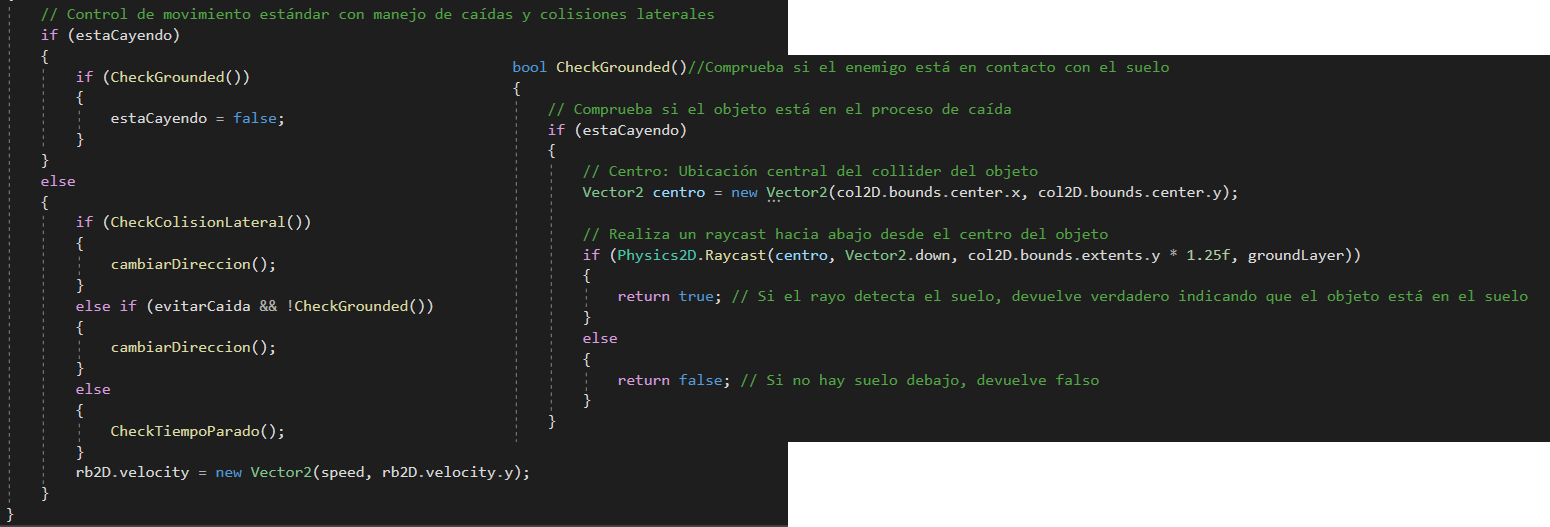


Ilustración 5.78. Utilización de las paletas del Tilemap

* **Movimiento Automatizado**: Utilizando el script ‘AutoMovimiento’, el Koopa Rojo sigue un patrón de movimiento horizontal hasta que detecta un borde o una colisión lateral, momento en el cual invierte su dirección.

**Koopa Con Alas:**

El Koopa con alas añade una capa de complejidad con su capacidad de volar y perder las alas:

* **Capacidad de Vuelo:** Al inicio, el Koopa con Alas es capaz de volar, lo que se gestiona a través de animaciones específicas y cambios en la lógica de movimiento para simular el vuelo.

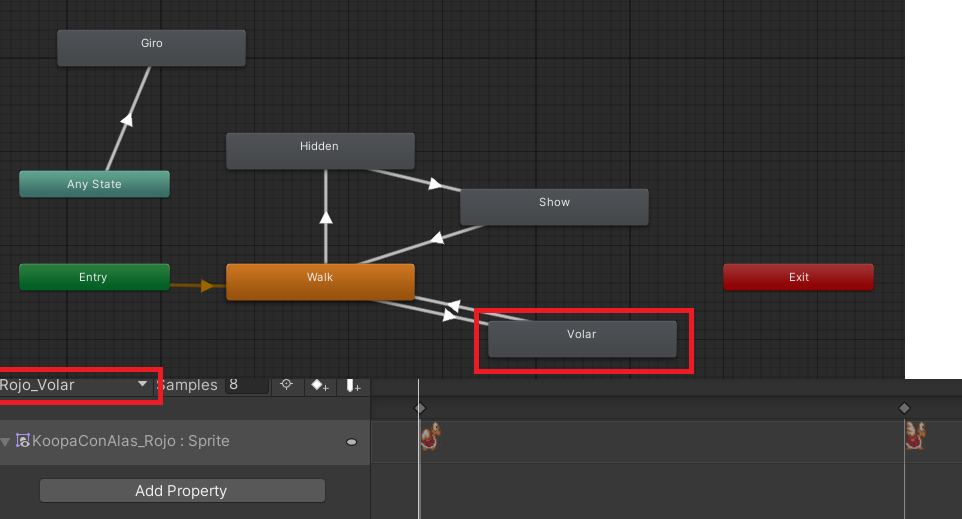


Ilustración 5.79. Utilización de las paletas del Tilemap

* **Pérdida de Alas:** Al ser pisoteado por primera vez, el Koopa con Alas pierde sus alas. Esto se manejó mediante una transición en la que el Koopa pasa de un estado de vuelo a un movimiento normal, modificando dinámicamente sus propiedades físicas. Tras perder las alas, el Koopa con Alas adopta comportamientos de un Koopa normal.

