Informe de recopilación de Scripts (Proyecto Arkmed)

Santiago Castañeda Pérez

Universidad Tecnologica de Pereira
Computación Grafica
Profesor: Andrés Felipe Ramírez

22 de mayo de 2025

Índice

1. AudioManager.cs

```csharp
using UnityEngine;

public class AudioManager : MonoBehaviour
{
    public static AudioManager instancia;

    public AudioSource Musica_Fondo;
    public AudioClip Sonido_Click;

    private void Awake()
    {
        if (instancia == null)
        {
            instancia = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    public void ReproducirClick()
    {
        AudioSource.PlayClipAtPoint(Sonido_Click, Camera.main.transform.position);
    }

    public void DetenerMusica()
    {
        if (Musica_Fondo != null)
            Musica_Fondo.Stop();
    }
}
```

Listing 1: AudioManager.cs

2. MainMenuManager.cs

```csharp
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenuManager : MonoBehaviour
{
    public GameObject PanelControles;  // Panel donde est n los controles
    public GameObject Jugar;           // Bot n "Jugar"
    public GameObject Salir;           // Bot n "Salir"
    public GameObject Controles;       // Bot n "Controles"
    public GameObject Cerrar;          // Bot n "Cerrar" en el Panel de Controles

    // Funci n para iniciar el juego
    public void JugarJuego()
    {
        if (AudioManager.instancia != null)
        {
            AudioManager.instancia.DetenerMusica();
        }

        PlayerPrefs.DeleteAll();

        SceneManager.LoadScene("IntroScene");
    }

    // Funci n para mostrar los controles
    public void MostrarControles()
    {
        PanelControles.SetActive(true);  // Activa el panel de controles
        Jugar.SetActive(false);          // Desactiva el bot n Jugar
        Salir.SetActive(false);          // Desactiva el bot n Salir
        Controles.SetActive(false);      // Desactiva el bot n
   Controles
        Cerrar.SetActive(true);          // Activa el bot n Cerrar
    }

    // Funci n para ocultar los controles y regresar al men  principal
    public void OcultarControles()
    {
        PanelControles.SetActive(false); // Desactiva el panel de
   controles
        Jugar.SetActive(true);           // Activa el bot n Jugar
        Salir.SetActive(true);           // Activa el bot n Salir
        Controles.SetActive(true);       // Activa el bot n Controles
        Cerrar.SetActive(false);         // Desactiva el bot n Cerrar
    }

    public void SalirDelJuego()
    {
        Debug.Log("Saliendo del juego...");
        Application.Quit();
    }

    // M todo Update para desactivar la tecla Escape
    private void Update()
    {
        // No hacemos nada si se presiona Escape
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            // Escape est  desactivado
        }
    }
}
```

Listing 2: MainMenuManager.cs

3. IntroManager.cs

```
  using   System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;
using UnityEngine.UI;
```

```

public class IntroManager : MonoBehaviour
{
    [Header("UI")]
    public TextMeshProUGUI textoUI;
    public Image panelFondo; // para fade

    [Header("Texto")]
    [TextArea(3, 10)]
    public string[] parrafos;
    public float velocidadTipeo = 0.03f;

    [Header("Audio")]
    public AudioSource audioSource;
    public AudioClip sonidoLetra;

    [Header("Fade")]
    public float duracionFade = 0.5f;

    private int indiceParrafo = 0;
    private bool escribiendo = false;
    private bool puedeContinuar = false;

    void Start()
    {
        textoUI.text = "";

        // Inicializar datos del jugador
        if (PlayerPrefs.HasKey("Vida"))
        {
            Debug.Log("Cargando datos del jugador desde PlayerPrefs en IntroManager.");
            PlayerPrefs.GetInt("Vida"); // Vida del jugador
            PlayerPrefs.GetInt("Municion"); // Munici n total
            PlayerPrefs.GetInt("Puntuacion"); // Puntuaci n
            PlayerPrefs.GetInt("Botiquines"); // Botiquines
        }
        else
        {
            Debug.Log("No se encontraron datos guardados. Inicializando valores predeterminados en IntroManager.");
            PlayerPrefs.SetInt("Vida", 100); // Vida m xima
            PlayerPrefs.SetInt("Municion", 30); // Munici n m xima
            PlayerPrefs.SetInt("Puntuacion", 0); // Puntuaci n inicial
            PlayerPrefs.SetInt("Botiquines", 0); // Sin botiquines
            PlayerPrefs.Save();
        }

        StartCoroutine(MostrarParrafo()); // Mostrar el texto de introducci n
    }



    void Update()
    {
        if (!escribiendo && puedeContinuar && (Input.anyKeyDown || Input.GetMouseButtonDown(0)))
        {
            if (indiceParrafo < parrafos.Length)
            {
                StartCoroutine(MostrarParrafo());
            }
            else
            {
                SceneManager.LoadScene("GameplayScene 1");
            }
        }
    }

    IEnumerator MostrarParrafo()
    {
        escribiendo = true;
        puedeContinuar = false;
        textoUI.text = "";

        // Inicia sonido largo del p rrafo
        if (audioSource != null && sonidoLetra != null)
        {
            audioSource.Stop();
            audioSource.clip = sonidoLetra;
            audioSource.Play();
        }

        yield return StartCoroutine(FadeIn());

        string parrafo = parrafos[indiceParrafo];
        foreach (char letra in parrafo)
        {
            textoUI.text += letra;
            yield return new WaitForSeconds(velocidadTipeo);
        }

        // Detiene el sonido al terminar de escribir
        if (audioSource != null)
        {
            audioSource.Stop();
        }

        yield return new WaitForSeconds(0.5f);
        indiceParrafo++;
        puedeContinuar = true;
        escribiendo = false;
    }



    IEnumerator FadeIn()
    {
        float tiempo = 0;
        Color color = panelFondo.color;

        while (tiempo < duracionFade)
        {
            tiempo += Time.deltaTime;
            float alpha = Mathf.Lerp(0f, 1f, tiempo / duracionFade);
            panelFondo.color = new Color(color.r, color.g, color.b,
    alpha);
            yield return null;
        }

        panelFondo.color = new Color(color.r, color.g, color.b, 1f);
    }
}
```

Listing 3: IntroManager.cs

4. FinalManager.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;
using UnityEngine.UI;

public class FinalManager : MonoBehaviour
{
    [Header("UI")]
    public TextMeshProUGUI textoUI;
    public Image panelFondo; // para fade

    [Header("Texto")]
    [TextArea(3, 10)]
    public string[] parrafos;
    public float velocidadTipeo = 0.03f;

    [Header("Audio")]
    public AudioSource audioSource;
    public AudioClip sonidoLetra;

    [Header("Fade")]
    public float duracionFade = 0.5f;

    private int indiceParrafo = 0;
    private bool escribiendo = false;
    private bool puedeContinuar = false;

    void Start()
    {
        textoUI.text = "";
        StartCoroutine(MostrarParrafo());
    }

    void Update()
    {
        if (!escribiendo && puedeContinuar && (Input.anyKeyDown || Input
    .GetMouseButtonDown(0)))
        {
            if (indiceParrafo < parrafos.Length)
            {
                StartCoroutine(MostrarParrafo());
            }
            else
            {
                SceneManager.LoadScene("Credits");
            }
        }
    }

    IEnumerator MostrarParrafo()
    {
        escribiendo = true;
        puedeContinuar = false;
        textoUI.text = "";

        // Sonido de tipeo
        if (audioSource != null && sonidoLetra != null)
        {
            audioSource.Stop();
            audioSource.clip = sonidoLetra;
            audioSource.Play();
        }

        yield return StartCoroutine(FadeIn());

        string parrafo = parrafos[indiceParrafo];
        foreach (char letra in parrafo)
        {
            textoUI.text += letra;
            yield return new WaitForSeconds(velocidadTipeo);
        }

        if (audioSource != null)
        {
            audioSource.Stop();
        }

        yield return new WaitForSeconds(0.5f);
        indiceParrafo++;
        puedeContinuar = true;
        escribiendo = false;
    }

