Sonic's Adventure in Super Mario Bros: A Blend of Worlds

Javier J. Cordero Álvarez

Índice

1 Descripción General del Juego

Título del Juego:

Sonic's Adventure in Super Mario Bros: A Blend of Worlds

Descripción: ***Sonic's Adventure in Super Mario Bros: A Blend of Worlds*** es un juego de plataformas en 2D que fusiona el universo de Sonic con el mundo de Super Mario Bros. Encarna a Sonic en un mundo familiar de Mario, enfrentándote a enemigos como Pikachu, Koopa y la planta carnívora. Explora la capacidad de transformación de Sonic para enfrentar desafíos y descubre cómo cada transformación afecta la jugabilidad.

2 Mecánicas Clave

2.1 Personajes:

- **Sonic:** Rápido y ágil, vulnerable a cualquier enemigo.
- **Knuckles y Shadow:** Capaces de volver al estado Sonic al ser alcanzados por enemigos, cada uno con resistencias particulares.

2.2 Enemigos:

- **Pikachu:** Derrotable saltando sobre él.
- **Koopa:** Solo puede ser vencido si Sonic lo pisa para que se esconda o lo pisa mientras está escondido para lanzarlo.
- **Planta Carnívora:** Mortal para Sonic, pero devuelve a Knuckles y Shadow al estado Sonic.

2.3 Transformaciones:

- Cambiar entre Sonic, Knuckles y Shadow afecta la jugabilidad y la resistencia ante enemigos.

2.4 Saltos:

- Método principal para derrotar enemigos y superar obstáculos.

2.5 Caparazones y Habilidades:

- Sonic tiene la capacidad de enfrentar los enemigos únicamente al saltar sobre ellos.
- Las transformaciones a Knuckles y Shadow permiten resistir y recuperar el estado Sonic al ser alcanzados por enemigos específicos.
- Caparazón de Koopa: Si se queda escondido durante 5 segundos, Koopa volverá a salir de su caparazón.

2.6 Movimiento y Velocidad:

- Variaciones en la velocidad y movimientos entre los personajes.

2.7 Entorno:

- Elementos familiares de Super Mario que ofrecen plataformas y obstáculos interactivos.

3 Implementaciones de mecánicas y configuraciones clave en el código

En este apartado se describen las configuraciones y mecánicas de los principales scripts del proyecto.

3.1 Script Sonic:

El script **Sonic** se encarga de controlar el comportamiento del personaje principal, quien es el jugador controlado en el juego. Este script maneja las interacciones del personaje, sus transformaciones y sus reacciones ante ciertos eventos en el entorno del juego.

Funciones Principales del Script Sonic:

➢ **Manejo de Estados:**

- Utiliza un sistema de estados (**enum State**) para determinar el estado actual del personaje (Sonic, Knuckles o Shadow) y cambia entre ellos según ciertas condiciones.

➢ **Referencias a Componentes y Clases:**

- Obtiene referencias a otros componentes importantes para el control del personaje, como **Mover**, **Colisiones**, **Animaciones**, **Rigidbody2D**, entre otros.

➢ **Reacción a Eventos de Juego:**

- Define cómo reacciona el personaje ante eventos clave del juego, como ser tocado por enemigos (**Hit()**), morir (**Dead()**), y verificar si cae por debajo de un límite en la pantalla (**transform.position.y < bottomOfCamera**).

➢ **Control de Transformaciones:**

- Cambia entre los estados de Sonic, Knuckles y Shadow según condiciones específicas.
- Define cómo el personaje regresa al estado Sonic cuando es afectado por enemigos en estados alternativos.

➢ **Ajuste de Colliders:**

- Contiene métodos para ajustar el **BoxCollider2D** del personaje dependiendo de la forma actual, por ejemplo, modificando la posición del collider cuando está en estado Knuckles.

➢ **Activación de Zonas de Ataque (stompBox):**

- Activa y desactiva una zona de ataque (**stompBox**) cuando el personaje está cayendo, lo que permite derrotar a enemigos al saltar sobre ellos.

➢ **Pausa y Reactivación de Movimiento:**

- Controla la pausa y reanudación del movimiento del personaje en situaciones específicas, como al ser alcanzado por enemigos o durante ciertas animaciones.

En resumen, el script **Sonic** es el controlador principal del personaje jugable, gestionando su comportamiento, transformaciones, interacciones con el entorno y reacciones a eventos clave en el juego.

3.2 Script CameraFollow

El script **CameraFollow** se encarga de controlar el movimiento de la cámara en el juego para mantener al personaje principal centrado y visible en la pantalla a medida que se desplaza por el nivel. Esta funcionalidad es común en juegos de plataformas en 2D para asegurar que el jugador siempre tenga una buena visión del entorno y del personaje que controla.