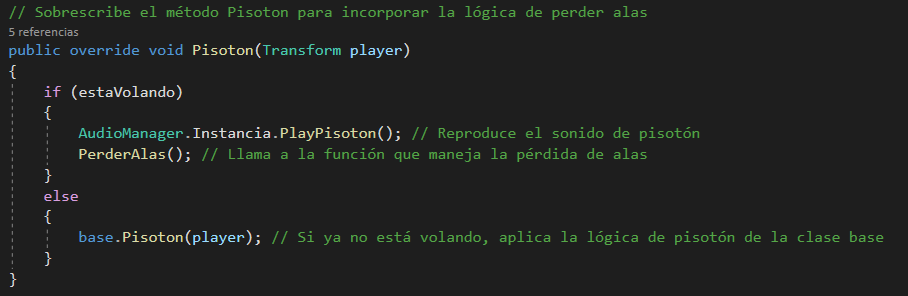


Ilustración 5.80. Utilización de las paletas del Tilemap

**TERCER NIVEL**

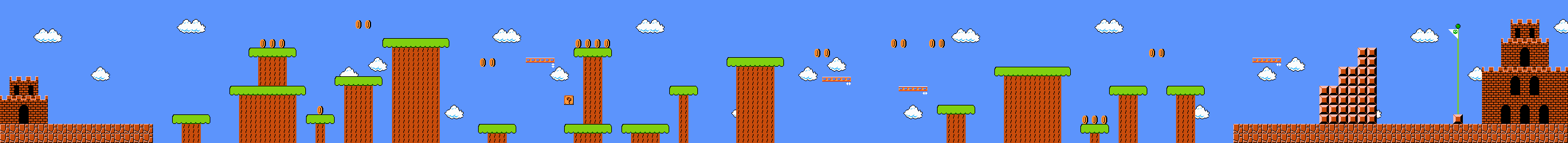


Ilustración 5.81. Utilización de las paletas del Tilemap

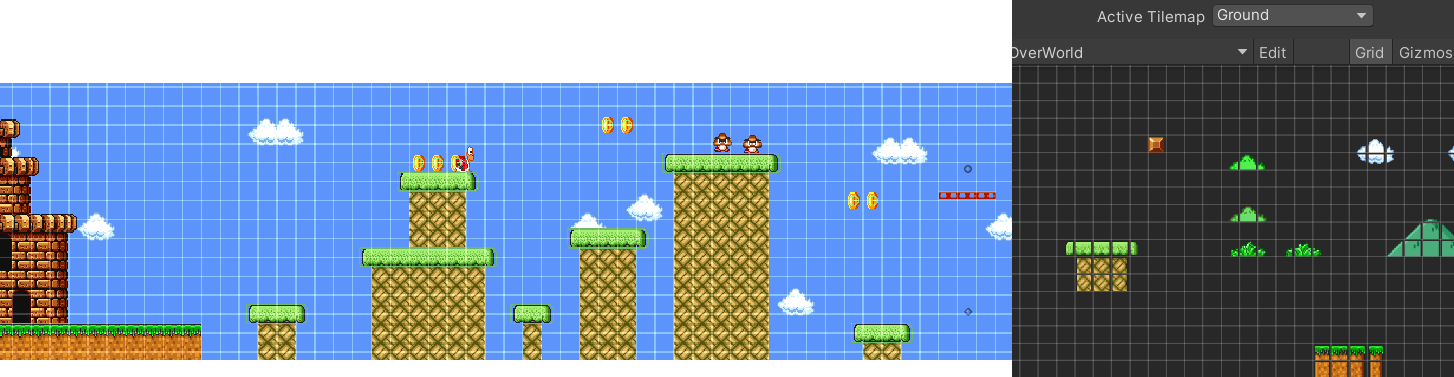


Ilustración 5.82. Utilización de las paletas del Tilemap

**CUARTO NIVEL**

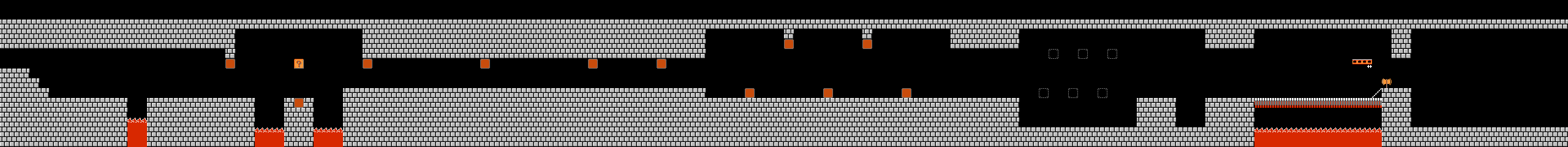


Ilustración 5.83. Utilización de las paletas del Tilemap

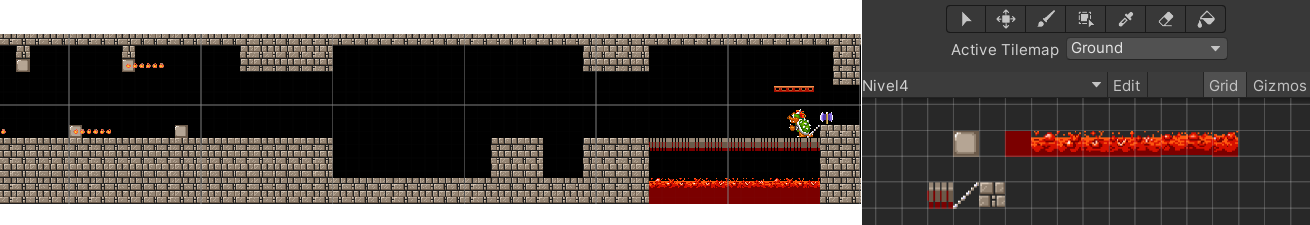


Ilustración 5.84. Utilización de las paletas del Tilemap

Además, en este cuarto nivel he añadido lo siguiente:

* **Lava**

Se ha creado una nueva capa llamada ‘Lava’ para manejar específicamente las interacciones con este elemento.

He programado la lava para que cause daño o la muerte a Mario al contacto, dependiendo de su estado. Esto lo he implementado modificando el script de colisiones en el método OnCollisionEnter2D.

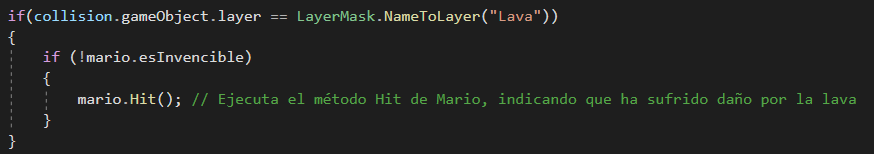


Ilustración 5.85. Utilización de las paletas del Tilemap

* **Barra de Fuego**

Las barras de Fuego se han diseñado como objetos rotativos que son colocados a lo largo de los niveles.