    IEnumerator FadeIn()
    {
        float tiempo = 0;
        Color color = panelFondo.color;

        while (tiempo < duracionFade)
        {
            tiempo += Time.deltaTime;
            float alpha = Mathf.Lerp(0f, 1f, tiempo / duracionFade);
            panelFondo.color = new Color(color.r, color.g, color.b,
   alpha);
            yield return null;
        }

        panelFondo.color = new Color(color.r, color.g, color.b, 1f);
    }
}
```

Listing 4: FinalManager.cs

5. Credits.cs

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;
using UnityEngine.UI;

public class Credits : MonoBehaviour
{
    [Header("UI")]
    public TextMeshProUGUI textoUI;
    public Image panelFondo; // para fade

    [Header("Texto")]
    [TextArea(3, 10)]
    public string[] parrafos;
    public float velocidadTipeo = 0.03f;

    [Header("Audio")]
    public AudioSource audioSource;
    public AudioClip sonidoLetra;

    [Header("Fade")]
    public float duracionFade = 0.5f;

    private int indiceParrafo = 0;
    private bool escribiendo = false;
    private bool puedeContinuar = false;

    void Start()
    {
        textoUI.text = "";
        StartCoroutine(MostrarParrafo());
    }

    void Update()
    {
        if (!escribiendo && puedeContinuar && (Input.anyKeyDown || Input
   .GetMouseButtonDown(0)))
        {
            if (indiceParrafo < parrafos.Length)
            {
                StartCoroutine(MostrarParrafo());
            }
            else
            {
                SceneManager.LoadScene("MainMenu");
            }
        }
    }

    IEnumerator MostrarParrafo()
    {
        escribiendo = true;
        puedeContinuar = false;
        textoUI.text = "";

        // Sonido de tipeo
        if (audioSource != null && sonidoLetra != null)
        {
            audioSource.Stop();
            audioSource.clip = sonidoLetra;
            audioSource.Play();
        }

        yield return StartCoroutine(FadeIn());

        string parrafo = parrafos[indiceParrafo];
        foreach (char letra in parrafo)
        {
            textoUI.text += letra;
            yield return new WaitForSeconds(velocidadTipeo);
        }

        if (audioSource != null)
        {
            audioSource.Stop();
        }

        yield return new WaitForSeconds(0.5f);
        indiceParrafo++;
        puedeContinuar = true;
        escribiendo = false;
    }

    IEnumerator FadeIn()
    {
        float tiempo = 0;
        Color color = panelFondo.color;

        while (tiempo < duracionFade)
        {
            tiempo += Time.deltaTime;
            float alpha = Mathf.Lerp(0f, 1f, tiempo / duracionFade);
            panelFondo.color = new Color(color.r, color.g, color.b,
   alpha);
            yield return null;
        }

        panelFondo.color = new Color(color.r, color.g, color.b, 1f);
    }
}
```

Listing 5: Credits.cs

6. ZombieAI.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ZombieAI : MonoBehaviour
{
    public float speed = 2f;
    public float attackRange = 1f;
    public float attackCooldown = 1f;
    public float health = 100f;
    public float stunDuration = 1f;

    public AudioClip growlSound;
    public AudioClip hitSound;
    public AudioClip deathSound;

    public GameObject healthBarPrefab;
    private Image healthBarFill;
    private GameObject healthBarInstance;
    private Vector3 healthBarOffset = new Vector3(0, 1.5f, 0); // Ajusta la altura

    private Transform player;
    private Animator animator;
    private Rigidbody2D rb;
    private AudioSource audioSource;

    private float lastAttackTime;
    private bool isStunned = false;
    private bool isDead = false;

    [SerializeField] private GameObject medkitPrefab;
    [SerializeField] private GameObject ammoPrefab;
    [SerializeField] private GameObject keyPrefab; // Prefab de la llave

    void Start()
    {
        if (healthBarPrefab != null)
        {
            // Instanciar la barra de vida y obtener el componente de relleno
            healthBarInstance = Instantiate(healthBarPrefab, transform.position + healthBarOffset, Quaternion.identity);
            healthBarFill = healthBarInstance.GetComponentInChildren<Image>();
        }

        player = GameObject.FindGameObjectWithTag("Player")?.transform;
        animator = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        audioSource = GetComponent<AudioSource>();

        PlayGrowl();
    }

    void Update()
    {
        if (player == null || isDead) return;

        if (isStunned)
        {
            animator.SetBool("isRunning", false);
            return;
        }

        float distance = Vector2.Distance(transform.position, player.position);
        bool isAttacking = distance <= attackRange;

        animator.SetBool("isAttacking", isAttacking);
        animator.SetBool("isRunning", !isAttacking);

        if (!isAttacking)
        {
            Vector2 direction = (player.position - transform.position).normalized;
            transform.position += (Vector3)direction * speed * Time.deltaTime;

            if (direction.x > 0)
                transform.localScale = new Vector3(1, 1, 1);
            else if (direction.x < 0)
                transform.localScale = new Vector3(-1, 1, 1);
        }
        else
        {
            if (Time.time - lastAttackTime >= attackCooldown)
            {
                Attack();
                lastAttackTime = Time.time;
            }
        }

        // Seguir al zombie
        if (healthBarInstance != null)
        {
            healthBarInstance.transform.position = transform.position + healthBarOffset;
        }
    }

    public void TakeDamage(float damage)
    {
        if (isDead) return;

        health -= damage;
        PlaySound(hitSound);

        if (health <= 0)
        {
            Die();
        }
        else
        {
            if (!isStunned)
            {
                isStunned = true;
                Invoke("ResetStun", stunDuration);
            }
        }

        if (healthBarFill != null)
        {
            float percentage = health / 100f;
            healthBarFill.fillAmount = percentage;
```

```csharp

            if (percentage > 0.6f)
                healthBarFill.color = Color.green;
            else if (percentage > 0.3f)
                healthBarFill.color = Color.yellow;
            else
                healthBarFill.color = Color.red;
        }
    }

    void DropLoot()
    {
        Vector3 dropPosition = transform.position + new Vector3(Random.Range(-0.3f, 0.3f), -0.3f, 0);

        if (Random.value < 0.5f)
            Instantiate(medkitPrefab, dropPosition, Quaternion.identity);
        else
            Instantiate(ammoPrefab, dropPosition, Quaternion.identity);
    }


    void DropKey()
    {
        if (keyPrefab != null)
        {
            if (GameObject.FindGameObjectWithTag("Key") == null)
            {
                Vector3 dropPosition = transform.position + new Vector3(0f, -0.3f, 0);
                Instantiate(keyPrefab, dropPosition, Quaternion.identity);
                Debug.Log("Llave soltada por el zombie.");
            }
            else
            {
                Debug.Log("Ya hay una llave en la escena, no se soltar otra.");
            }
        }
        else
        {
            Debug.LogWarning("El prefab de la llave no est  asignado en el inspector.");
        }
    }


    void Die()
    {
        isDead = true;
        animator.SetTrigger("DieTrigger");
        PlaySound(deathSound);

        // Suelta bot n y llave
        DropLoot();
        DropKey();

        // A adir puntuaci n al jugador
        if (player != null)
        {
            PlayerMovement playerScript = player.GetComponent<PlayerMovement>();
            if (playerScript != null)
            {
                int points = Random.value < 0.2f ? 10 : 5;
                string popup = points == 10 ? "  Crtico   +10!" : $"+{points}";
                playerScript.AddScore(points, transform.position, popup);
            }
        }

        Destroy(gameObject, 0.5f);
        Destroy(healthBarInstance);
    }

    void ResetStun()
    {
        isStunned = false;
        animator.SetBool("isStunned", false);
        rb.bodyType = RigidbodyType2D.Dynamic;
    }

    void Attack()
    {
        if (player != null)
        {
            PlayerMovement playerScript = player.GetComponent<PlayerMovement>();
            if (playerScript != null)
            {
                playerScript.TakeDamage(10);
            }
        }
    }

    void PlaySound(AudioClip clip)
    {
        if (audioSource != null && clip != null)
        {
            audioSource.PlayOneShot(clip);
        }
    }

    void PlayGrowl()
    {
        if (audioSource != null && growlSound != null)
        {
            audioSource.loop = true;
            audioSource.clip = growlSound;
            audioSource.Play();
        }
    }
```

