Anexo: Código Fuente del Proyecto Bomberman 3D

Este anexo contiene el código fuente de los scripts desarrollados para el proyecto Bomberman 3D.

Índice

Scripts de Jugador

Player.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\Player.cs
using UnityEngine;
using UnityEngine.UI;
using System.Collections;

public class Player : MonoBehaviour
{
    [Range(1, 2)]
    public int playerNumber = 1;

    public float moveSpeed = 5f;
    public bool canDropBombs = true;
    public bool canMove = true;
    public bool dead = false;

    public int lives = 3;

    private Rigidbody rigidBody;
    private Transform myTransform;
    private Animator animator;
    public GameObject bombPrefab;

    // Array de posiciones de respawn para evitar reaparecer siempre en el mismo
sitio
    public Transform[] spawnPoints;

    // Referencia al UI Manager para actualizar vidas en pantalla
    public LivesUIManager livesUIManager;

    void Start()
    {
        rigidBody = GetComponent<Rigidbody>();
```

```
        myTransform = transform;
        animator = myTransform.Find("PlayerModel").GetComponent<Animator>();

        // Inicializo UI después de un ratito para asegurar que LivesUIManager
esté listo
        StartCoroutine(InitializeUI());
    }

    private IEnumerator InitializeUI()
    {
        // Espero un poco más cuando estamos en modo campaña para darle tiempo a
todo a inicializarse
        if (GlobalStateManager.Instance != null)
        {
            Debug.Log("Player " + playerNumber + ": GlobalStateManager encontrado,
modo campaña: " + GlobalStateManager.Instance.isCampaignMode);
            if (GlobalStateManager.Instance.isCampaignMode)
            {
                Debug.Log("Player " + playerNumber + ": Esperando delay adicional
para modo campaña");
                yield return new WaitForSeconds(0.3f); // Delay más largo para
modo campaña
            }
        }
        else
        {
            Debug.LogWarning("Player " + playerNumber + ": GlobalStateManager es
nulo, no se puede verificar modo de juego");
            // Espero un poco de todos modos por si acaso
            yield return new WaitForSeconds(0.1f);
        }

        // Espero un frame para que todos los objetos se inicialicen
        yield return new WaitForEndOfFrame();

        // Busco LivesUIManager si no tengo referencia
        if (livesUIManager == null)
        {
            // Intento obtener la instancia singleton primero
            livesUIManager = LivesUIManager.Instance;

            // Si aún no existe, busco por nombre específico según la imagen de la
interfaz
            if (livesUIManager == null)
            {
                GameObject livesUIObj = GameObject.Find("Lives UI Manager");
                if (livesUIObj != null)
                {
                    Debug.Log("Player " + playerNumber + ": Encontrado objeto
'Lives UI Manager' directamente");
                    livesUIManager = livesUIObj.GetComponent<LivesUIManager>();
                }
            }
```

```
            // Si aún no existe, busco en la escena de forma genérica
            if (livesUIManager == null)
            {
                livesUIManager = FindObjectOfType<LivesUIManager>();

                // Si aún no se encuentra, creo uno nuevo
                if (livesUIManager == null)
                {
                    Debug.Log("Player " + playerNumber + ": Creando un nuevo
LivesUIManager");
                    GameObject livesUIObj = new GameObject("LivesUIManager");
                    livesUIManager = livesUIObj.AddComponent<LivesUIManager>();

                    // Busco las imágenes de corazones en la escena
                    GameObject player1UI = GameObject.Find("Player1UI");
                    GameObject player2UI = GameObject.Find("Player2UI");

                    if (player1UI != null)
                    {
                        // Activo el UI
                        player1UI.SetActive(true);
                        // Intento con Player1Lives primero
                        Transform heartsParent =
player1UI.transform.Find("Player1Lives");

                        // Si no existe, intento con Hearts (nombre alternativo)
                        if (heartsParent == null) {
                            heartsParent = player1UI.transform.Find("Hearts");
                            if (heartsParent != null) {
                                Debug.Log("Usando fallback 'Hearts' para Player
1");
                            }
                        }