Se ha creado un script específico (BarraFuego) para controlar la rotación continua de las barras de fuego, usando la propiedad ‘velocidadRotacion’ para ajustar su rapidez.

Las barras de Fuego se configuraron para interactuar con Mario, utilizando colliders y ajustando el script de colisiones para gestionar los efectos de contacto.

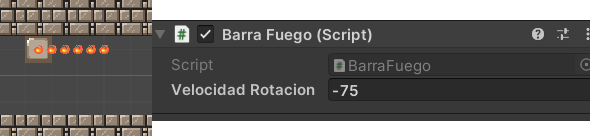


Ilustración 5.86. Utilización de las paletas del Tilemap

**PUENTE Y FINAL:**

* **Creación del puente:** Se creó un objeto vacio ‘puente’ que sirve como contenedor para todas las piezas del puente.
* **Interacción con el Hacha:** Se creó un Sprite para el hacha al final del puente. Al detectar la colisión con Mario, mediante el script ‘HachaFinal’, se inicia una corrutina que destruye las piezas del puente una a una con un pequeño retardo entre cada acción, para dar tiempo a visualizar cómo se desmorona el puente.

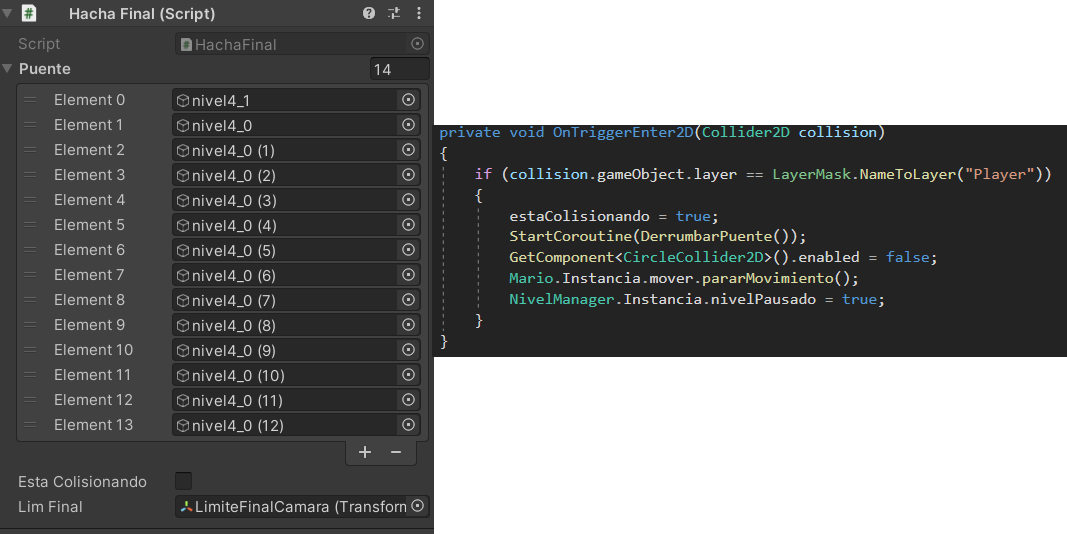


Ilustración 5.87. AspectRatioCamara.cs

* **Finalización del Nivel:** Tras la destrucción del puente, se gestionó la transición hacia el encuentro final con Toad mediante la activación de un canvas que contiene los mensajes finales del nivel.

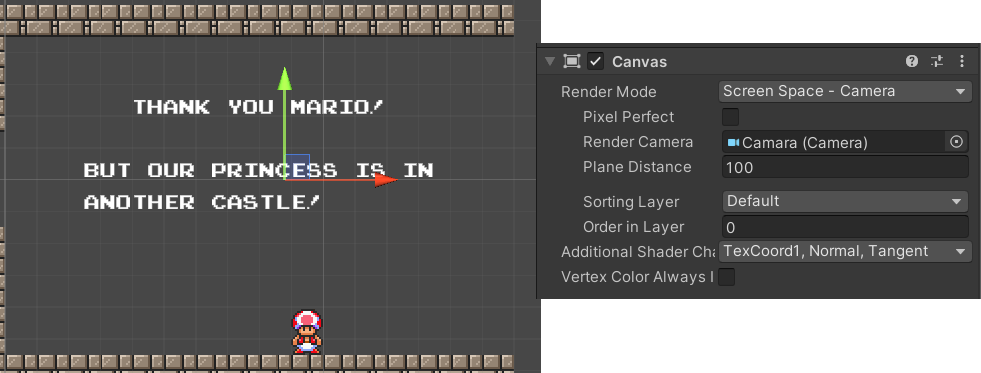


Ilustración 5.88. AspectRatioCamara.cs

Se añadió un collider alrededor de Toad para detectar cuando Mario lo alcanza después de que el puente se destruye. Al tocar a Toad, se detiene a Mario y se activa el canvas con los textos, los cuales se revelan de forma secuencial.

El canvas utilizado para los mensajes finales fue configurado para operar en el espacio de la cámara, asegurando que los textos se mostraran correctamente en relación con la vista del jugador, sin importar la resolución o la configuración de la pantalla.

**BOWSER:**

Bowser es implementado como una clase derivada de Enemigo, lo que le permite heredar y extender funcionalidades comunes a todos los enemigos, como la gestión de animaciones y movimientos básicos. Sin embargo, he añadido características únicas para adaptarlo a los desafíos específicos que presenta este jefe final:

* **Control de Movimientos y Ataques:**

**Movimiento**: Bowser tiene la capacidad de moverse hacia adelante y hacia atrás en relación a la posición de Mario, lo cual se controla mediante una variable ‘puedeMoverse’ que se activa cuando Mario entra en un rango mínimo definido. Esto hace que Bowser inicie su aproximación sólo cuando Mario está suficientemente cerca, aumentando la tensión y la dificultad de la confrontación.

**Salto**: Se implementa un temporizador que gestiona los intervalos de salto, añadiendo una capa de imprevisibilidad al comportamiento de Bowser. Los saltos se utilizan tanto para esquivar ataques como para intentar aplastar a Mario si está debajo.