```
}
```

Listing 6: ZombieAI.cs

7. TextoParpadeante.cs

```
using UnityEngine;
using TMPro;

public class TextoParpadeante : MonoBehaviour
{
    public float velocidadParpadeo = 1f;
    private TextMeshProUGUI textoUI;
    private Color colorOriginal;

    void Start()
    {
        textoUI = GetComponent<TextMeshProUGUI>();
        colorOriginal = textoUI.color;
    }

    void Update()
    {
        float alfa = Mathf.Abs(Mathf.Sin(Time.time * velocidadParpadeo))
    ;
        textoUI.color = new Color(colorOriginal.r, colorOriginal.g,
    colorOriginal.b, alfa);
    }
}
```

Listing 7: TextoParpadeante.cs

8. PoliceSiren.cs

```
using UnityEngine;

public class PoliceSiren : MonoBehaviour
{
    public GameObject redLight;
    public GameObject blueLight;
    public float blinkInterval = 0.3f;

    private float timer;
    private bool isRedActive = true;

    void Start()
    {
        redLight.SetActive(true);
        blueLight.SetActive(false);
    }

    void Update()
    {
        timer += Time.deltaTime;
        if (timer >= blinkInterval)
        {
            timer = 0f;
            isRedActive = !isRedActive;
            redLight.SetActive(isRedActive);
            blueLight.SetActive(!isRedActive);
        }
    }
}
```

Listing 8: PoliceSiren.cs

9. PauseManager.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class PausaManager : MonoBehaviour
{
    public GameObject panelPausa;
    public Slider healthBar;

    private bool juegoPausado = false;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (juegoPausado)
                Reanudar();
            else
                Pausar();
        }
    }

    public void Pausar()
    {
        panelPausa.SetActive(true);
        Time.timeScale = 0f;
        juegoPausado = true;

        // Ocultar todo lo que contenga Player
        PlayerMovement player = Object.FindFirstObjectByType<
    PlayerMovement>();
        if (player != null)
        {
            if (player.medkitText != null)
                player.PanelBotiquines.gameObject.SetActive(false);

            if (player.ammoText != null)
                player.PanelMunicion.gameObject.SetActive(false);

            if (player.healthBar != null)
                player.healthBar.gameObject.SetActive(false); // Ocultar
     healthBar

            if (player.keyPanel != null)
                player.keyPanel.gameObject.SetActive(false); // Ocultar
    keyPanel
        }
    }

    public void Reanudar()
    {
        panelPausa.SetActive(false);
        Time.timeScale = 1f;
        juegoPausado = false;

        // Mostrar todo lo que contenga Player
        PlayerMovement player = Object.FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            if (player.medkitText != null)
                player.PanelBotiquines.gameObject.SetActive(true);

            if (player.ammoText != null)
                player.PanelMunicion.gameObject.SetActive(true);

            if (player.healthBar != null)
                player.healthBar.gameObject.SetActive(true); // Mostrar healthBar

        player.UpdateKeyPanel();
        }
    }

    public void Salir()
    {
        panelPausa.SetActive(false);
        Time.timeScale = 1f;
        juegoPausado = false;

        // Destruir el jugador y todos sus hijos
        PlayerMovement player = Object.FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            Destroy(player.gameObject); // Destruye el jugador y todos sus hijos
        }

        SceneManager.LoadScene("MainMenu"); // Cambia a la escena del men principal
    }

}
```

Listing 9: PauseManager.cs

10. DeathMenu.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class DeathMenu : MonoBehaviour
{
    private static DeathMenu instance;
```

```csharp

    private void Awake()
    {
        if (instance != null && instance != this)
        {
            Destroy(gameObject);
            return;
        }

        instance = this;
        DontDestroyOnLoad(gameObject);
    }

    private void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    private void OnDisable()
    {
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        // Si persiste en una escena nueva, se destruye autom ticamente
        if (scene.name != "GameplayScene 1" && scene.name != "GameplayScene 2" && scene.name != "GameplayScene 3")
        {
            Destroy(gameObject);
        }
    }

    public void RestartLevel()
    {
        Time.timeScale = 1f;

        // Reiniciar el estado del jugador
        PlayerMovement player = FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            Destroy(player.gameObject); // Destruye el jugador y todos sus hijos
        }

        SceneManager.LoadScene("GameplayScene 1"); // Carga la escena deseada
    }

    public void GoToMainMenu()
    {
        Time.timeScale = 1f;

        // Buscar y destruir el objeto PlayerMovement
        PlayerMovement player = FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            Destroy(player.gameObject); // Destruye el jugador y todos
```

```
   sus hijos
        }

        SceneManager.LoadScene("MainMenu");
    }
}
```

Listing 10: DeathMenu.cs

11. PlayerMovement.cs

```
 using  UnityEngine;
using TMPro;
using System.Collections;
using UnityEngine.SceneManagement;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;
    public float jumpForce = 7f;
    public Animator animator;
    public GameObject bulletPrefab;
    public Transform firePoint;
    public float bulletSpeed = 10f;
    public TextMeshProUGUI ammoText;
    public AudioClip casingSound; // Nuevo sonido de casquillo
    public AudioClip reloadSound;
    public int maxAmmo = 30;      // Capacidad m xima de balas en total (valor fijo para referencia)
    public int clipSize = 5;      // Balas por recarga
    private int currentAmmo;      // Balas en el cargador (izquierda del /)
    private int totalAmmo;        // Balas en reserva (derecha del /)
    public GameObject PanelBotiquines;   // UI del jugador (opcional, puedes eliminarlo si no lo necesitas)
    public GameObject PanelMunicion;     // UI del jugador (opcional, puedes eliminarlo si no lo necesitas)
    private static PlayerMovement instance;
    public GameObject keyPanel; // UI opcional para mostrar la vida (puedes eliminarlo si no lo necesitas)
    // Removed unused field 'escenaCargando' as it was not being used in the code.

    public int medkitCount = 0;
    public TextMeshProUGUI medkitText; // Muestra cu ntos botiquines tienes

    public GameObject pickupPrompt; // UI de   Presiona   E para recoger
    [HideInInspector] public GameObject itemNearby;
    public AudioClip pickupSound;

    public TextMeshProUGUI scoreText;
    public GameObject scorePopupPrefab; // Prefab con un texto que flota

    private int score = 0;

    // Sonidos
```

```csharp
    public AudioClip shootSound;
    public AudioClip jumpSound;
    public AudioClip stepSound;

    private Rigidbody2D rb;
    private SpriteRenderer spriteRenderer;
    private AudioSource audioSource;
    private AudioSource stepAudioSource;

    // Suelo
    private bool isGrounded;
    public Transform groundCheck;
    public float groundCheckRadius = 0.2f;
    public LayerMask groundLayer;

    // Control de disparos
    private float lastShootTime = 0f;  // Tiempo del  ltimo  disparo
    private int shotsSinceLastReload = 0;
    public float shootCooldown = 0.2f;   // Tiempo de recarga entre disparos
    private bool isReloading = false;

    public int maxHealth = 100;
    private int currentHealth;

    public AudioClip hurtSound;
    public UnityEngine.UI.Slider healthBar;
    public AudioClip deathSound;
    public GameObject deathEffect;
    public TextMeshProUGUI healthText; // UI opcional para mostrar la vida

    public GameObject deathMenuCanvas;

    public string sceneToLoad; // Nombre de la escena a cargar

    void Start()
    {
        if (PlayerPrefs.HasKey("Vida"))
        {
            Debug.Log("Cargando datos del jugador desde PlayerPrefs.");
            currentHealth = PlayerPrefs.GetInt("Vida");
            totalAmmo = PlayerPrefs.GetInt("Municion");
            score = PlayerPrefs.GetInt("Puntuacion");
            medkitCount = PlayerPrefs.GetInt("Botiquines");
            currentAmmo = clipSize;

            UpdateKeyPanel(); // Inicializa el estado del panel

            UpdateUI();
            UpdateHealthUI();
        }
        else
        {
            Debug.Log("No se encontraron datos guardados. Inicializando valores predeterminados.");
            currentHealth = maxHealth;
            currentAmmo = clipSize;
            totalAmmo = maxAmmo - clipSize;
            medkitCount = 0;
            score = 0;

            UpdateKeyPanel(); // Inicializa el estado del panel

            UpdateUI();
            UpdateHealthUI();
        }

        rb = GetComponent<Rigidbody2D>();
        spriteRenderer = GetComponent<SpriteRenderer>();
        animator = GetComponent<Animator>();

        audioSource = GetComponent<AudioSource>();

        stepAudioSource = gameObject.AddComponent<AudioSource>();
        stepAudioSource.clip = stepSound;
        stepAudioSource.loop = true;
        stepAudioSource.playOnAwake = false;
        stepAudioSource.volume = 0.5f;

        if (healthBar != null)
        {
            healthBar.maxValue = maxHealth;
            healthBar.value = currentHealth;
        }

        if (healthText != null)
            healthText.text = $"Vida: {currentHealth}";