                        if (heartsParent != null)
                        {
                            Image[] heartsPlayer1 =
heartsParent.GetComponentsInChildren<Image>(true);
                            livesUIManager.player1Lives = heartsPlayer1;
                            Debug.Log("Encontrados " + heartsPlayer1.Length + "
corazones para Player 1");
                        }
                        else
                        {
                            Debug.LogWarning("No se encontró Player1Lives ni
Hearts dentro de Player1UI");
                            // Listo los hijos para depuración
                            for (int i = 0; i < player1UI.transform.childCount;
i++) {
                                Debug.Log("Player1UI child " + i + ": " +
player1UI.transform.GetChild(i).name);
                            }
                        }
                    }
```

```
                    if (player2UI != null)
                    {
                        // Activar el UI
                        player2UI.SetActive(true);
                        // Intentar con Player2Lives primero
                        Transform heartsParent =
player2UI.transform.Find("Player2Lives");

                        // Si no existe, intentar con Hearts (nombre alternativo)
                        if (heartsParent == null) {
                            heartsParent = player2UI.transform.Find("Hearts");
                            if (heartsParent != null) {
                                Debug.Log("Usando fallback 'Hearts' para Player
2");
                            }
                        }

                        if (heartsParent != null)
                        {
                            Image[] heartsPlayer2 =
heartsParent.GetComponentsInChildren<Image>(true);
                            livesUIManager.player2Lives = heartsPlayer2;
                            Debug.Log("Encontrados " + heartsPlayer2.Length + "
corazones para Player 2");
                        }
                        else
                        {
                            Debug.LogWarning("No se encontró Player2Lives ni
Hearts dentro de Player2UI");
                            // Listar los hijos para depuración
                            for (int i = 0; i < player2UI.transform.childCount;
i++) {
                                Debug.Log("Player2UI child " + i + ": " +
player2UI.transform.GetChild(i).name);
                            }
                        }
                    }
                }
            }

            if (livesUIManager == null)
            {
                Debug.LogError("Player " + playerNumber + ": No se pudo
crear/encontrar LivesUIManager en la escena");
                yield break;
            }
        }

        // Me aseguro de que los GameObjects de UI estén activos antes de
actualizar
        string uiName = "Player" + playerNumber + "UI";
        GameObject playerUI = GameObject.Find(uiName);
        if (playerUI != null && !playerUI.activeSelf)
```

```
        {
            Debug.Log("Activando " + uiName + " que estaba desactivado");
            playerUI.SetActive(true);
        }

        // Me aseguro de que tengo las imágenes antes de actualizar la UI
        if ((playerNumber == 1 && (livesUIManager.player1Lives == null ||
livesUIManager.player1Lives.Length == 0)) ||
            (playerNumber == 2 && (livesUIManager.player2Lives == null ||
livesUIManager.player2Lives.Length == 0)))
        {
            Debug.LogWarning("Player " + playerNumber + ": Forzando busqueda de
imágenes antes de actualizar UI");
            // Doy una última oportunidad para encontrar las imágenes
            GameObject livesUIObj = GameObject.Find("Lives UI Manager");
            if (livesUIObj != null)
            {
                Debug.Log("Player " + playerNumber + ": Encontrado Lives UI
Manager en la escena");
                // Si ya existe en la escena, uso esa instancia
                LivesUIManager existingManager =
livesUIObj.GetComponent<LivesUIManager>();
                if (existingManager != null && existingManager != livesUIManager)
                {
                    Debug.Log("Player " + playerNumber + ": Reemplazando
referencia al LivesUIManager existente");
                    livesUIManager = existingManager;
                }
            }
        }

        // Actualizo UI con las vidas completas
        livesUIManager.UpdateLivesUI(playerNumber, lives);
        Debug.Log("Player " + playerNumber + ": UI inicializada con " + lives + "
vidas");
    }

    void Update()
    {
        UpdateMovement();
    }

    private void UpdateMovement()
    {
        animator.SetBool("Walking", false);

        if (!canMove || dead)
            return;

        if (playerNumber == 1)
            UpdatePlayer1Movement();
        else
            UpdatePlayer2Movement();
    }
```