**Disparo de Fuego:** Similar a los saltos, un temporizador controla los intervalos entre disparos. Bowser dispara bolas de fuego que se dirigen hacia Mario.

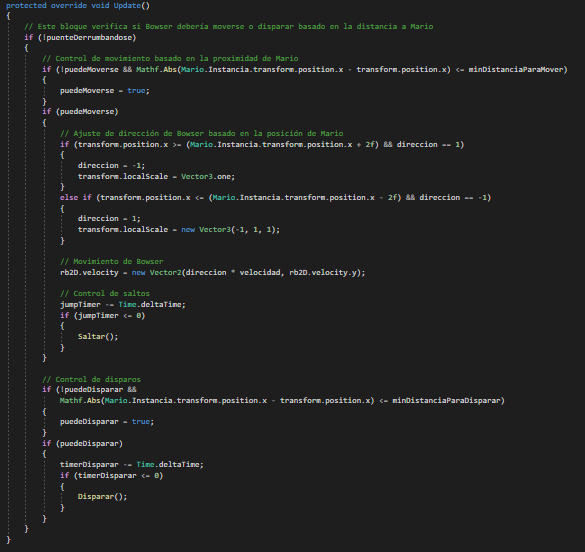


Ilustración 5.89. AspectRatioCamara.cs

* **Gestión de Salud y Respuestas a Ataques:**

Bowser cuenta con un sistema de salud que decrece cada vez que es golpeado por las bolas de fuego de Mario. Una vez que la salud llega a cero, se activa una secuencia de muerte que implica animaciones específicas para mostrar su derrota, lo que incluye cambios en la interacción con el entorno al convertirse en una parte no colisionable.

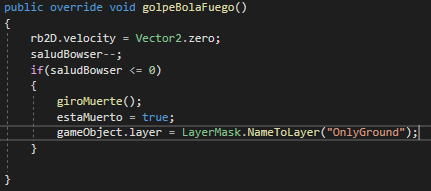


Ilustración 5.90. AspectRatioCamara.cs

* **Interacción con el Entorno:**

**Derrumbe del Puente:** Un elemento crucial en la batalla contra Bowser es el puente que se colapsa cuando Mario toca el hacha al final de la plataforma. Esto inicia una secuencia donde las piezas del puente se destruyen progresivamente, con Bowser cayendo al vacío si aún está vivo.

**GUARDAR**

Para implementar y gestionar el sistema de guardado en el videojuego, se desarrolló una estrategia eficiente que asegura la persistencia de los datos esenciales del jugador entre sesiones. Este sistema se basa en el uso de la clase ‘**PlayerPrefs’** de Unity para almacenar y recuperar la información del progreso del jugador.

**Implementación del Sistema de Guardado**

* **Gestión de Puntuaciones (ScoreManager)**

Al final de cada partida, se verifica si la puntuación actual supera la máxima puntuación previamente guardada. Si es así, se actualiza el valor y se guarda en ‘PlayerPrefs’ bajo la clave "Puntos". Esto asegura que la puntuación máxima siempre esté actualizada y disponible para sesiones futuras.

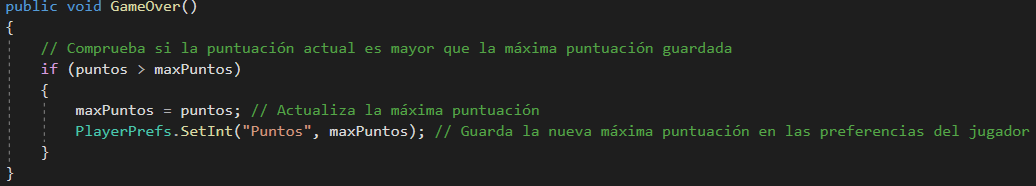


Ilustración 5.91. AspectRatioCamara.cs

* **Gestión del Progreso del Juego (GameManager)**

Cuando el jugador pierde o el juego termina, se guardar el mundo y nivel actual en ‘PlayerPrefs’. Esto no solo incluye puntuaciones, sino también el progreso exacto en términos de ubicación en el juego.

Además, al iniciar el juego, se recuperan los valores de mundo y nivel de ‘PlayerPrefs’. Si no hay valores previos, se establecen valores predeterminados. Este sistema permite que el jugador retome el juego exactamente donde lo dejó.

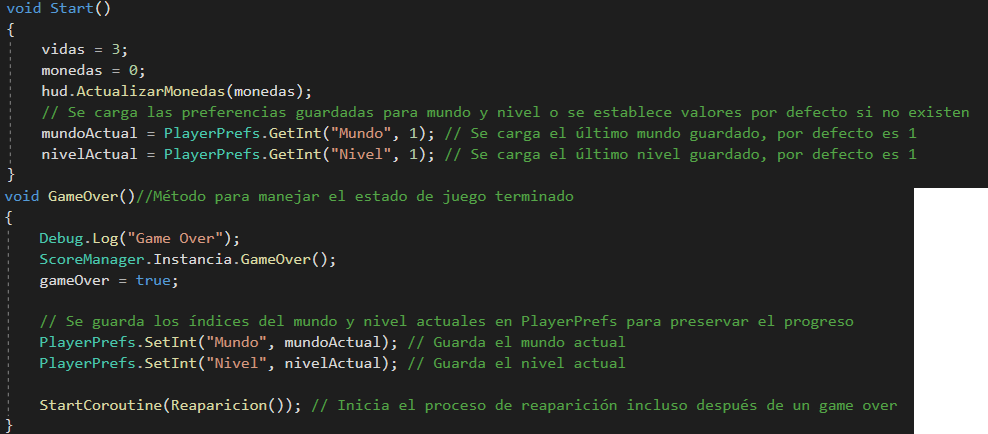


Ilustración 5.92. AspectRatioCamara.cs

**PlayerPrefs** permite que los datos persistan incluso después de cerrar el juego, lo que es esencial para mantener la integridad del sistema de puntuaciones.

**MENU INICIO:**

El menú inicial sirve como puerta de entrada al juego, ofreciendo opciones para comenzar una nueva partida o continuar una existente, además de mostrar el puntaje máximo alcanzado.