Funciones Principales del Script CameraFollow:

- **Referencia al Personaje Principal (target):**
  - El script tiene una referencia al objeto que debe seguir, usualmente el transform del personaje principal.
- **Control del Desplazamiento de la Cámara:**
  - El script calcula la posición a la que la cámara debe moverse para seguir al personaje. Suele hacerlo usando la posición del personaje más un desplazamiento (**followAhead**) que puede ajustarse para crear cierta holgura entre el personaje y el borde de la pantalla.
- **Restricción de Movimiento de la Cámara (minPosX, maxPosX):**
  - Puede tener límites en el movimiento horizontal para evitar que la cámara se desplace fuera de ciertos límites en el nivel. Esto asegura que la cámara no muestre áreas no deseadas o vacías.
- **Actualización del Movimiento en el Método Update():**
  - Normalmente, el movimiento de la cámara se actualiza en cada frame (en el método **Update()** o **LateUpdate()**) para seguir al personaje mientras se mueve por el nivel.
- **Centrado y Seguimiento Continuo:**
  - Ajusta la posición de la cámara continuamente para mantener al personaje centrado y visible en la pantalla, lo que proporciona una experiencia de juego más fluida y cómoda para el jugador.

En resumen, el script **CameraFollow** es responsable de asegurar que la cámara siga al personaje principal y se mantenga centrada en él mientras se desplaza por el nivel, permitiendo una visualización óptima del entorno y una experiencia de juego más cómoda.

3.3 Script Enemy

El script **Enemy** es una clase padre que actúa como un esquema general para todos los enemigos presentes en el juego. Proporciona funcionalidades comunes y métodos base que pueden ser heredados por todas las clases de los enemigos en el juego.

Funciones Principales del Script Enemy:

- **Referencias a Componentes:**
  - Obtiene referencias a componentes comunes que los enemigos podrían necesitar. Esto incluye el componente **Animator** para controlar las animaciones, **AutoMovement** para el movimiento automatizado y **Rigidbody2D** para la física del objeto.
- **Métodos Virtuales:**
  - Contiene métodos que pueden ser sobrescritos (o extendidos) por los enemigos específicos que hereden de esta clase base. Por ejemplo:
  - **protected virtual void Awake()**: Permite que las clases derivadas realicen inicializaciones adicionales específicas para cada enemigo.
  - **protected virtual void Update()**: Puede ser extendido para incluir comportamientos únicos de cada enemigo sin reescribir todo el comportamiento común.
- **Método public virtual void Stomped(Transform player):**
  - Define la acción que ocurre cuando un enemigo es "aplastado" o derrotado por el jugador.
  - Se define como virtual, lo que significa que los enemigos específicos pueden sobrescribir este método para definir su comportamiento al ser derrotados.
- **Uso en Enemigos Específicos:**
  - Este script sirve como una base común para otros scripts de enemigos como **Pikachu**, **Koopa** y **Planta Carnívora**, quienes heredan de **Enemy** y probablemente extiendan o sobrescriban algunos de sus métodos para definir su comportamiento único, como la reacción al ser derrotados, sus movimientos específicos, etc.

En resumen, el script **Enemy** es una clase base que proporciona funcionalidades comunes y métodos generales que pueden ser heredados y extendidos por enemigos específicos del juego para definir su comportamiento individual.

3.4 Script Pikachu

Este script controla las acciones que realiza el enemigo Pikachu cuando interactúa con el personaje principal (Sonic u otros) o con el entorno del juego.

Funciones Principales del Script Pikachu:

- **Herencia de la Clase Base Enemy:**
  - Hereda funcionalidades y comportamientos básicos de la clase base **Enemy**, lo que incluye métodos como **Stomped()** que pueden ser sobreescritos para definir el comportamiento específico del enemigo al ser derrotado.
- **Reacción al ser Aplastado (Stomped(Transform player)):**
  - Define la acción que ocurre cuando el personaje principal (o cualquier otro objeto) "aplasta" o derrota al enemigo Pikachu.
  - En este caso, reproduce un sonido específico (**AudioManager.instance.PlayFlipDie()**) y activa una animación (**animator.SetTrigger("Hit")**) para indicar que ha sido derrotado.
  - Finalmente, el objeto de Pikachu se destruye después de un tiempo (**Destroy(gameObject, 1f)**) para eliminarlo del juego de manera gradual.

En resumen, el script **Pikachu** se encarga de definir cómo reacciona el enemigo Pikachu al ser derrotado por el personaje principal. Esta reacción implica reproducir efectos de sonido, activar animaciones y eliminar el objeto del juego después de un cierto tiempo para simular su derrota.

3.5 Script Koopa

Este script controla las acciones que realiza el enemigo Koopa cuando interactúa con el personaje principal (Sonic u otros) o con el entorno del juego.