```
    private void UpdatePlayer1Movement()
    {
        if (Input.GetKey(KeyCode.W))
        {
            rigidBody.velocity = new Vector3(0, rigidBody.velocity.y, moveSpeed);
            myTransform.rotation = Quaternion.Euler(0, 0, 0);
            animator.SetBool("Walking", true);
        }
        else if (Input.GetKey(KeyCode.A))
        {
            rigidBody.velocity = new Vector3(-moveSpeed, rigidBody.velocity.y, 0);
            myTransform.rotation = Quaternion.Euler(0, 270, 0);
            animator.SetBool("Walking", true);
        }
        else if (Input.GetKey(KeyCode.S))
        {
            rigidBody.velocity = new Vector3(0, rigidBody.velocity.y, -moveSpeed);
            myTransform.rotation = Quaternion.Euler(0, 180, 0);
            animator.SetBool("Walking", true);
        }
        else if (Input.GetKey(KeyCode.D))
        {
            rigidBody.velocity = new Vector3(moveSpeed, rigidBody.velocity.y, 0);
            myTransform.rotation = Quaternion.Euler(0, 90, 0);
            animator.SetBool("Walking", true);
        }
        else
        {
            rigidBody.velocity = new Vector3(0, rigidBody.velocity.y, 0);
        }

        if (canDropBombs && Input.GetKeyDown(KeyCode.Space))
            DropBomb();
    }

    private void UpdatePlayer2Movement()
    {
        if (Input.GetKey(KeyCode.UpArrow))
        {
            rigidBody.velocity = new Vector3(0, rigidBody.velocity.y, moveSpeed);
            myTransform.rotation = Quaternion.Euler(0, 0, 0);
            animator.SetBool("Walking", true);
        }
        else if (Input.GetKey(KeyCode.LeftArrow))
        {
            rigidBody.velocity = new Vector3(-moveSpeed, rigidBody.velocity.y, 0);
            myTransform.rotation = Quaternion.Euler(0, 270, 0);
            animator.SetBool("Walking", true);
        }
        else if (Input.GetKey(KeyCode.DownArrow))
        {
            rigidBody.velocity = new Vector3(0, rigidBody.velocity.y, -moveSpeed);
            myTransform.rotation = Quaternion.Euler(0, 180, 0);
```

```
            animator.SetBool("Walking", true);
        }
        else if (Input.GetKey(KeyCode.RightArrow))
        {
            rigidBody.velocity = new Vector3(moveSpeed, rigidBody.velocity.y, 0);
            myTransform.rotation = Quaternion.Euler(0, 90, 0);
            animator.SetBool("Walking", true);
        }
        else
        {
            rigidBody.velocity = new Vector3(0, rigidBody.velocity.y, 0);
        }

        if (canDropBombs && (Input.GetKeyDown(KeyCode.KeypadEnter) ||
Input.GetKeyDown(KeyCode.Return)))
            DropBomb();
    }

    private void DropBomb()
    {
        if (bombPrefab)
        {
            Instantiate(
                bombPrefab,
                new Vector3(Mathf.RoundToInt(myTransform.position.x),
bombPrefab.transform.position.y, Mathf.RoundToInt(myTransform.position.z)),
                bombPrefab.transform.rotation
            );
        }
    }

    public void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Explosion") && !dead)
        {
            Debug.Log("P" + playerNumber + " hit by explosion!");
            LoseLife();
        }
    }

    private void LoseLife()
    {
        if (dead) return;

        lives--;
        Debug.Log("P" + playerNumber + " lost a life. Remaining lives: " + lives);

        // Actualizo la UI cada vez que pierde vida
        if (livesUIManager != null)
        {
            livesUIManager.UpdateLivesUI(playerNumber, lives);
        }
        else
        {
```

```
            // Si por alguna razón no tengo referencia, intento obtenerla de nuevo
            livesUIManager = LivesUIManager.Instance;
            if (livesUIManager == null)
            {
                // Último intento: busco en la escena
                livesUIManager = FindObjectOfType<LivesUIManager>();
            }

            if (livesUIManager != null)
            {
                livesUIManager.UpdateLivesUI(playerNumber, lives);
                Debug.Log("P" + playerNumber + ": Recuperada referencia a
LivesUIManager");
            }
            else
            {
                Debug.LogError("P" + playerNumber + ": No se pudo actualizar UI de
vidas, LivesUIManager no encontrado");
            }
        }

        if (lives <= 0)
        {
            dead = true;
            canMove = false;
            rigidBody.velocity = Vector3.zero;
            Debug.Log("P" + playerNumber + " has no lives left!");

            // Mejor no desactivo GameObject aquí
            if (GlobalStateManager.Instance != null)
                GlobalStateManager.Instance.PlayerDied(playerNumber);
        }
        else
        {
            StartCoroutine(RespawnPlayer());
        }
    }

    private IEnumerator RespawnPlayer()
    {
        canMove = false;
        animator.SetBool("Walking", false);
        rigidBody.velocity = Vector3.zero;