**Componentes del Menú:**

* **Interfaz de Usuario:**



Ilustración 5.93. AspectRatioCamara.cs

**Top Score:** Utilizando PlayerPrefs, el sistema recupera y muestra el puntaje más alto alcanzado, presentado mediante TextMeshProUGUI. Esto no solo informa al jugador de su mejor rendimiento anterior, sino que también añade un elemento competitivo al menú.

**Botones de Interacción**: El menú incluye dos botones principales gestionados a través de EventSystem de Unity:

* **Nuevo Juego:** Al seleccionar esta opción, se reinicia el progreso del juego, estableciendo el nivel y el mundo a su estado inicial y comenzando una nueva partida.
* **Continuar Juego:** Esta opción permite al jugador retomar su última partida guardada, cargando los niveles y puntos desde donde lo dejó.
* **Gestión de Estados:**

**Script ‘MenuInicial’**: Encargado de activar las configuraciones iniciales como mostrar los puntos y posicionar a Mario. Además, define el comportamiento de los botones del menú, asignando las acciones correspondientes a cada uno para iniciar o continuar el juego.

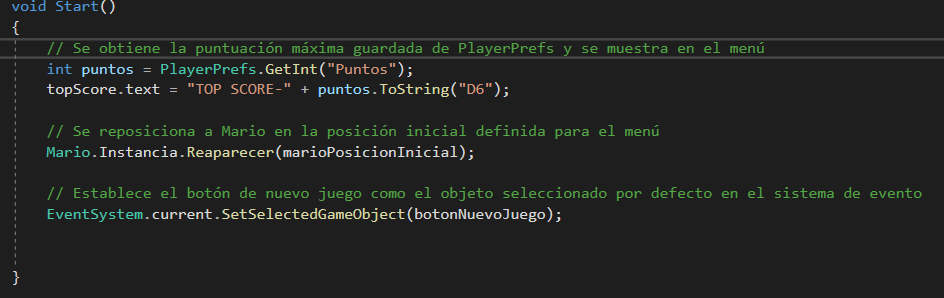


Ilustración 5.94. AspectRatioCamara.cs

**Script ‘BotonMenu’:** Maneja la visualización del icono de un cursor al seleccionar cada botón, mejorando la interacción visual del usuario con el menú mediante feedback visual directo cuando un botón es seleccionado o deseleccionado.

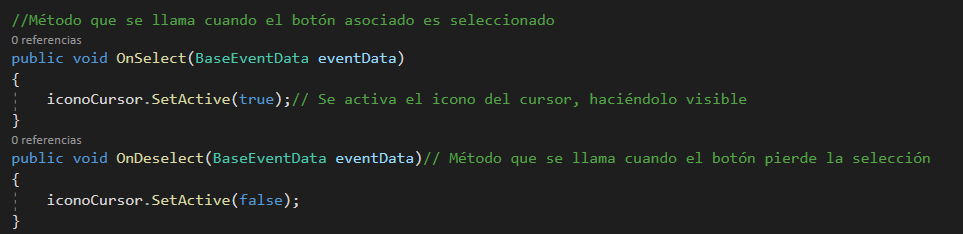


Ilustración 5.95. AspectRatioCamara.cs

### Configuración de la Cámara y Ajustes Visuales

En el desarrollo del videojuego, la gestión, implementación y configuración de la cámara es una parte crucial para mantener la jugabilidad y la estética fiel al juego original, mientras se adapta a las nuevas resoluciones.

A continuación, detallo los pasos y consideraciones específicas sobre la configuración de la cámara:

1. **Configuración de la Proporción de Aspecto de la Cámara**

Para mantener la proporción de aspecto visual del juego original, he desarrollado el script ‘**AspectRatioCamara’** que se encarga de:

* Calcular la proporción de aspecto del juego original, 16:15.
* Ajustar el tamaño y posición del viewport de la cámara para asegurar que el juego se muestre correctamente en cualquier resolución de pantalla, utilizando una comparación entre la proporción de aspecto objetivo y la actual de la pantalla.
* Configurar el ‘Viewport Rect’ de la cámara para ajustar su anchura y centrarlo, garantizando que la visualización no se estire ni comprima.

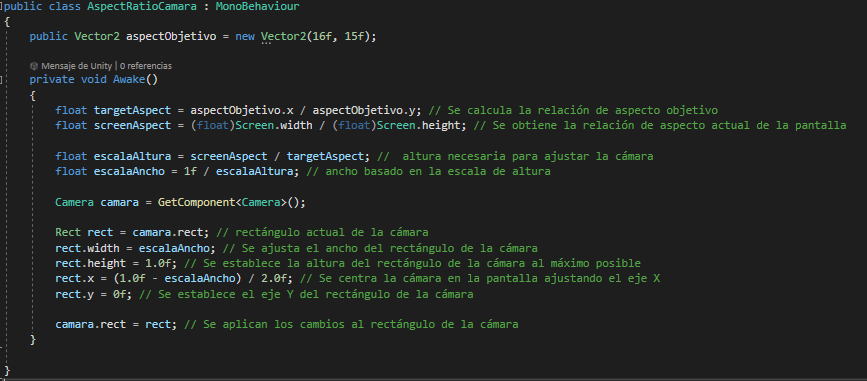


Ilustración 5.96. AspectRatioCamara.cs

1. **Seguimiento Dinámico de Mario**

Para hacer que la cámara siga a Mario he desarrollado el script ‘**SegumientoCamara’** que se encarga de:

* Definir un offset para que la cámara se sitúe ligeramente delante de Mario, mejorando la visibilidad del área hacia la que se dirige.
* Establecer límites horizontales (mínimos y máximos) para el movimiento de la cámara, basados en posiciones específicas en el nivel (usando GameObjects ‘limiteIzq’ y ‘limiteDer’), que previenen que la cámara muestre áreas fuera de los límites del nivel.
* Ajustar dinámicamente la posición de la cámara en el eje X para que nunca retroceda, manteniendo la última posición máxima alcanzada como límite inferior para la cámara. Esto evita que el jugador pueda hacer que Mario vuelva hacia áreas previamente exploradas, lo cual es fiel a la mecánica original del juego.