Funciones Principales del Script Koopa:

- **Atributos y Variables:**
  - **isHidden**: Indica si el Koopa está escondido dentro de su caparazón.
  - **maxStoppedTime**: Tiempo máximo que el Koopa permanece escondido antes de reaparecer.
  - **rollingSpeed**: Velocidad de movimiento del Koopa al ser pisado o esquivado.
- **Método Update():**
  - Verifica si el Koopa está escondido (**isHidden**) y su velocidad es cero (**rb2D.velocity.x == 0f**). Si cumple estas condiciones, comienza un temporizador para determinar cuánto tiempo permanecerá oculto (**stoppedTimer**).
- **Reacción al ser Aplastado (Stomped(Transform player)):**
  - Define la acción que ocurre cuando el personaje principal (u otro objeto) aplasta al enemigo Koopa.
  - Si el Koopa no está escondido, se esconde (**isHidden = true**) y detiene su movimiento (**autoMovement.PausedMovement()**).
  - Si está escondido y se le aplasta mientras está escondido, determina la dirección de movimiento basándose en la posición del jugador para lanzarse en esa dirección.
  - Luego, ajusta la capa del objeto (**gameObject.layer**) para cambiar su comportamiento de colisión temporalmente y restablece esta capa después de un breve período (**Invoke("ResetLayer", 0.1f)**).
- **Métodos Internos:**
  - **ResetLayer()**: Restaura la capa del objeto a la capa de enemigos después de un tiempo.
  - **ResetMove()**: Reanuda el movimiento del Koopa, lo hace visible y reinicia el temporizador de escondite.

En resumen, el script **Koopa** controla el comportamiento del enemigo Koopa, permitiéndole esconderse dentro de su caparazón cuando es aplastado, lanzarse en una dirección determinada al ser pisado mientras está oculto y manejar su interacción con otros elementos del juego.

3.6 Script Mover

El script **Mover** se encarga de controlar el movimiento del personaje principal en el juego, gestionando su movimiento horizontal, saltos, aceleración, y otras interacciones físicas básicas.

Funciones Principales del Script Mover:

- **Control de Dirección y Velocidad:**
  - Utiliza un enum **Direction** para controlar la dirección del movimiento (**Left = -1**, **None = 0**, **Right = 1**).
  - Ajusta la velocidad del personaje basado en la dirección actual y el valor de aceleración.
  - Gestiona la fricción para detener el movimiento gradualmente cuando no se presiona ninguna tecla de dirección.
- **Saltos y Gravedad:**
  - Maneja la lógica del salto del personaje, verificando si está en el suelo (**grounded**) y aplicando una fuerza vertical (**jumpForce**) para realizar saltos.
  - Controla el tiempo máximo de salto (**maxJumpingTime**) para limitar la duración del salto.
  - Ajusta la gravedad del personaje durante el salto para proporcionar una sensación más realista.
- **Animaciones y Movimiento Visual:**
  - Actualiza animaciones relacionadas con la velocidad (**Velocity()**) y los saltos (**Jumping()**), cambiando entre animaciones según las acciones del personaje.
  - Controla la animación de "deslizamiento" (**Skid**) basado en la fricción y la dirección del movimiento.

- **Referencias a Componentes y Clases:**

  - Obtiene referencias a otros componentes necesarios para el control del personaje, como **Rigidbody2D**, **Colisiones** y **Animaciones**.

- **Métodos para Manipulación del Movimiento:**

  - Define métodos para manejar la muerte del personaje (**Dead()**) y para realizar un impulso hacia arriba (**BounceUp()**).

- **Control de Estado:**

  - Gestiona estados del personaje como la habilidad de moverse (**inputMoveEnabled**) y la verificación de si está saltando (**isJumping**).

En resumen, el script **Mover** controla el movimiento, saltos, físicas y animaciones del personaje principal en el juego, proporcionando funcionalidades esenciales para la jugabilidad y la interacción del jugador con el entorno del juego.

3.7 Script Animaciones

El script **Animaciones** se encarga de controlar las animaciones del personaje principal en el juego, gestionando la transición y reproducción de las diferentes animaciones dependiendo de su estado y acciones.

Funciones Principales del Script Animaciones:

- **Referencia al Componente Animator:**

  - Obtiene una referencia al componente **Animator** del objeto para poder controlar las animaciones.

- **Control de Animaciones por Estado:**

  - Gestiona las animaciones relacionadas con los estados del personaje, como la animación de caminar, correr, estar en el suelo, saltar, etc.

  - Actualiza las variables de animación según el estado actual del personaje, como si está en el suelo (**Grounded()**), la velocidad (**Velocity()**), si está saltando (**Jumping()**), si está realizando un deslizamiento (**Skid()**), etc.

- **Control de Transiciones entre Animaciones:**

  - Activa y desactiva las animaciones correspondientes según las acciones del personaje y su estado.

  - Utiliza triggers (**SetTrigger**) o booleanos (**SetBool**) en el **Animator** para activar o desactivar ciertas animaciones en respuesta a eventos específicos.

- **Métodos para Activar Animaciones Específicas:**

  - Contiene métodos específicos para activar ciertas animaciones en momentos clave, como cuando el personaje muere (**Dead()**), cambia de estado (**NewState()**), obtiene un power-up (**PowerUp()**), o es golpeado (**Hit()**).