        UpdateKeyPanel(); // Inicializa el estado del panel

        UpdateUI();

        LoadPlayerData();
    }


    void Update()
    {
        if (Time.timeScale == 0f) return;

        // Verificar si est  tocando suelo
        isGrounded = Physics2D.OverlapCircle(groundCheck.position,
   groundCheckRadius, groundLayer);

        float move = 0f;
        if (Input.GetKey(KeyCode.D))
            move = 1f;

        else if (Input.GetKey(KeyCode.A))
            move = -1f;

        rb.linearVelocity = new Vector2(move * speed, rb.linearVelocity.
   y);

        // Voltear sprite
        if (move > 0)
            spriteRenderer.flipX = false;

        else if (move < 0)
            spriteRenderer.flipX = true;

        // Animaci n
        animator.SetFloat("Speed", Mathf.Abs(move));

        // Sonido de pasos: solo si moviendo y tocando el suelo
        if (Mathf.Abs(move) > 0.1f && isGrounded)
        {
            if (!stepAudioSource.isPlaying)
                stepAudioSource.Play();
        }

        else
        {
            if (stepAudioSource.isPlaying)
                stepAudioSource.Stop();
        }

        // Saltar
        if (isGrounded && Input.GetKeyDown(KeyCode.Space))
        {
            rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
            PlaySound(jumpSound);
        }

        if (!isReloading && Input.GetKeyDown(KeyCode.F) && Mathf.Abs(rb.linearVelocity.x) < 0.1f && currentAmmo > 0)
        {
            if (Time.time - lastShootTime >= shootCooldown)
            {
                Shoot();
                animator.SetTrigger("Shoot");
                PlaySound(shootSound);
                currentAmmo--;
                lastShootTime = Time.time;

                shotsSinceLastReload++;

                if (shotsSinceLastReload >= clipSize)
                {
                    StartCoroutine(ReloadRoutine());
                }
            }
        }

        if (Input.GetKeyDown(KeyCode.G))
        {
            SavePlayerData();
            Debug.Log("Datos guardados");
        }

        if (ammoText != null)
            ammoText.text = $"{currentAmmo}/{totalAmmo}";

        float minX = -10f;
        float maxX = 10f;

        Vector3 clampedPosition = transform.position;
        clampedPosition.x = Mathf.Clamp(clampedPosition.x, minX, maxX);
        transform.position = clampedPosition;

        if (Input.GetKeyDown(KeyCode.H))
        {
            TakeDamage(10); // Quita 10 de vida al presionar H (solo para pruebas)
        }

        // Recargar manualmente con R
        if (Input.GetKeyDown(KeyCode.R) && !isReloading && currentAmmo < clipSize && totalAmmo > 0)
        {
            StartCoroutine(ReloadRoutine());
        }

        if (Input.GetKeyDown(KeyCode.Tab))
        {
            if (medkitCount > 0 && currentHealth < maxHealth)
            {
                currentHealth = Mathf.Min(currentHealth + 30, maxHealth); // Cura 30 puntos
                medkitCount--;
                UpdateHealthUI();
                UpdateUI();
            }
        }

        // Verificar si el jugador presiona la tecla 'E' para recoger el tem
        if (Input.GetKeyDown(KeyCode.E) && itemNearby != null)
        {
            PickUpItem(itemNearby);  // Llama al m todo para destruir el  tem
        }
    }

    void PickUpItem(GameObject item)
    {
        string tag = item.tag;

        if (tag == "Medkit")
        {
            medkitCount++;
            Destroy(item); // Destruir el  tem  completo
        }
        else if (tag == "Ammo")
        {
            totalAmmo += 2;
            Destroy(item);
        }

    else if (tag == "Key")
    {
        PlayerInventory inventory = GetComponent<PlayerInventory>();
        if (inventory != null)
        {
            inventory.llaves++;
            Destroy(item);
            UpdateKeyPanel(); // Actualiza el estado del panel
        }
    }
        // Reproducir el sonido de recolecci n
        AudioSource.PlayClipAtPoint(pickupSound, transform.position, 3.0
f);

        UpdateUI();
    }

    public void UpdateKeyPanel()
    {
        PlayerInventory inventory = GetComponent<PlayerInventory>();
        if (inventory != null && keyPanel != null)
        {
            keyPanel.SetActive(inventory.llaves > 0); // Muestra el
panel si el jugador tiene al menos una llave
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        Debug.Log($"Colisi n detectada con: {collision.gameObject.name}
");

        if (collision.CompareTag("Key"))
        {
            itemNearby = collision.gameObject; // Asigna la llave como
el  tem  cercano
            pickupPrompt.SetActive(true); // Muestra el mensaje de "
Presiona E para recoger"
            Debug.Log("Llave detectada. Mostrando mensaje de recoger.");
        }
        else if (collision.CompareTag("EnergyBall"))
        {
            TakeDamage(5); // Ajusta el da o que recibe el jugador
            Destroy(collision.gameObject); // Destruye la bola de
energ a
            Debug.Log("El jugador ha recibido da o de la bola de
energ a.");
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        Debug.Log($"Saliendo de colisi n con: {collision.gameObject.
name}");
        if (itemNearby == collision.gameObject)
        {
            itemNearby = null;
            if (pickupPrompt != null)
                pickupPrompt.SetActive(false);
            Debug.Log(" tem  fuera de alcance. Ocultando mensaje.");
        }
    }
    void Awake()
    {
        if (UnityEngine.SceneManagement.SceneManager.GetActiveScene().name == "GameplayScene 1")
        {
            PlayerPrefs.DeleteAll();
            Debug.Log("PlayerPrefs reiniciados al iniciar GameplayScene 1.");
        }

        if (Object.FindObjectsByType<PlayerMovement>(FindObjectsSortMode.None).Length > 1)
        {
            Destroy(gameObject); // evita duplicados
        }
        else
        {
            DontDestroyOnLoad(gameObject); // persiste el jugador entre escenas
            SceneManager.sceneLoaded += OnSceneLoaded;
        }
    }


    void OnDisable()
    {
        Debug.Log("Desuscribiendo OnSceneLoaded del evento SceneManager.sceneLoaded.");
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    public void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (this == null)
        {
            Debug.LogWarning("El objeto PlayerMovement ha sido destruido. Ignorando OnSceneLoaded.");
            return;
        }

        GameObject spawnPoint = GameObject.Find("SpawnPoint");
        if (spawnPoint != null)
        {
            transform.position = spawnPoint.transform.position;
        }
        else
        {
            Debug.LogWarning("SpawnPoint not found in the new scene. Ensure a GameObject named 'SpawnPoint' exists.");
        }

        foreach (SpriteRenderer sr in GetComponentsInChildren<SpriteRenderer>())
        {
            sr.sortingLayerName = "Player";
            sr.sortingOrder = 5;
        }

    UpdateHealthUI();

        if (deathMenuCanvas != null)
            deathMenuCanvas.SetActive(false);

        if (scoreText != null)
        {
            scoreText.gameObject.SetActive(true);
            scoreText.text = $"Puntos: {score}";
        }
        if (rb != null)
        {
            rb.linearVelocity = Vector2.zero;
        }
        isReloading = false; // Reinicia el estado de recarga
        this.enabled = true;
    }

    public void UpdateUI()
    {
        PlayerInventory inventory = GetComponent<PlayerInventory>();
        if (medkitText != null)
            medkitText.text = $"{medkitCount}";

        if (ammoText != null)
            ammoText.text = $"{currentAmmo}/{totalAmmo}";

        if (scoreText != null)
            scoreText.text = $"Puntos: {score}";

        if (keyPanel != null)
            keyPanel.SetActive(inventory.llaves > 0); // Muestra el panel si el jugador tiene al menos una llave
    }

    public void AddScore(int amount, Vector3 worldPosition, string popupText = "")
    {
        score += amount;

        if (scoreText != null)
            scoreText.text = $"Puntos: {score}";

        if (scorePopupPrefab != null)
        {
            GameObject popup = Instantiate(scorePopupPrefab, worldPosition, Quaternion.identity);
            TextMeshPro popupTMP = popup.GetComponentInChildren<TextMeshPro>();
            if (popupTMP != null)
            {
                popupTMP.text = string.IsNullOrEmpty(popupText) ? $"+{amount}" : popupText;
            }