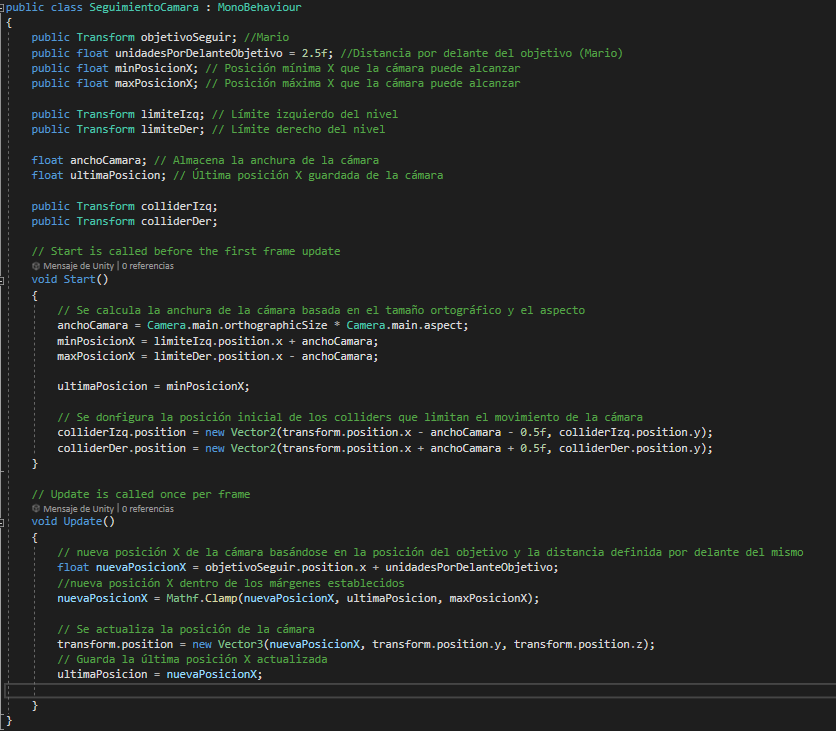


Ilustración 5.97. SegumientoCamara.cs

Además, he colocado colliders en los bordes de la cámara para evitar que Mario salga de la vista, asegurando que siempre esté visible en pantalla.

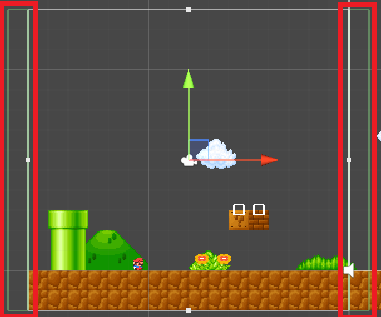


Ilustración 5.98. Collider – Configuración Camara

1. **Adaptación de la Cámara a Diferentes Resoluciones**

Utilizando el ajuste dinámico en el script ‘AspectRatioCamara’ se garantiza que la experiencia visual se mantenga consistente en diferentes dispositivos.

* **FUERA DE CÁMARA**

En el desarrollo del juego, una de las optimizaciones que he implementado ha sido la **gestión de los elementos fuera de cámara** para asegurar el rendimiento óptimo y la experiencia de juego fluida.

Para evitar el consumo innecesario de recursos, se ha implementado un sistema para eliminar automáticamente enemigos, ítems y objetos que salen de la visibilidad de la cámara. Esto lo he llevado a cabo mediante el script ‘**EliminarFueraCamara’**, una vez que un objeto deja de ser visible y supera cierta distancia del centro de la cámara, determinada por ‘*minDistanciaEliminado’*, se elimina del juego. Esto es crucial para mantener el rendimiento.

La **eliminación de enemigos en el juego** se maneja de diferente forma según el tipo de enemigo y su estado específico:

* **Koopa, Koopa Rojo y Koopa con Alas**

Cuando Koopa es golpeado y se refugia dentro de su caparazón, el enfoque de eliminación cambia. Si el Koopa escondido no está en movimiento (estático dentro del caparazón), puede eliminarse si está fuera del alcance visible y además ha superado la *‘minDistanciaEliminado’* establecida.

Sin embargo, si el Koopa está activo y moviéndose dentro del caparazón (lanzado en modo caparazón), puede eliminarse también si esta por delante de la cámara.

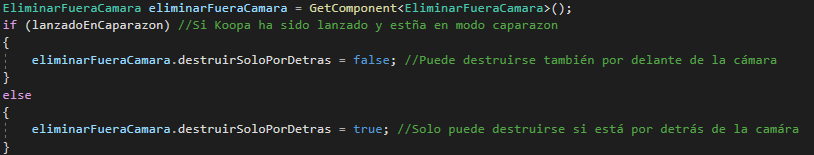


Ilustración 5.99. Eliminación de Koopa de la Escena

* **Goomba**

Los Goombas se eliminan directamente cuando salen de la pantalla por detrás y superan la distancia mínima configurada.

* **Planta Piraña**

La eliminación de las Plantas Piraña se maneja eliminando el objeto padre.