        SetPlayerVisible(false);

        yield return new WaitForSeconds(1.5f);

        if (spawnPoints != null && spawnPoints.Length > 0)
        {
            int index = Random.Range(0, spawnPoints.Length);
            transform.position = spawnPoints[index].position;
        }
        else
```

```
        {
            transform.position = new Vector3(0, transform.position.y, 0);
        }

        dead = false;  // Lo pongo aquí, justo antes de reaparecer
        canMove = true;
        SetPlayerVisible(true);
    }

    private void SetPlayerVisible(bool visible)
    {
        var renderer = GetComponentInChildren<Renderer>();
        if (renderer != null)
            renderer.enabled = visible;

        var collider = GetComponent<Collider>();
        if (collider != null)
            collider.enabled = visible;
    }
}
```

Scripts de Bombas y Explosiones

Bomb.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\Bomb.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Bomb : MonoBehaviour {

    public GameObject explosionPrefab;
    public LayerMask levelMask;
    private bool exploded = false;



    // Inicialización
    void Start () {
        Invoke("Explode", 3f);
    }

    // Se llama cada frame
    void Update () {

    }

    void Explode()
    {
```

```
        Instantiate(explosionPrefab, transform.position, Quaternion.identity); //
Explosión central

        StartCoroutine(CreateExplosions(Vector3.forward));
        StartCoroutine(CreateExplosions(Vector3.right));
        StartCoroutine(CreateExplosions(Vector3.back));
        StartCoroutine(CreateExplosions(Vector3.left));

        GetComponent<MeshRenderer>().enabled = false;
        exploded = true;
        transform.Find("Collider").gameObject.SetActive(false); // Desactivo el
collider
        Destroy(gameObject, .3f); // Destruyo la bomba tras un ratito


    }

    private IEnumerator CreateExplosions(Vector3 direction)
    {
        // Creo explosiones en la dirección dada
for (int i = 1; i < 3; i++)
  {
  // Compruebo si hay algo que bloquee
  RaycastHit hit;
  // Lanzo un raycast para detectar obstáculos
  Physics.Raycast(transform.position + new Vector3(0,.5f,0), direction, out hit,
    i, levelMask);

  // Si no hay obstáculo, pongo la explosión
  if (!hit.collider)
  {
    Instantiate(explosionPrefab, transform.position + (i * direction),
    // Uso la rotación del prefab
      explosionPrefab.transform.rotation);
    // Continúo con la siguiente posición
  }
  else
  { // Si hay obstáculo, paro aquí
    break;
  }

  // Espero un poquito antes de la siguiente explosión
  yield return new WaitForSeconds(.05f);
}

    }

    public void OnTriggerEnter(Collider other)
{
    if (!exploded && other.CompareTag("Explosion"))
    { // Si no he explotado ya y me toca una explosión
  CancelInvoke("Explode"); // Cancelo la explosión programada
  Explode(); // Exploto inmediatamente por reacción en cadena
    }
```

```
}



}
```

DestroySelf.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\DestroySelf.cs
using UnityEngine;
using System.Collections;

/// <summary>
/// Script sencillo para destruir un objeto después de un rato
/// </summary>
public class DestroySelf : MonoBehaviour
{
    public float Delay = 3f;
    // Tiempo en segundos antes de destruir el objeto

    void Start ()
    {
        Destroy (gameObject, Delay);
    }
}
```

Scripts de UI y Menús

infoScenne.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\infoScenne.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class infoScenne : MonoBehaviour {

    public void CargarInfoScene()
    {
        SceneManager.LoadScene("infoScenne");

    }

    }
```

HideGameUIOnMenu.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\HideGameUIOnMenu.c
s
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class HideGameUIOnMenu : MonoBehaviour
{
    // Nombres de UI que busco - varias opciones posibles
    private string[] uiNamesToFind = { "Player1UI", "Player2UI", "Player 1 Lives",
"Player 2 Lives", "Lives" };
    private string[] menuScenes = { "MainMenu", "LevelSelector", "WinnerScene" };