En resumen, el script **Animaciones** administra y controla las transiciones y reproducciones de las animaciones del personaje principal en el juego, asegurando que las animaciones se reproduzcan adecuadamente según el estado y las acciones del personaje para ofrecer una experiencia visual coherente y atractiva.

3.8 Script AudioManager

El script **AudioManager** se encarga de gestionar y controlar los efectos de sonido dentro del juego. Controla la reproducción de clips de audio en respuesta a eventos específicos que ocurren durante el juego.

Funciones Principales del Script AudioManager:

- **Referencias a Clips de Audio:**

  - Contiene referencias a varios clips de audio (**AudioClip**), como los sonidos de salto, muerte, impacto, entre otros, que se utilizarán en el juego.

- **Control de Reproducción de Audio:**

  - Utiliza un componente **AudioSource** para reproducir los clips de audio.

  - Proporciona métodos para reproducir sonidos individuales (**PlayJump()**, **PlayBigJump()**, **PlayStomp()**, etc.) utilizando el método **PlayOneShot**, que reproduce el sonido una vez sin interrumpir los sonidos anteriores que aún se están reproduciendo.

➢ **Singleton Pattern:**

- Implementa el patrón Singleton para garantizar que solo haya una instancia activa de **AudioManager** en el juego, permitiendo el acceso global a sus métodos y evitando la duplicación accidental del objeto.

➢ **Inicialización y Asignación de Referencias:**

- En el método **Awake()**, comprueba si ya existe una instancia de **AudioManager**. Si no existe, asigna la instancia actual a **instance** y obtiene una referencia al componente **AudioSource** adjunto al objeto.

➢ **Métodos para Reproducir Sonidos:**

- Cada método público del **AudioManager** corresponde a un sonido específico que puede ser activado desde otras clases o scripts del juego.

En resumen, el script **AudioManager** controla la reproducción de efectos de sonido en el juego, proporcionando métodos para reproducir diferentes clips de audio en respuesta a eventos específicos, mejorando así la experiencia auditiva del jugador durante el juego.

4 Objetivos:

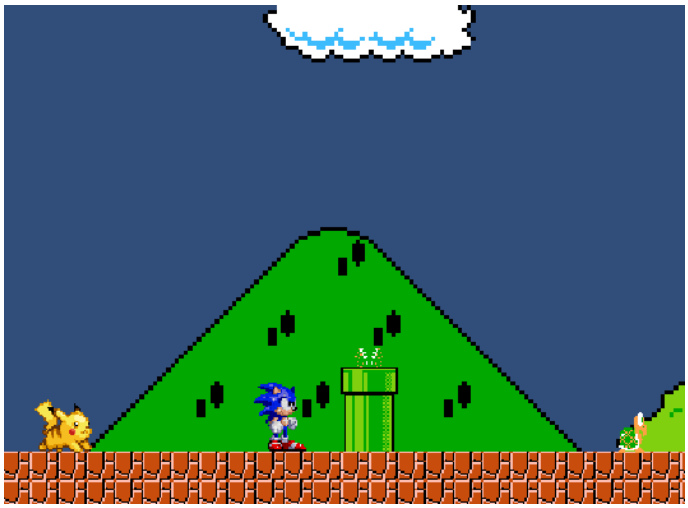
- Superar obstáculos clásicos de Super Mario Bros con las habilidades de Sonic, Knuckles o Shadow.
- Derrotar enemigos y llegar a la meta del nivel para avanzar en la historia.

5 Capturas de Pantalla

A continuación, se muestran las capturas más representativas en las que se incluyen momentos clave o características visuales distintivas del juego:

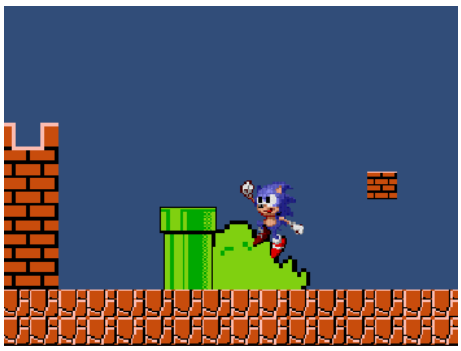
1. Escenas Jugables:

- Una captura de pantalla del personaje principal interactuando con el entorno del juego.

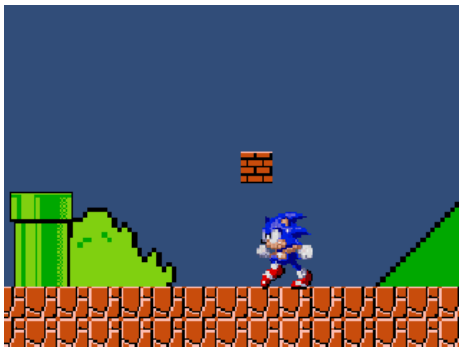


- Una escena que muestre la mecánica de salto o movimiento característico del personaje.