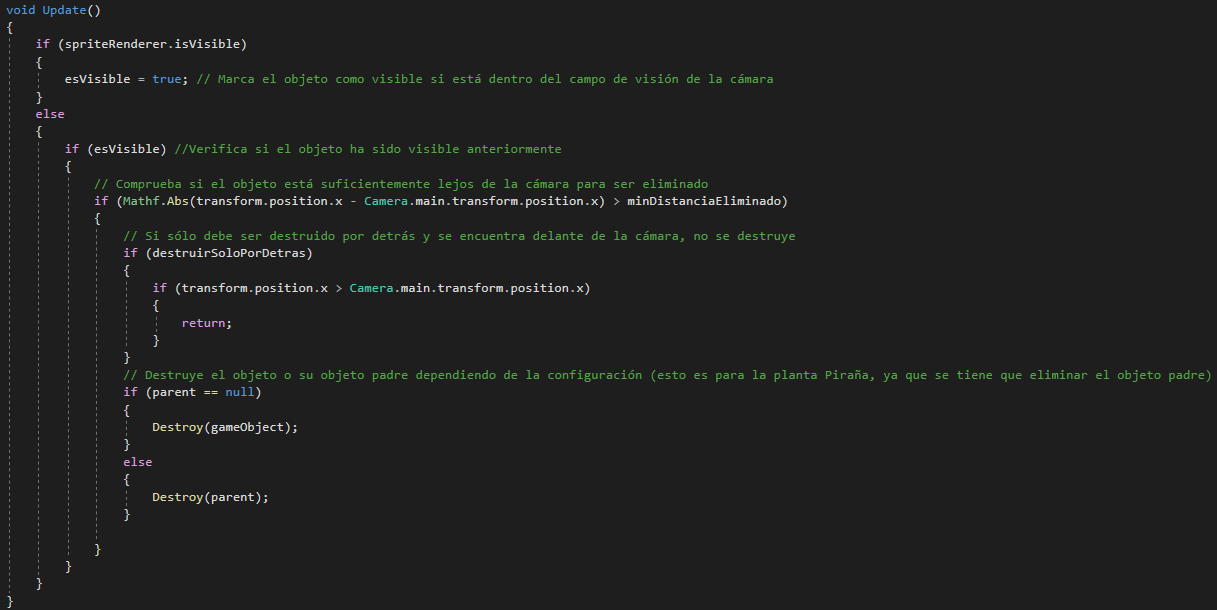


Ilustración 5.100. Eliminación de Enemigos e items de la Escena

Además, los enemigos se activan no solo cuando entran en el campo visual de la cámara sino también en respuesta a la proximidad de otros enemigos activos.

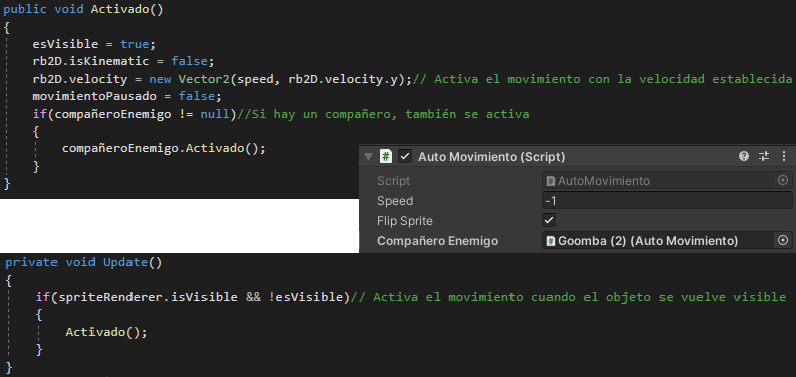


Ilustración 5.101. Activación de Enemigos en la Escena

**VIDAS**

La implementación de vidas de Mario la he realizado de la siguiente manera:

**Sistemas de Vidas:**

* **Inicialización de Vidas**: Al comenzar una partida, el ‘GameManager’ inicializa las vidas de Mario a tres.
* **Pérdidas de Vidas:**

Las vidas se pierden cuando Mario cae en zonas de muerte o cuando el tiempo se agota. Estas zonas están gestionadas en el juego mediante un collider, que detecta cuando Mario entra en contacto con ella y llama a ‘GameManager’ para manejar la pérdida de vida.

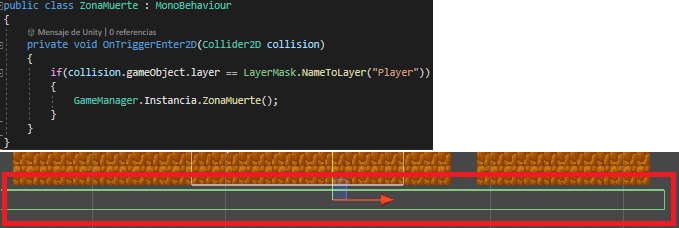


Ilustración 5.102. Zona Muerte

* **Game Over**: Si las vidas llegan a cero, el juego termina.

**Recuperación de Vidas:**

* **Por Recolección de Monedas:** Cada vez que Mario alcanza un total de 100 monedas, no solo se resetea el contador de monedas, sino que también se otorga una vida extra.
* **Por ítems:** Cuando Mario recoge el ítem *‘Seta de Vida’*, le otorga una vida adicional directamente.

**RESPAWN**

El proceso de Respawn comienza cuando Mario muere, ya sea por caer al vacío (Zona de muerte) o al agotarse el tiempo. En estos eventos, el script *‘GameManager’* es el encargado de iniciar el proceso de respawn.

**Detección de la muerte de Mario y activación del respawn**

* **GameManager:** Gestiona la muerte de Mario controlando las vidas restantes y ejecutando la corrutina de respawn cuando es necesario.

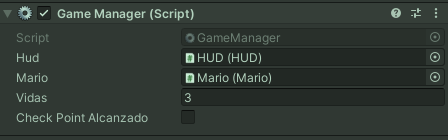


Ilustración 5.103. GameManager

* **CheckPointAlcanzado:** Esta variable es utilizada para determinar si Mario debe respawnear en el checkpoint o en el inicio del nivel.

**Implementación de CheckPoints**

* Se ha colocado un GameObject con un collider configurado como Trigger en los puntos clave del nivel. El script asociado ‘CheckPoint’ detecta la interacción de Mario con ese punto, como se ha hecho con las Zonas de Muerte.

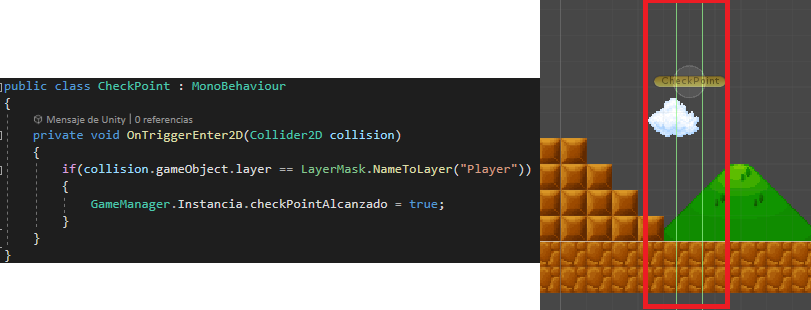


Ilustración 5.104. CheckPoint

Cada checkpoint tiene asignado un ‘id’ y una posición inicial del jugador que determina dónde debe reaparecer Mario. Este sistema es manejado además por el script ‘GameManager’ que registra el último checkpoint activado y maneja la lógica de respawn desde ese punto.