    // Esto hace que el script persista entre escenas y siempre gestione la UI
    private static HideGameUIOnMenu instance;
    void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else if (instance != this)
        {
            Destroy(gameObject);
            return;
        }
    }

    void Start()
    {
        string currentScene = SceneManager.GetActiveScene().name;
        bool isMenuScene = System.Array.Exists(menuScenes, scene => scene ==
currentScene);

        Debug.Log("HideGameUIOnMenu Start: Scene=" + currentScene + ", IsMenu=" +
isMenuScene);

        if (isMenuScene)
        {
            // En escenas de menú, oculto inmediatamente las UI de juego
            HideGameUI();
        }
        else
        {
            // En escenas de juego, muestro inmediatamente las UI
            ShowGameUI();
        }
    }
```

```
    private void HideGameUI()
    {
        Debug.Log("HideGameUI: Ocultando UI de juego...");

        GameObject gsm = GameObject.Find("Global State Manager");
        if (gsm != null)
        {
            Transform canvas = gsm.transform.Find("Canvas");
            if (canvas != null)
            {
                Transform p1 = canvas.Find("Player1UI");
                if (p1 != null) p1.gameObject.SetActive(false);
                Transform p2 = canvas.Find("Player2UI");
                if (p2 != null) p2.gameObject.SetActive(false);
                Debug.Log("HideGameUIOnMenu: Ocultando Player1UI y Player2UI del
Canvas de GSM");
            }
        }
    }

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

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

    void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        string currentScene = scene.name;
        bool isMenuScene = System.Array.Exists(menuScenes, s => s ==
currentScene);
        Debug.Log("HideGameUIOnMenu OnSceneLoaded: Scene=" + currentScene + ",
IsMenu=" + isMenuScene);

        if (isMenuScene)
        {
            // Un poquito de delay para asegurar que los objetos estén cargados
            StartCoroutine(HideGameUIDelayed());
        }
        else
        {
            // En escenas de juego, uso varios intentos para mostrar la UI
            StartCoroutine(ShowGameUIWithRetry());
        }
    }

    private IEnumerator HideGameUIDelayed()
    {
        yield return new WaitForSeconds(0.1f);
```

```
            HideGameUI();
        }

        private IEnumerator ShowGameUIWithRetry()
        {
            // Primer intento inmediato
            ShowGameUI();

            // Segundo intento después de medio segundo
            yield return new WaitForSeconds(0.5f);
            Debug.Log("ShowGameUIWithRetry: Segundo intento después de 0.5s");
            ShowGameUI();

            // Tercer intento después de 1 segundo total
            yield return new WaitForSeconds(0.5f);
            Debug.Log("ShowGameUIWithRetry: Tercer intento después de 1s");
            ShowGameUI();
        }

        private void ShowGameUI()
        {
            Debug.Log("ShowGameUI: Mostrando UI de juego...");