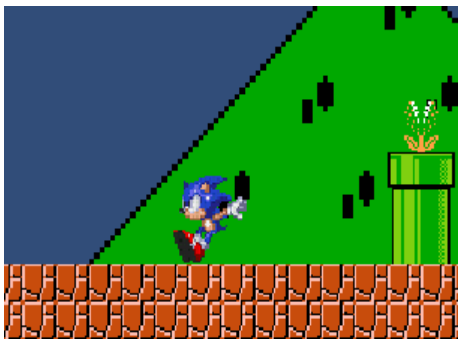
Salto



Correr

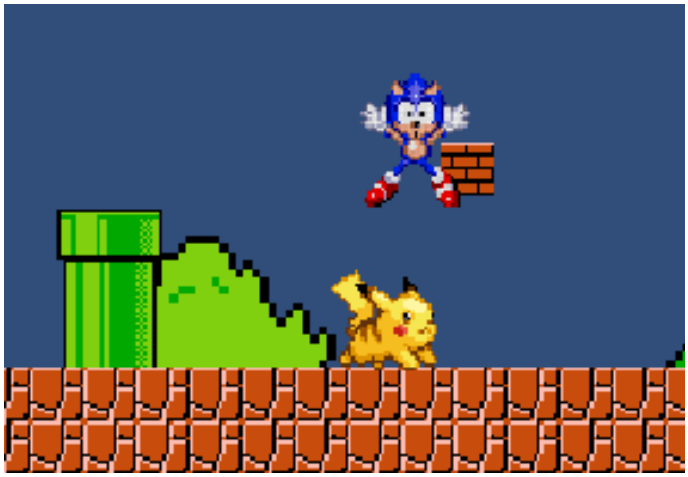


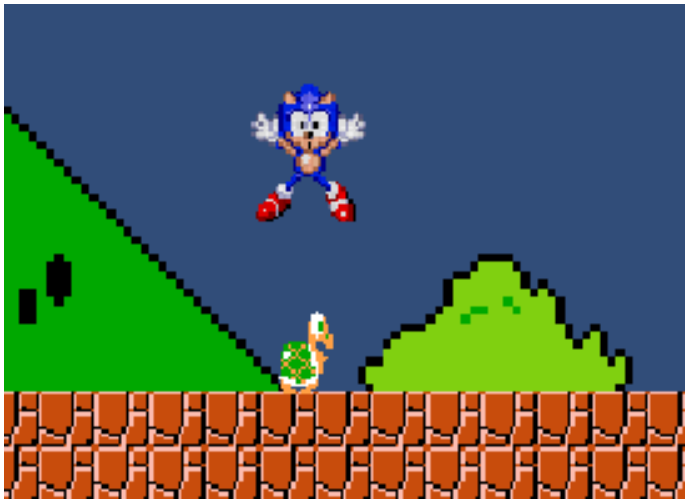
Skid

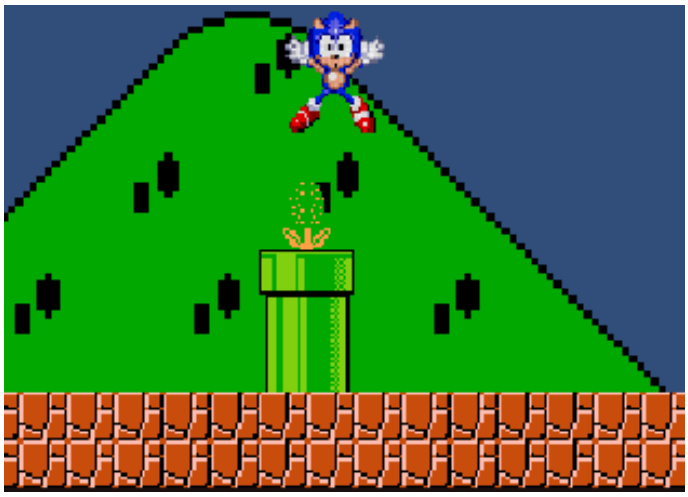


2. **Interacciones con Enemigos:**

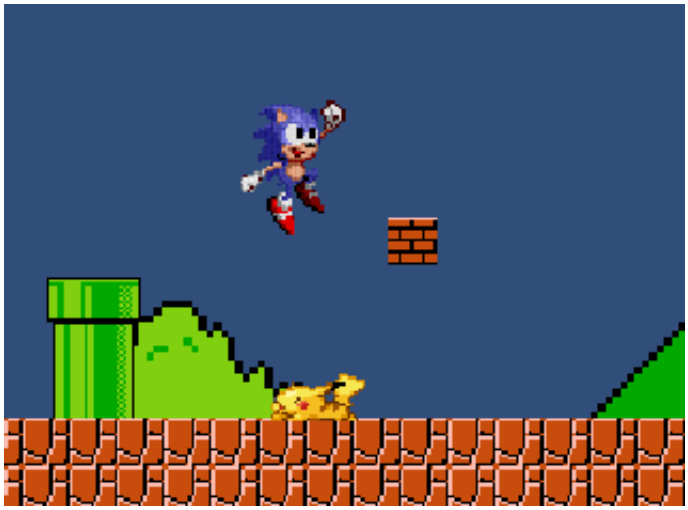
- Una imagen que muestre la confrontación con enemigos como Pikachu, Koopa o la Planta Carnívora.