            Destroy(popup, 1.5f);
        }
    }

    public void UpdateHealthUI()
    {
        healthBar.value = currentHealth;
        if (healthText != null)
            healthText.text = $"Vida: {currentHealth}";
    }

    void Shoot()
    {
        float direction = spriteRenderer.flipX ? -1f : 1f;
        Vector3 spawnOffset = new Vector3(0.5f * direction, 0f, 0f);

        Vector3 spawnPosition = firePoint.position + spawnOffset;
        GameObject bullet = Instantiate(bulletPrefab, spawnPosition, Quaternion.identity);

        Rigidbody2D bulletRb = bullet.GetComponent<Rigidbody2D>();
        bulletRb.linearVelocity = new Vector2(direction * bulletSpeed, 0f);

        // Ignorar colisi n con el jugador
        Collider2D bulletCollider = bullet.GetComponent<Collider2D>();
        Collider2D playerCollider = GetComponent<Collider2D>();
        Physics2D.IgnoreCollision(bulletCollider, playerCollider);

        // Disparar el sonido de casquillo despu s de 0.2 segundos
        Invoke(nameof(PlayCasingSound), Random.Range(0.3f, 0.6f));
    }

    void PlayCasingSound()
    {
        PlaySound(casingSound);
    }

    void PlaySound(AudioClip clip)
    {
        if (clip != null)
            AudioSource.PlayClipAtPoint(clip, transform.position);
    }

    void OnDrawGizmosSelected()
    {
        // Para ver el c rculo en el editor
        if (groundCheck != null)
        {
            Gizmos.color = Color.red;
            Gizmos.DrawWireSphere(groundCheck.position, groundCheckRadius);
        }
    }

    private IEnumerator ReloadRoutine()
    {
        isReloading = true;
        animator.SetTrigger("Reload");
        PlaySound(reloadSound);

        yield return new WaitForSeconds(1.5f);

        int needed = clipSize - currentAmmo;

        // Recarga solo la cantidad necesaria y disponible
        int ammoToReload = Mathf.Min(needed, totalAmmo);
        currentAmmo += ammoToReload;
        totalAmmo -= ammoToReload;


        isReloading = false;
        shotsSinceLastReload = 0;
    }

    public void TakeDamage(int damage)
    {
        currentHealth -= damage;
        currentHealth = Mathf.Clamp(currentHealth, 0, maxHealth);
        healthBar.value = currentHealth;

        if (hurtSound != null)
            PlaySound(hurtSound);  //     reproducir sonido de da o

        if (currentHealth <= 0)
        {
            Die();
        }
    }

    public void SavePlayerData()
    {
        PlayerPrefs.SetInt("Municion", totalAmmo);
        PlayerPrefs.SetInt("MunicionActual", currentAmmo);
        PlayerPrefs.SetInt("Puntuacion", score);
        PlayerPrefs.SetInt("Botiquines", medkitCount);
        PlayerPrefs.SetInt("Vida", currentHealth);
        PlayerPrefs.Save();
    }

    void LoadPlayerData()
    {
        if (PlayerPrefs.HasKey("Vida"))
        {
            Debug.Log("Recargando datos desde PlayerPrefs en LoadPlayerData.");
            currentHealth = PlayerPrefs.GetInt("Vida");
            totalAmmo = PlayerPrefs.GetInt("Municion");
            currentAmmo = PlayerPrefs.GetInt("MunicionActual");
            score = PlayerPrefs.GetInt("Puntuacion");
            medkitCount = PlayerPrefs.GetInt("Botiquines");
        }
        else
        {
            Debug.Log("No hay datos en PlayerPrefs. Usando valores predeterminados en LoadPlayerData.");
            currentAmmo = clipSize;
            totalAmmo = maxAmmo - clipSize;
            currentHealth = maxHealth;
            score = 0;
            medkitCount = 0;
        }

        UpdateUI();
        UpdateHealthUI();
    }

    public void ChangeScene(string nextScene)
    {
        SavePlayerData();
        SceneManager.LoadScene(nextScene);
    }

    void CargarEscena()
    {
        PlayerMovement player = Object.FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            player.SavePlayerData();
            SceneManager.sceneLoaded -= player.OnSceneLoaded; // Desuscribirse del evento
            Destroy(player.gameObject); // Destruir el objeto Player
        }

        if (!string.IsNullOrEmpty(sceneToLoad) && Application.CanStreamedLevelBeLoaded(sceneToLoad))
        {
            Debug.Log($"Cargando escena: {sceneToLoad}");
            SceneManager.LoadScene(sceneToLoad);
        }
        else
        {
            Debug.LogError($"La escena '{sceneToLoad}' no existe o no est  incluida en los Build Settings.");
        }
    }

    void Die()
    {
        Debug.Log(" El  jugador ha muerto!");

        if (deathSound != null)
            PlaySound(deathSound);

        if (deathEffect != null)
            Instantiate(deathEffect, transform.position, Quaternion.identity);

        // Activar animaci n y desactivar movimiento
        animator.SetTrigger("Die");
        rb.linearVelocity = Vector2.zero;
        this.enabled = false;

```

```csharp
        StartCoroutine(ShowDeathMenuAfterDelay(0.5f));
    }

    public void ResetPlayerState()
    {
        currentHealth = maxHealth;
        currentAmmo = clipSize;
        totalAmmo = maxAmmo - clipSize;
        medkitCount = 0;
        score = 0;

        // Reiniciar el contador de llaves
        PlayerInventory inventory = GetComponent<PlayerInventory>();
        if (inventory != null)
        {
            inventory.llaves = 0; // Reinicia las llaves a 0
            Debug.Log("El contador de llaves se ha reiniciado.");
        }

        UpdateUI();
        UpdateHealthUI();

        rb.linearVelocity = Vector2.zero;
        this.enabled = true;

        // Reiniciar el Animator
        animator.ResetTrigger("Die");
        animator.Play("swat_idle");

        // Reiniciar el men  de muerte
        if (deathMenuCanvas != null)
        {
            deathMenuCanvas.SetActive(false);
        }
    }

    IEnumerator ShowDeathMenuAfterDelay(float delay)
    {
        yield return new WaitForSeconds(delay);

        if (deathMenuCanvas != null)
        {
            deathMenuCanvas.SetActive(true);
        }
    }

    public int GetCurrentHealth() => currentHealth;
    public void SetCurrentHealth(int value) => currentHealth = value;

    public int GetCurrentAmmo() => currentAmmo;
    public void SetCurrentAmmo(int value) => currentAmmo = value;

    public int GetTotalAmmo() => totalAmmo;
    public void SetTotalAmmo(int value) => totalAmmo = value;

    public int GetMedkitCount() => medkitCount;
    public void SetMedkitCount(int value) => medkitCount = value;

    public int GetScore() => score;
    public void SetScore(int value) => score = value;

}
```

Listing 11: PlayerMovement.cs

12. PlayerInventory.cs

```
using UnityEngine;

public class PlayerInventory : MonoBehaviour
{
    public int llaves = 0;

    public void TomarLlave()
    {
        llaves++;
        Debug.Log("Llave obtenida. Total: " + llaves);
    }

    public bool UsarLlave()
    {
        if (llaves > 0)
        {
            llaves--;
            return true;
        }
        return false;
    }
}
```

Listing 12: PlayerInventory.cs

13. PlayerDoorRaycast.cs