**Preservación de estados entre la carga de las escenas**

* **DontDestroyOnLoad:** Se aplica a los objetos ‘*GameManager’*, ‘*AudioManager’*, ‘*ScoreManager’* y ‘*Mario’* para mantener su estado después de recargar la escena. Esto asegura que, a pesar de la recarga de la escena para el respawn, la información como las puntuaciones y la configuración de audio permanezcan intactas.

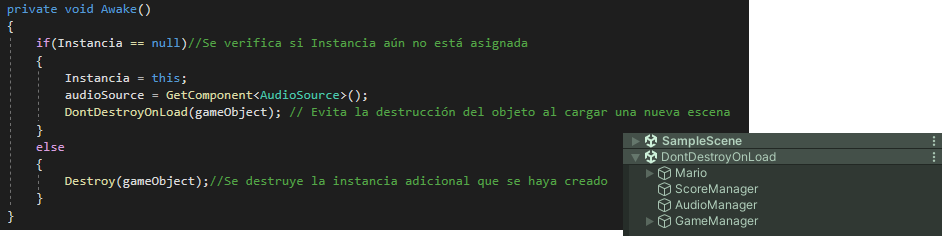


Ilustración 5.105. Uso de ‘DontDestoryOnLoad()’

**Recarga de la escena**

* En lugar de reposicionar manualmente a todos los objetos y enemigos del nivel, la escena completa se recarga utilizando ‘SceneManager.LoadScene()’.

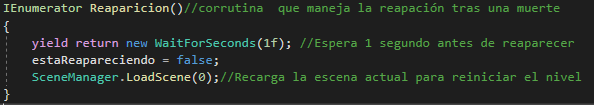


Ilustración 5.106. Uso de ‘SceneManager.LoadScene()’

**Ajuste del sistema de cámara y otros elementos después del respawn**

* Después de un respawn, para que la cámara se inicialice en la posición nueva de Mario he implementado la función ‘EmpezarSeguir’ en ‘NivelManager’ para reajustar la posición de la cámara según la nueva posición de Mario.

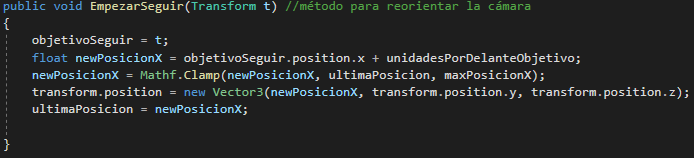


Ilustración 5.107. Modo Invencible

# 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

[1] Unity Technologies, 2024: <https://unity.com/es>

[2] Microsoft, 2024. Visual Studio: <https://visualstudio.microsoft.com/>

[3] Microsoft, 2024. Paint: <https://www.microsoft.com/es-es/windows/paint>

[4] GIMP, 2024. GIMP: <https://www.gimp.org/>

[5] Schwaber, K. y Sutherland, J., 2020. The Scrum Guide: <https://scrumguides.org/scrum-guide.html>

[6] Unity Technologies, 2024. Unity - Manual: Sprites: <https://docs.unity3d.com/Manual/Sprites.html>

[7] Unity Technologies, 2024. Unity - Manual: Physics 2D Reference: <https://docs.unity3d.com/Manual/Physics2DReference.html>

[8] Unity Technologies, 2024. Unity - Manual: Physics 2D Raycaster: [Unity - Manual: Physics 2D Raycaster (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/script-Physics2DRaycaster.html)

[9] Unity Technologies, 2024. Unity - Manual: Animation system overview: [Unity - Manual: Animation system overview (unity3d.com)](https://docs.unity3d.com/2021.3/Documentation/Manual/AnimationOverview.html)

[10] Unity Technologies, 2024. Unity - Manual: Animation Clips: [Unity - Manual: Animation Clips (unity3d.com)](https://docs.unity3d.com/2021.3/Documentation/Manual/AnimationClips.html)

[11] Unity Technologies, 2024. Unity - Manual: Animator Controller: [Unity - Manual: Animator Controller (unity3d.com)](https://docs.unity3d.com/2021.3/Documentation/Manual/class-AnimatorController.html)

[12] Unity Technologies, 2024. Unity - Manual: Prefabs: [Unity - Manual: Prefabs (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/Prefabs.html)

[13] Patrón de Diseño Singleton: <https://aspnetcoremaster.com/csharp/patron-de-dise%C3%B1o-singleton-csharp.html>

[14] Unity Technologies, 2024. Unity - Manual: Audio Clip: [Unity - Manual: Audio Clip (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/class-AudioClip.html)

[15] Unity Technologies, 2024. Unity - Manual: Audio Source: [Unity - Manual: Audio Source (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/class-AudioSource.html)

[16] Unity Technologies, 2024. Unity - Manual: Canvas: [Unity - Manual: Canvas (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/UICanvas.html)

[17] Unity Technologies, 2024. Unity - Manual: Event System: [Unity - Manual: Event System (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/EventSystem.html)

[18] Unity Technologies, 2024. Unity - Manual: Canvas Components: [Unity - Manual: Canvas (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/class-Canvas.html)

[19] Unity Technologies, 2024. Unity - Manual: Canvas Scaler: [Unity - Manual: Canvas Scaler (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/script-CanvasScaler.html)

[20] Unity Technologies, 2024. Unity - Manual: Visual Components: [Unity - Manual: Visual Components (unity3d.com)](https://docs.unity3d.com/540/Documentation/Manual/comp-UIVisual.html)

[21] Unity Technologies, 2024. Unity - Manual: Tilemap: [Unity - Manual: Tilemaps (unity3d.com)](https://docs.unity3d.com/Manual/Tilemap.html)

Audios del juego: [TMK | Downloads | Sounds & Music | Sound Clips | Super Mario Bros. (NES) (themushroomkingdom.net)](https://themushroomkingdom.net/media/smb/wav)

Sprites del juego: [NES - Super Mario Bros. - The Spriters Resource (spriters-resource.com)](https://www.spriters-resource.com/nes/supermariobros/)