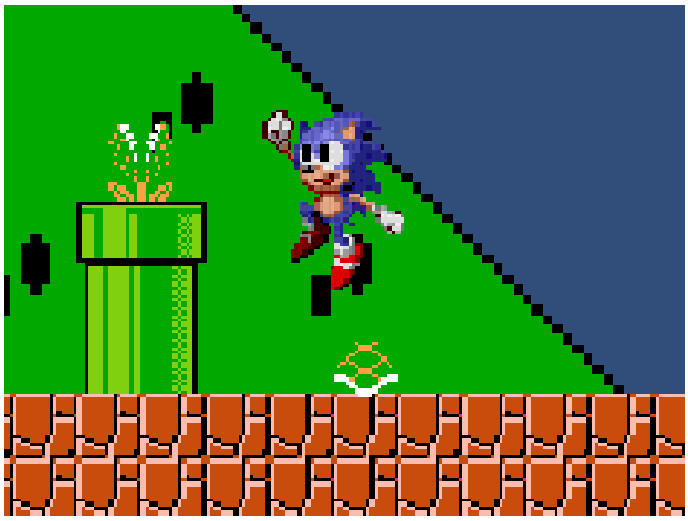


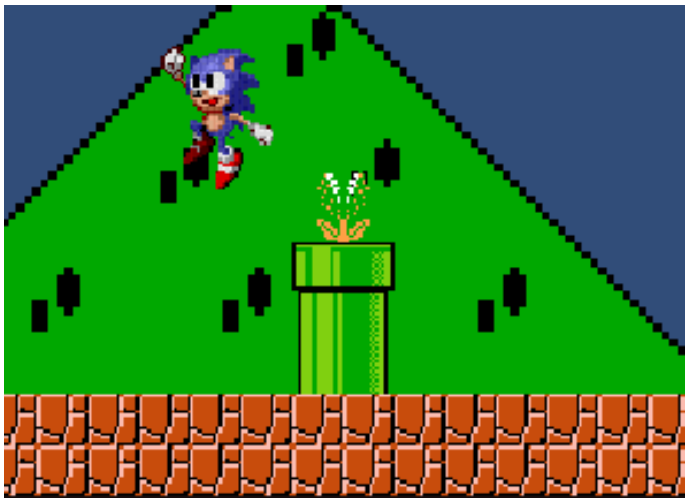




- Momentos de interacción entre el personaje y los enemigos, como saltar sobre Koopa, Pikachu o evitar a la Planta Carnívora.