```
  using   UnityEngine;
using TMPro;
using UnityEngine.SceneManagement;

public class PlayerDoorRaycast : MonoBehaviour
{
    public float rayDistance = 2f;
    public LayerMask doorLayer;
    public GameObject mensajeEntrarUI;
    public GameObject mensajeSinLlaveUI;
    public string sceneToLoad;
    public AudioSource audioSource;
    public AudioClip doorOpenSound;

    private bool puedeEntrar = false;
    private bool escenaCargando = false;

    void Update()
    {
        MostrarMensajeEntrar();

        if (Input.GetKeyDown(KeyCode.E) && puedeEntrar && !escenaCargando)
        {
            PlayerInventory inventory = GetComponent<PlayerInventory>();
            if (inventory != null && inventory.llaves > 0)
            {
                escenaCargando = true;
                inventory.UsarLlave();
                audioSource.PlayOneShot(doorOpenSound);
                Invoke("CargarEscena", doorOpenSound.length);
                if (mensajeSinLlaveUI != null)
                    mensajeSinLlaveUI.SetActive(false); // Oculta el mensaje si ten a llave
            }
            else
            {
                Debug.Log("No tienes llaves para abrir esta puerta.");
                if (mensajeSinLlaveUI != null)
                {
                    mensajeSinLlaveUI.SetActive(true);
                    CancelInvoke("OcultarMensajeSinLlave");
                    Invoke("OcultarMensajeSinLlave", 2f); // Oculta el mensaje despu s de 2 segundos
                }
            }
        }
    }

    void OcultarMensajeSinLlave()
    {
        if (mensajeSinLlaveUI != null)
            mensajeSinLlaveUI.SetActive(false);
    }

    void CargarEscena()
    {
        PlayerMovement player = Object.FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            player.SavePlayerData();
            SceneManager.sceneLoaded -= player.OnSceneLoaded;
            Destroy(player.gameObject);
        }

        if (!string.IsNullOrEmpty(sceneToLoad) && Application.CanStreamedLevelBeLoaded(sceneToLoad))
        {
            Debug.Log($"Cargando escena: {sceneToLoad}");
            SceneManager.LoadScene(sceneToLoad);
        }
        else
        {
            Debug.LogError($"La escena '{sceneToLoad}' no existe o no est  incluida en los Build Settings.");
        }
    }

    void MostrarMensajeEntrar()
    {
        Vector2 direccion = transform.right * (GetComponent<SpriteRenderer>().flipX ? -1 : 1);
        Debug.DrawRay((Vector2)transform.position + Vector2.up * 0.5f, direccion * rayDistance, Color.red);
        RaycastHit2D hit = Physics2D.Raycast((Vector2)transform.position + Vector2.up * 0.5f, direccion, rayDistance, doorLayer);

        if (hit.collider != null && hit.collider.CompareTag("Puerta"))
        {
            mensajeEntrarUI.SetActive(true);
            puedeEntrar = true;
        }
        else
        {
            mensajeEntrarUI.SetActive(false);
            puedeEntrar = false;
            if (mensajeSinLlaveUI != null)
                mensajeSinLlaveUI.SetActive(false);
        }
    }

}
```

Listing 13: PlayerDoorRaycast.cs

14. DoorScript.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Door : MonoBehaviour
{
    public string sceneToLoad; // Nombre de la escena a cargar

    private void OnValidate()
    {
        if (string.IsNullOrEmpty(sceneToLoad))
        {
            Debug.LogWarning($"The 'sceneToLoad' field is empty on {gameObject.name}. Assign a scene in the Inspector.");
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            PlayerInventory inv = other.GetComponent<PlayerInventory>();
            if (inv != null && inv.UsarLlave())
            {
                Debug.Log("Puerta abierta. Cambiando de escena...");
                LoadScene();
            }
            else
            {
                Debug.Log("Necesitas una llave para abrir esta puerta.");
            }
        }
    }

    public void LoadScene()
    {
        if (!string.IsNullOrEmpty(sceneToLoad))
        {
            if (Application.CanStreamedLevelBeLoaded(sceneToLoad))
            {
                Debug.Log("Cargando escena: " + sceneToLoad);
                SceneManager.LoadScene(sceneToLoad);
            }
            else
            {
                Debug.LogError($"La escena '{sceneToLoad}' no existe o no est  incluida en Build Settings.");
            }
        }
        else
        {
            Debug.LogError("El campo 'sceneToLoad' en el script Door est  vac o. Asigna una escena en el Inspector.");
        }
    }
}
```

Listing 14: DoorScript.cs

15. KeyPickup.cs

```
using UnityEngine;

public class KeyPickup : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            PlayerInventory inventory = other.GetComponent<PlayerInventory>();
            if (inventory != null)
            {
                // Limpia el mensaje de recoger si existe
                PlayerMovement player = other.GetComponent<PlayerMovement>();
                if (player != null)
                {
                    player.itemNearby = null;
                    if (player.pickupPrompt != null)
                        player.pickupPrompt.SetActive(false);
                }

                inventory.TomarLlave();
                Destroy(gameObject); // Destruye la llave despu s de recogerla
            }
        }
    }
}
```

Listing 15: KeyPickup.cs

16. ItemPickup.cs

```
using UnityEngine;

public class ItemPickup : MonoBehaviour
{
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            PlayerMovement player = other.GetComponentInParent<PlayerMovement>();
            if (player != null)
            {
                player.itemNearby = gameObject;
                player.pickupPrompt?.SetActive(true);
            }
        }
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        Debug.Log($"Saliendo de colisi n con: {collision.gameObject.name}");

        PlayerMovement player = collision.GetComponentInParent<PlayerMovement>();
        if (player != null && player.itemNearby == gameObject)
        {
            player.itemNearby = null;
            if (player.pickupPrompt != null)
                player.pickupPrompt.SetActive(false);
            Debug.Log(" tem  fuera de alcance. Ocultando mensaje.");
        }
    }
}
```

Listing 16: ItemPickup.cs

17. PlayerSystemRaycast.cs

```
using UnityEngine;

public class PlayerSystemRaycast : MonoBehaviour
{
    public float rayDistance = 2f;
    public LayerMask systemLayer;
    public GameObject mensajeInteractuarUI;
    public SystemInteractionMenu systemMenu;

    private bool puedeInteractuar = false;
    private PlayerInventory playerInventory;

    void Start()
    {
        playerInventory = GetComponent<PlayerInventory>();
    }

    void Update()
    {
        MostrarMensajeInteractuar();

        if (Input.GetKeyDown(KeyCode.E) && puedeInteractuar)
        {
            if (systemMenu != null && playerInventory != null && playerInventory.llaves > 0)
            {
                systemMenu.AbrirMenu();
            }
        }
    }

    void MostrarMensajeInteractuar()
    {
        Vector2 direccion = transform.right * (GetComponent<SpriteRenderer>().flipX ? -1 : 1);
        Debug.DrawRay((Vector2)transform.position + Vector2.up * 0.5f, direccion * rayDistance, Color.cyan);
        RaycastHit2D hit = Physics2D.Raycast((Vector2)transform.position + Vector2.up * 0.5f, direccion, rayDistance, systemLayer);

        // Solo muestra el mensaje si el jugador tiene al menos una llave
        if (hit.collider != null && hit.collider.CompareTag("System") && playerInventory != null && playerInventory.llaves > 0)
        {
            mensajeInteractuarUI.SetActive(true);
            puedeInteractuar = true;
            systemMenu = hit.collider.GetComponent<SystemInteractionMenu>();
        }
        else
        {
            mensajeInteractuarUI.SetActive(false);
            puedeInteractuar = false;
            systemMenu = null;
        }
    }
}
```

Listing 17: PlayerSystemRaycast.cs

18. SystemInteractionMenu.cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

```

```csharp
public class SystemInteractionMenu : MonoBehaviour
{
    public GameObject menuUI; // Panel del men  (Canvas)
    public Button opcion1Button;
    public Button opcion2Button;
    public string escenaOpcion1;
    public string escenaOpcion2;

    private bool jugadorCerca = false;

    void Start()
    {
        menuUI.SetActive(false);

    opcion1Button.onClick.AddListener(() => CargarEscenaFinal(escenaOpcion1));
    opcion2Button.onClick.AddListener(() => CargarEscenaFinal(escenaOpcion2));
    }

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            jugadorCerca = true;
        }
    }

    void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            jugadorCerca = false;
            menuUI.SetActive(false);
            Time.timeScale = 1f;
        }
    }

    void CargarEscenaFinal(string nombreEscena)
    {
        // Elimina el jugador y su UI antes de cambiar de escena
        PlayerMovement player = Object.FindFirstObjectByType<PlayerMovement>();
        if (player != null)
        {
            Destroy(player.gameObject);
        }

        Time.timeScale = 1f;
        UnityEngine.SceneManagement.SceneManager.LoadScene(nombreEscena);
    }

    public void AbrirMenu()
    {
        menuUI.SetActive(true);
        Time.timeScale = 0f;
    }
}
```

Listing 18: SystemInteractionMenu.cs

19. PlayerDocumentRaycast.cs