            GameObject gsm = GameObject.Find("Global State Manager");
            if (gsm != null)
            {
                Transform canvas = gsm.transform.Find("Canvas");
                if (canvas != null)
                {
                    Transform p1 = canvas.Find("Player1UI");
                    if (p1 != null) p1.gameObject.SetActive(true);
                    Transform p2 = canvas.Find("Player2UI");
                    if (p2 != null) p2.gameObject.SetActive(true);
                    Debug.Log("ShowGameUI: Mostrando Player1UI y Player2UI del Canvas
de GSM");
                }
            }
        }
    }
```

LevelSelectorManager.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\LevelSelectorManag
er.cs
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LevelSelectorManager : MonoBehaviour
{
```

```
    public Button level1Button;
    public Button level2Button;
    public Button level3Button;
    public Button backButton;

    void Start()
    {
        // Busco y configuro los botones si no están asignados
        if (level1Button == null)
        {
            GameObject lvl1Obj = GameObject.Find("NIVEL 1");
            if (lvl1Obj != null)
                level1Button = lvl1Obj.GetComponent<Button>();
        }

        if (level2Button == null)
        {
            GameObject lvl2Obj = GameObject.Find("NIVEL 2");
            if (lvl2Obj != null)
                level2Button = lvl2Obj.GetComponent<Button>();
        }

        if (level3Button == null)
        {
            GameObject lvl3Obj = GameObject.Find("NIVEL 3");
            if (lvl3Obj != null)
                level3Button = lvl3Obj.GetComponent<Button>();
        }

        if (backButton == null)
        {
            GameObject backObj = GameObject.Find("VOLVER");
            if (backObj != null)
                backButton = backObj.GetComponent<Button>();
        }

        // Añado los listeners a los botones
        if (level1Button != null)
            level1Button.onClick.AddListener(LoadLevel1);
        else
            Debug.LogError("Botón de Nivel 1 no encontrado!");

        if (level2Button != null)
            level2Button.onClick.AddListener(LoadLevel2);
        else
            Debug.LogError("Botón de Nivel 2 no encontrado!");

        if (level3Button != null)
            level3Button.onClick.AddListener(LoadLevel3);
        else
            Debug.LogError("Botón de Nivel 3 no encontrado!");

        if (backButton != null)
            backButton.onClick.AddListener(BackToMenu);
```

```
        else
            Debug.LogError("Botón de Volver no encontrado!");
    }
    public void LoadLevel1()
    {
        if (GlobalStateManager.Instance != null)
        {
            // No cambio el modo de juego - mantengo el que se estableció desde
MainMenu
            // El FastGame ya configuró isCampaignMode = false
            GlobalStateManager.Instance.ResetMatch();
        }
        SceneManager.LoadScene("Game");
    }

    public void LoadLevel2()
    {
        if (GlobalStateManager.Instance != null)
        {
            // No cambio el modo de juego - mantengo el que se estableció desde
MainMenu
            GlobalStateManager.Instance.ResetMatch();
        }
        SceneManager.LoadScene("Game 1");
    }

    public void LoadLevel3()
    {
        if (GlobalStateManager.Instance != null)
        {
            // No cambio el modo de juego - mantengo el que se estableció desde
MainMenu
            GlobalStateManager.Instance.ResetMatch();
        }
        SceneManager.LoadScene("Game 2");
    }

    public void BackToMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```

LivesUIManager.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\LivesUIManager.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

```
using UnityEngine.SceneManagement;

public class LivesUIManager : MonoBehaviour
{
    public static LivesUIManager Instance;

    [Tooltip("Corazones que representan las vidas del jugador 1")]
    public Image[] player1Lives;

    [Tooltip("Corazones que representan las vidas del jugador 2")]
    public Image[] player2Lives;

    private void Awake()
    {
        // Singleton para acceso global
        if (Instance == null)
        {
            Instance = this;

            // Evito que sea destruido al cambiar de escena
            DontDestroyOnLoad(gameObject);

            // Verifico si somos el LivesUIManager de la interfaz de Unity
            CheckIfUILivesManager();

            // Busco los componentes de UI inmediatamente
            EnsureUIComponentsExist();
        }
        else if (Instance != this)
        {
            Debug.Log("Ya existe un LivesUIManager. Destruyendo duplicado.");
            Destroy(gameObject);
            return;
        }

        // Me apunto para eventos de cambio de escena
        SceneManager.sceneLoaded += OnSceneLoaded;

        Debug.Log("LivesUIManager inicializado como singleton");
    }

    // Verifico si este componente es el LivesUIManager de la interfaz de Unity
    private void CheckIfUILivesManager()
    {
        // Compruebo si este objeto tiene el nombre esperado en la interfaz
        if (gameObject.name == "Lives UI Manager")
        {
            Debug.Log("Este LivesUIManager es el componente de UI del editor");

            // Busco si ya tiene asignadas las imágenes de corazones en el
inspector
            if (player1Lives == null || player1Lives.Length == 0)
            {
                Transform player1LivesTransform = transform.Find("Player 1
```

```
Lives");
                if (player1LivesTransform != null)
                {
                    Image[] hearts =
player1LivesTransform.GetComponentsInChildren<Image>(true);
                    if (hearts != null && hearts.Length > 0)
                    {
                        player1Lives = hearts;
                        Debug.Log("Encontradas " + hearts.Length + " imágenes para
Player 1 Lives en el inspector");
                    }
                }
            }

            if (player2Lives == null || player2Lives.Length == 0)
            {
                Transform player2LivesTransform = transform.Find("Player 2
Lives");
                if (player2LivesTransform != null)
                {
                    Image[] hearts =
player2LivesTransform.GetComponentsInChildren<Image>(true);
                    if (hearts != null && hearts.Length > 0)
                    {
                        player2Lives = hearts;
                        Debug.Log("Encontradas " + hearts.Length + " imágenes para
Player 2 Lives en el inspector");
                    }
                }
            }
        }
    }

    private void OnDestroy()
    {
        // Limpio suscripción al destruirse
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        // Si cambiamos a una escena de juego, me aseguro que la UI se inicialice
        if (scene.name.StartsWith("Game") || scene.name == "Game")
        {
            Debug.Log("LivesUIManager: Escena de juego detectada, inicializando
UI");
            StartCoroutine(InitializeUIAfterDelay(0.1f));