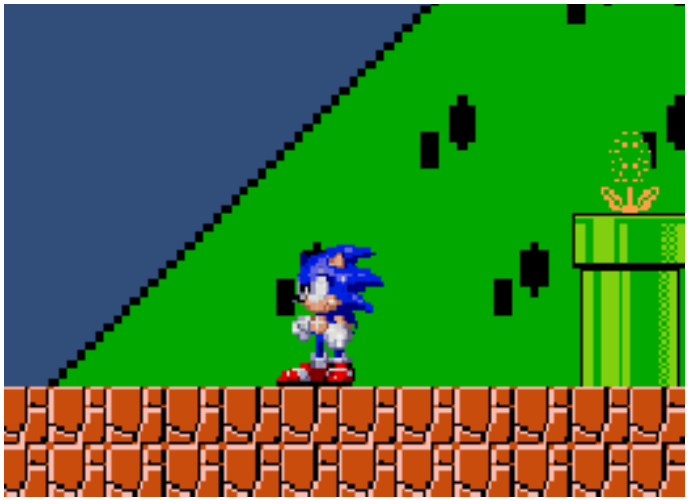


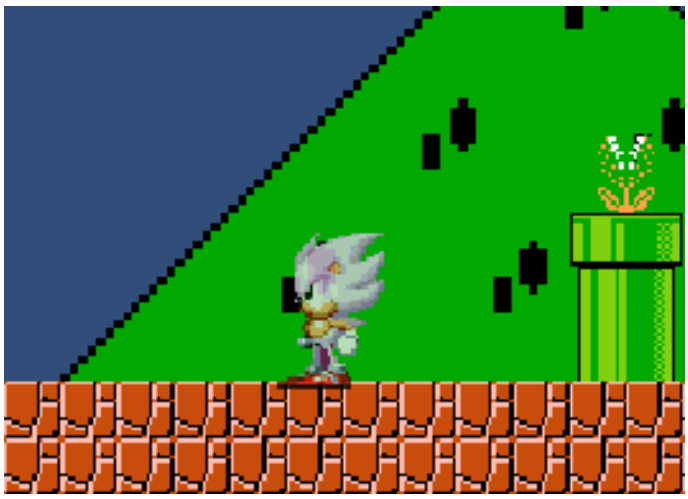


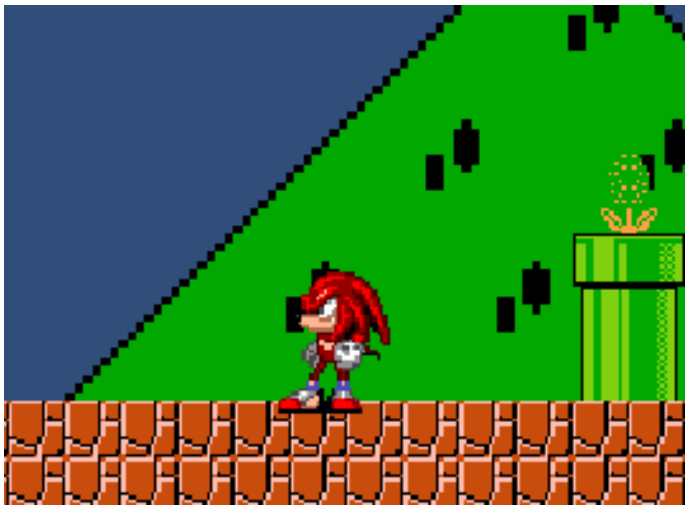
3. **Power-ups o Transformaciones:**

- Capturas que muestren la transformación del personaje principal al obtener un power-up o al cambiar de estado.