```
using UnityEngine;
using TMPro;

public class PlayerDocumentRaycast : MonoBehaviour
{
    public float rayDistance = 2f;
    public LayerMask documentLayer;
    public GameObject lecturaPanel;
    public TextMeshProUGUI textoUI;
    public TextMeshProUGUI mensajeLecturaUI;


    private bool leyendo = false;
    private DocumentReader documentoActual;

    void Update()
    {
        if (!leyendo)
        {
            MostrarMensajeLectura(); // Nueva funcin para mostrar el
 mensaje si hay un documento
        }

        if (Input.GetKeyDown(KeyCode.Q))
        {
            if (leyendo)
            {
                CerrarLectura();
            }
            else
            {
                DetectarDocumento();
            }
        }
    }


    void DetectarDocumento()
    {
        Vector2 direccion = transform.right * (GetComponent<
 SpriteRenderer>().flipX ? -1 : 1);
        RaycastHit2D hit = Physics2D.Raycast((Vector2)transform.position
  + Vector2.up * 0.5f, direccion, rayDistance, documentLayer);

        if (hit.collider != null && hit.collider.CompareTag("Documento")
 )
        {
            documentoActual = hit.collider.GetComponent<DocumentReader
 >();
            if (documentoActual != null)
            {
                AbrirLectura(documentoActual.GetTexto());
            }
        }
    }

    void AbrirLectura(string texto)
    {
        lecturaPanel.SetActive(true);
        textoUI.text = texto;
        leyendo = true;
        mensajeLecturaUI.gameObject.SetActive(false);
    }

    void MostrarMensajeLectura()
    {
        Vector2 direccion = transform.right * (GetComponent<
   SpriteRenderer>().flipX ? -1 : 1);
        RaycastHit2D hit = Physics2D.Raycast((Vector2)transform.position
    + Vector2.up * 0.5f, direccion, rayDistance, documentLayer);

        if (hit.collider != null && hit.collider.CompareTag("Documento")
   )
        {
            mensajeLecturaUI.gameObject.SetActive(true);
        }
        else
        {
            mensajeLecturaUI.gameObject.SetActive(false);
        }
    }

    void CerrarLectura()
    {
        lecturaPanel.SetActive(false);
        mensajeLecturaUI.gameObject.SetActive(false); // Ocultar mensaje
        leyendo = false;
        documentoActual = null;
    }


}
```

Listing 19: PlayerDocumentRaycast.cs

20. PlayerDocumentInteraction.cs

```
using UnityEngine;
using TMPro;

public class PlayerDocumentInteraction : MonoBehaviour
{
    public GameObject panelLectura;
    public TextMeshProUGUI textoUI;

    private DocumentReader documentoActual;
    private bool leyendo = false;

    void Update()
    {
        if (documentoActual != null && Input.GetKeyDown(KeyCode.E))
        {
            Debug.Log("Presionando E para documento: " + documentoActual.name);
            if (!leyendo)
            {
                textoUI.text = documentoActual.GetTexto();
                panelLectura.SetActive(true);
                leyendo = true;
                Time.timeScale = 0;
            }
            else
            {
                panelLectura.SetActive(false);
                leyendo = false;
                Time.timeScale = 1;
            }
        }
    }

    public void SetDocumento(DocumentReader doc)
    {
        documentoActual = doc;
    }

    public void ClearDocumento()
    {
        documentoActual = null;
    }
}
```

Listing 20: PlayerDocumentInteraction.cs

21. InteractionZone.cs

```
using UnityEngine;

public class InteractionZone : MonoBehaviour
{
    private PlayerDocumentInteraction playerInteraction;

    private void Start()
    {
        playerInteraction = GetComponentInParent<PlayerDocumentInteraction>();
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Documento"))
        {
            Debug.Log("Documento detectado: " + other.name);
            playerInteraction.SetDocumento(other.GetComponent<DocumentReader>());
        }
    }



    void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Documento"))
        {
            playerInteraction.ClearDocumento();
        }
    }
}
```
Listing 21: InteractionZone.cs

22. DocumentReader.cs

```
using UnityEngine;

public class DocumentReader : MonoBehaviour
{
    [TextArea]
    public string textoDocumento;

    public AudioClip sonidoApertura;
    private AudioSource audioSource;

    private void Awake()
    {
        audioSource = gameObject.AddComponent<AudioSource>();
    }

    public string GetTexto()
    {
        if (sonidoApertura != null && audioSource != null)
        {
            audioSource.PlayOneShot(sonidoApertura);
        }
        return textoDocumento;
    }
}
```
Listing 22: DocumentReader.cs

23. BoosFinalAI.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class BossFinalAI : MonoBehaviour
{
    public float speed = 0.8f;
    public float attackCooldown = 2f;
    public float health = 100f;
    public float stunDuration = 2f;

    public float energyBallCooldown = 3f; // Nuevo cooldown para la bola
    de energ a
    private float lastEnergyBallTime = 0f; //  ltima  vez que dispar

    public AudioClip growlSound;
    public AudioClip hitSound;
    public AudioClip deathSound;
    public AudioClip specialAttackSound; // Sonido para el ataque
especial
    public AudioClip normalAttackSound;  // Sonido para el ataque normal

    public GameObject energyBallPrefab;
    public Transform firePoint;

    public Image bossHealthBar; // Referencia a la UI fija

    private Transform player;
    private Animator animator;
    private Rigidbody2D rb;
    private AudioSource audioSource;

    private float lastAttackTime;
    private bool isStunned = false;
    private bool isDead = false;

    [SerializeField] private GameObject keyPrefab;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player")?.transform;
        animator = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
        audioSource = GetComponent<AudioSource>();

        if (bossHealthBar != null)
            bossHealthBar.fillAmount = 2f;

        PlayGrowl();
    }

    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Bala"))
        {
            TakeDamage(50);
            Destroy(collision.gameObject); // Destruye el proyectil
            Debug.Log("El jefe ha recibido da o de la bala del jugador.
");
        }
    }

    void Update()
    {
        if (player == null || isDead) return;

        if (isStunned)
        {
            animator.SetBool("isRunning", false);
            return;
        }

        Vector2 direction = (player.position - transform.position).normalized;
        float distance = Vector2.Distance(transform.position, player.position);

        // Disparo de bola de energ a cada X segundos (independiente del ataque)
        if (Time.time - lastEnergyBallTime >= energyBallCooldown)
        {
            lastEnergyBallTime = Time.time;
            FireEnergyBall();
        }

        // Ataque especial o ataque a distancia cada attackCooldown
        bool isAttacking = Time.time - lastAttackTime < 1f;
        animator.SetBool("isRunning", !isAttacking);

        if (!isAttacking)
        {
            transform.position += (Vector3)direction * speed * Time.deltaTime;

            if (direction.x > 0)
                transform.localScale = new Vector3(1, 1, 1);
            else if (direction.x < 0)
                transform.localScale = new Vector3(-1, 1, 1);
        }

        if (Time.time - lastAttackTime >= attackCooldown)
        {
            if (distance <= 2f)
            {
                animator.SetTrigger("SpecialAttackTrigger");
                PlaySound(specialAttackSound);
                PerformSpecialAttack();
            }
            else
            {
                animator.SetTrigger("AttackTrigger");
                PlaySound(normalAttackSound);
            }
            lastAttackTime = Time.time;
        }
    }


    IEnumerator DelayedFire(float delay)
    {
        yield return new WaitForSeconds(delay);
        FireEnergyBall();
    }

    void PerformSpecialAttack()
    {
        float specialAttackRange = 2f; // Rango del ataque especial
        int specialAttackDamage = 5; // Da o del ataque especial

        Collider2D[] hitColliders = Physics2D.OverlapCircleAll(transform
   .position, specialAttackRange);
        foreach (Collider2D collider in hitColliders)
        {
            if (collider.CompareTag("Player"))
            {
                PlayerMovement player = collider.GetComponent<
   PlayerMovement>();
                if (player != null)
                {
                    player.TakeDamage(specialAttackDamage);
                    Debug.Log("El jugador ha recibido da o del ataque
   especial del jefe.");
                }
            }
        }
    }

    public void FireEnergyBall()
    {
        if (energyBallPrefab != null && firePoint != null)
        {
            GameObject ball = Instantiate(energyBallPrefab, firePoint.
   position, Quaternion.identity);
            Vector2 dir = (player.position - firePoint.position).
   normalized;
            ball.GetComponent<Rigidbody2D>().linearVelocity = dir * 2f;
        }
    }

    public void TakeDamage(float damage)
    {
        if (isDead) return;

        health -= damage;
        PlaySound(hitSound);

        if (bossHealthBar != null)
            bossHealthBar.fillAmount = health / 500f;

        if (health <= 0)
        {
            Die();
        }
        else if (!isStunned)
        {
            isStunned = true;
            Invoke("ResetStun", stunDuration);
        }
    }

    void Die()
    {
        isDead = true;
        animator.SetTrigger("DieTrigger");
        PlaySound(deathSound);

        DropKey();

        Destroy(gameObject, 1f);
        if (bossHealthBar != null)
            bossHealthBar.transform.parent.gameObject.SetActive(false);
    }

    void ResetStun()
    {
        isStunned = false;
    }

    void DropKey()
    {
        if (keyPrefab != null && GameObject.FindGameObjectWithTag("Key") == null)
        {
            Instantiate(keyPrefab, transform.position, Quaternion.identity);
            Debug.Log("Llave soltada por el Boss.");
        }
    }

    void PlaySound(AudioClip clip)
    {
        if (audioSource != null && clip != null)
        {
            audioSource.PlayOneShot(clip);
        }
    }

    void PlayGrowl()
    {
        if (audioSource != null && growlSound != null)
        {
            audioSource.loop = true;
            audioSource.clip = growlSound;
            audioSource.Play();
        }
    }
}
```

Listing 23: BoosFinalAI.cs

24. BulletScript.cs

```
using UnityEngine;

public class Bullet : MonoBehaviour
{
    public GameObject explosionEffect;
    public float destroyDelay = 0.1f;
    public float damage = 50f;

    void OnTriggerEnter2D(Collider2D collision)
    {
        // Filtrar: si no es Zombie ni Ground, ignorar completamente
        if (!collision.CompareTag("Zombie") && !collision.CompareTag("Ground"))
        {
            return;
        }

        Debug.Log("C o l i s i n  REAL con: " + collision.name + " | Tag: " + collision.tag);

        // Instanciar la e x p l o s i n
        if (explosionEffect != null)
        {
            GameObject explosion = Instantiate(explosionEffect, transform.position, Quaternion.identity);
            explosion.transform.localScale = new Vector3(4f, 4f, 1f);
            Destroy(explosion, 0.3f);
        }

        //  D a o  al zombie si aplica
        if (collision.CompareTag("Zombie"))
        {
            ZombieAI zombie = collision.GetComponent<ZombieAI>();
            if (zombie != null)
            {
                zombie.TakeDamage(damage);
            }
        }

        Destroy(gameObject);
    }
}
```

Listing 24: BulletScript.cs

25. FloatingText.cs

```
using UnityEngine;
using TMPro;

public class FloatingText : MonoBehaviour
{
    public float floatSpeed = 1f;
    public float lifetime = 1.2f;
    public Vector3 floatDirection = Vector3.up;
    public TextMeshPro text;

    private Color originalColor;
    private float timer;

    void Start()
    {
        if (text == null)
            text = GetComponent<TextMeshPro>();

        originalColor = text.color;
    }

    void Update()
    {
        transform.position += floatDirection * floatSpeed * Time.deltaTime;

        timer += Time.deltaTime;
        float alpha = Mathf.Lerp(originalColor.a, 0, timer / lifetime);
        text.color = new Color(originalColor.r, originalColor.g, originalColor.b, alpha);

        if (timer >= lifetime)
            Destroy(gameObject);
    }

    public void SetText(string content, Color color)
    {
        if (text == null)
            text = GetComponent<TextMeshPro>();

        text.text = content;
        text.color = color;
        originalColor = color;
    }
}
```

Listing 25: FloatingText.cs

26. CameraFollow.cs

```
using UnityEngine;

public class CameraFollow : MonoBehaviour
{
    public Transform target;       // El jugador
    public Vector3 offset;         // Para ajustar la p o s i c i n de la c mara

    public Vector2 minLimits;      //  L mite  inferior izquierdo del mapa
    public Vector2 maxLimits;      //  L mite  superior derecho del mapa

    void LateUpdate()
    {
        // Si el target es nulo, buscar al jugador nuevamente
        if (target == null)
        {
    #if UNITY_2023_1_OR_NEWER
            PlayerMovement foundPlayer = Object.FindFirstObjectByType<PlayerMovement>();
    #else
            PlayerMovement foundPlayer = FindObjectOfType<PlayerMovement>();
    #endif
            if (foundPlayer != null)
            {
                target = foundPlayer.transform;
            }
            else
            {
                return; // Si no se encuentra el jugador, salir del m todo
            }
        }

        // Calcular la nueva posici n deseada con el offset
        Vector3 desiredPosition = target.position + offset;

        // Limitar el movimiento de la c mara dentro de los bordes definidos
        float clampedX = Mathf.Clamp(desiredPosition.x, minLimits.x, maxLimits.x);
        float clampedY = Mathf.Clamp(desiredPosition.y, minLimits.y, maxLimits.y);

        // Establecer la nueva posici n de la c mara
        transform.position = new Vector3(clampedX, clampedY, transform.position.z);
    }
}
```

Listing 26: CameraFollow.cs

27. GameManager.cs

```
using UnityEngine;

public static class GameManager
{
    public static int municion = 0;
    public static int puntuacion = 0;
    public static int botiquines = 0;
}
```

Listing 27: GameManager.cs

28. PlayerData.cs

```
  using   UnityEngine;

public static class PlayerData
{
    public static int health;
    public static int currentAmmo;
    public static int totalAmmo;
    public static int medkitCount;
    public static int score;

    public static void SaveData(PlayerMovement player)
    {
        health = player.GetCurrentHealth();
        currentAmmo = player.GetCurrentAmmo();
        totalAmmo = player.GetTotalAmmo();
        medkitCount = player.GetMedkitCount();
        score = player.GetScore();
    }

    public static void LoadData(PlayerMovement player)
    {
        player.SetCurrentHealth(health);
        player.SetCurrentAmmo(currentAmmo);
        player.SetTotalAmmo(totalAmmo);
        player.SetMedkitCount(medkitCount);
        player.SetScore(score);

        // Update UI after loading data
        player.UpdateUI();
        player.UpdateHealthUI();
    }
}
```

Listing 28: PlayerData.cs

29. DialogoManager.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using TMPro;

public class DialogoManager : MonoBehaviour
{
    public GameObject panelDialogo;
    public TMP_Text textoDialogo;
    public string[] lineas;
    private int indice;


    void Start()
    {
        panelDialogo.SetActive(false);
    }

    public void ActivarDialogo(string[] nuevoDialogo)
    {
        lineas = nuevoDialogo;
        indice = 0;
        panelDialogo.SetActive(true);
        StartCoroutine(MostrarTexto());
    }

    IEnumerator MostrarTexto()
    {
        textoDialogo.text = "";
        foreach (char letra in lineas[indice].ToCharArray())
        {
            textoDialogo.text += letra;
            yield return new WaitForSeconds(0.03f); // efecto  m quina de escribir
        }
    }

    void Update()
    {
        if (panelDialogo.activeSelf && Input.GetKeyDown(KeyCode.E))
        {
            indice++;
            if (indice < lineas.Length)
            {
                StartCoroutine(MostrarTexto());
            }
            else
            {
                panelDialogo.SetActive(false);
            }
        }
    }

}
```
Listing 29: DialogoManager.cs

30. MissingScripts.cs

```
using UnityEngine;
using UnityEditor;

public class MissingScriptFinder : MonoBehaviour
{
#if UNITY_EDITOR
    [MenuItem("Tools/Find Missing Scripts")]
    public static void FindMissingScripts()
    {
        GameObject[] go = Object.FindObjectsByType<GameObject>(FindObjectsSortMode.None);
        int goCount = 0, componentsCount = 0, missingCount = 0;

        foreach (GameObject g in go)
        {
            goCount++;
            Component[] components = g.GetComponents<Component>();
            for (int i = 0; i < components.Length; i++)
            {
                componentsCount++;
                if (components[i] == null)
                {
                    missingCount++;
                    Debug.Log($"GameObject ’{g.name}’ in scene has a missing script!", g);
                }
            }
        }

        Debug.Log($"Searched {goCount} GameObjects, {componentsCount} components, found {missingCount} missing.");
    }
#endif
}
```
Listing 30: MissingScripts.cs