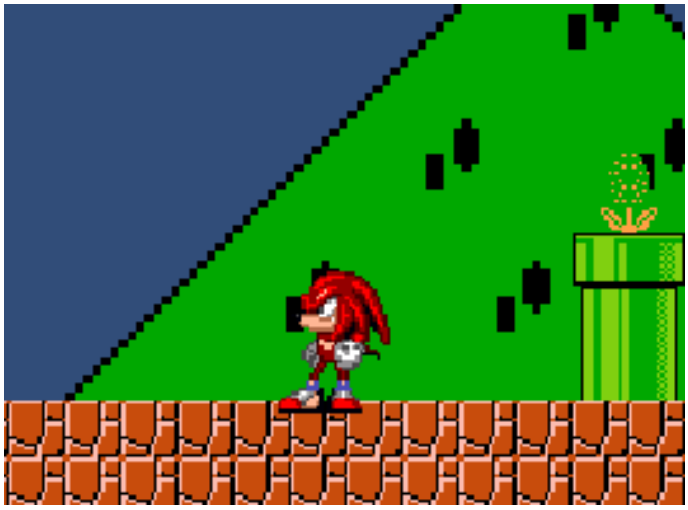
Sonic a Knuckles

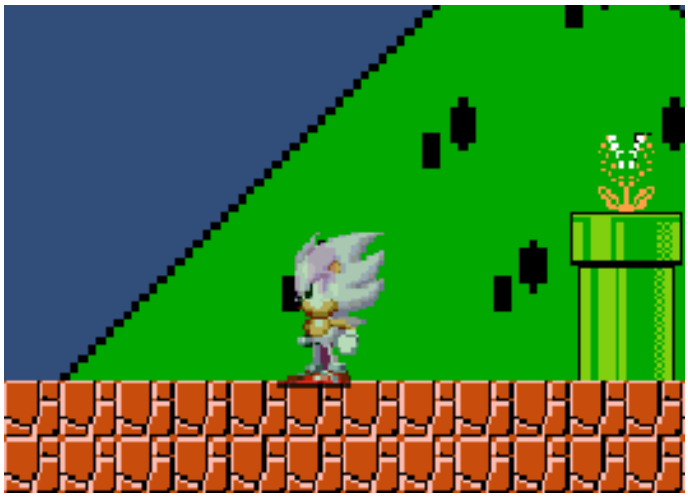


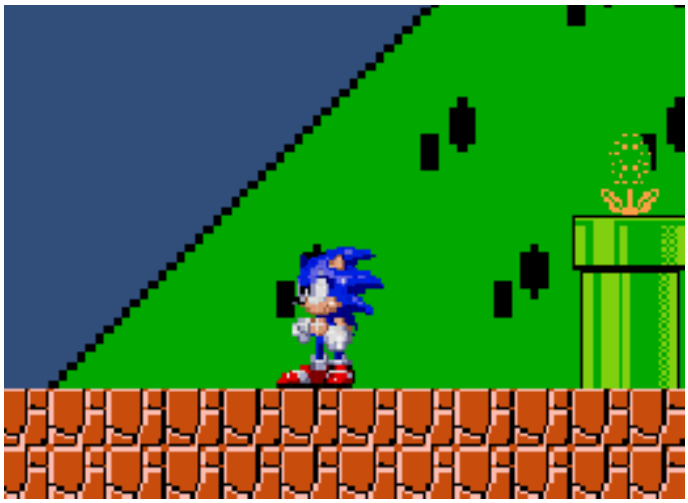




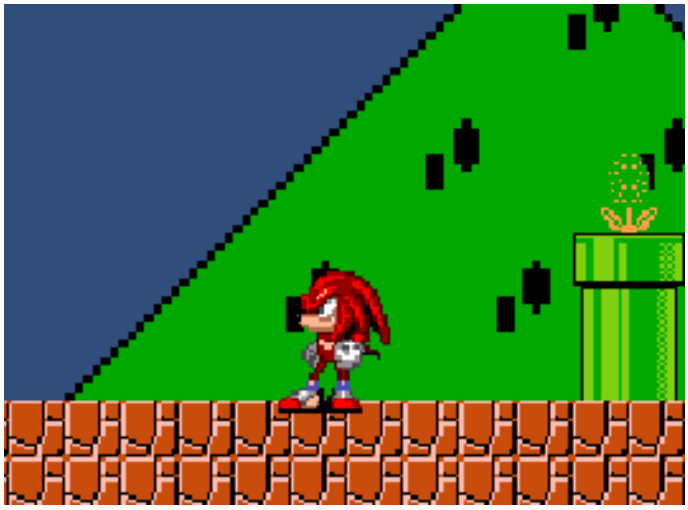
Knuckles a Sonic



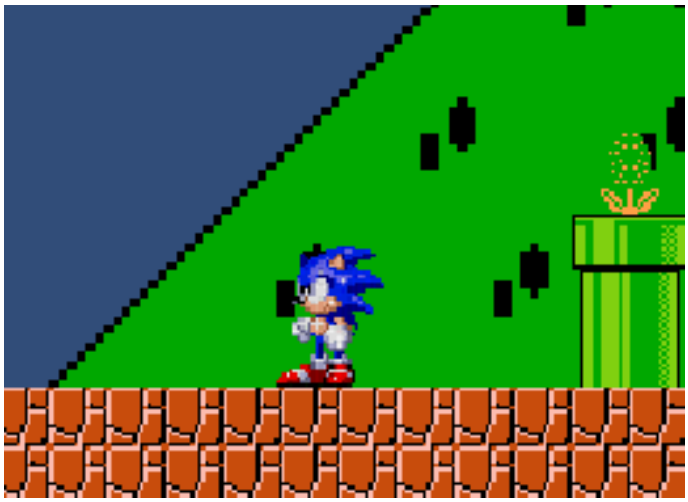




Knuckles a Shadow

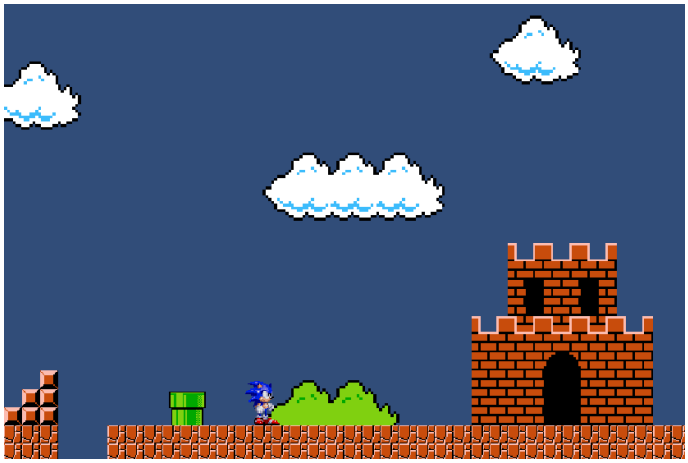


Shadow a Sonic



4. **Ambientes o Escenarios Destacados:**

- Imágenes que resalten el diseño visual del juego, ya sea un nivel específico, un paisaje característico o elementos de fondo significativos.



Estas capturas pueden proporcionar una visión general del juego, destacando sus mecánicas, interacciones clave y el diseño visual.

6 Conclusión

El juego combina la nostalgia de dos universos icónicos, ofreciendo una experiencia única donde los jugadores pueden explorar niveles familiares con un giro emocionante. La mecánica de transformación y la interacción con enemigos icónicos prometen desafíos emocionantes para los fanáticos de ambos juegos.

El proceso de desarrollo de este juego en Unity ha representado un desafiante y enriquecedor viaje hacia la comprensión y aplicación de conocimientos sólidos en el ámbito de la programación y el desarrollo de videojuegos. La manipulación de Unity, junto con la escritura de scripts en C#, ha permitido explorar y entender en profundidad aspectos esenciales del desarrollo de juegos: desde la creación de mecánicas de juego hasta la implementación de interacciones entre personajes y entornos. Este proyecto ha sido una plataforma fundamental para adquirir una comprensión más amplia y detallada tanto de la funcionalidad de Unity como del lenguaje de programación C#, consolidando así un conjunto diverso de habilidades que son esenciales en la creación de experiencias interactivas y entretenidas.

7 Repositorio del juego

https://github.com/javi-dev-79/Super-Sonic-Bros