            // Si estamos en modo campaña, me aseguro de que la UI se inicialice
correctamente
            if (GlobalStateManager.Instance != null &&
GlobalStateManager.Instance.isCampaignMode)
            {
                Debug.Log("LivesUIManager: Modo campaña detectado, inicialización
```

```
adicional");
                StartCoroutine(InitializeUIAfterDelay(0.3f));
            }
        }
    }

    private void Start()
    {
        // Me aseguro de que la UI se inicialice correctamente al cargar la escena
        StartCoroutine(InitializeUIAfterDelay());
    }

    private IEnumerator InitializeUIAfterDelay(float additionalDelay = 0f)
    {
        // Espero el delay adicional si se especifica
        if (additionalDelay > 0)
        {
            Debug.Log("LivesUIManager: Esperando delay adicional de " +
additionalDelay + " segundos");
            yield return new WaitForSeconds(additionalDelay);
        }

        // Si estamos en modo campaña, me aseguro de esperar suficiente
        if (GlobalStateManager.Instance != null &&
GlobalStateManager.Instance.isCampaignMode)
        {
            Debug.Log("LivesUIManager: En modo campaña, esperando frame
adicional");
            yield return new WaitForEndOfFrame(); // Frame adicional para modo
campaña
        }

        // Espero un frame para que los Players se inicialicen
        yield return new WaitForEndOfFrame();

        // Me aseguro de que tengo referencias a los arrays de corazones
        EnsureUIComponentsExist();

        // Busco players en la escena e inicializo UI
        Player[] players = FindObjectsOfType<Player>();

        if (players.Length == 0)
        {
            Debug.LogWarning("LivesUIManager: No se encontraron jugadores en la
escena");
            yield break;
        }

        foreach (Player player in players)
        {
            // Me aseguro de que los GameObjects de UI estén activos
            string uiName = "Player" + player.playerNumber + "UI";
            GameObject playerUI = GameObject.Find(uiName);
            if (playerUI != null)
```

```
            {
                if (!playerUI.activeSelf)
                {
                    Debug.Log("LivesUIManager: Activando " + uiName + " que estaba
desactivado");
                    playerUI.SetActive(true);
                }

                // Si no tengo los corazones del jugador actual, los busco
                if ((player.playerNumber == 1 && (player1Lives == null ||
player1Lives.Length == 0)) ||
                    (player.playerNumber == 2 && (player2Lives == null ||
player2Lives.Length == 0)))
                {
                    // Intento con la estructura Player1Lives/Player2Lives primero
                Transform heartsParent = playerUI.transform.Find("Player" +
player.playerNumber + "Lives");

                // Si no existe, intento con Hearts (nombre alternativo)
                if (heartsParent == null) {
                    heartsParent = playerUI.transform.Find("Hearts");
                    if (heartsParent != null) {
                        Debug.Log("LivesUIManager: Usando fallback 'Hearts' para
Player " + player.playerNumber);
                    }
                }

                if (heartsParent != null)
                {
                    Image[] hearts = heartsParent.GetComponentsInChildren<Image>
(true);
                    if (player.playerNumber == 1)
                        player1Lives = hearts;
                    else
                        player2Lives = hearts;
                }
                }
            }
            else
            {
                Debug.LogError("LivesUIManager: No se encontró " + uiName + " en
la escena");
            }

            UpdateLivesUI(player.playerNumber, player.lives);
            Debug.Log("LivesUIManager: UI inicializada para Player " +
player.playerNumber + " con " + player.lives + " vidas");
        }
    }

    /// <summary>
    /// Actualiza la interfaz gráfica de las vidas visibles para un jugador.
    /// </summary>
    /// <param name="playerNumber">1 o 2</param>
```

```
    /// <param name="livesRemaining">Número actual de vidas</param>
    public void UpdateLivesUI(int playerNumber, int livesRemaining)
    {
        // Intento obtener las referencias si no las tengo
        if ((playerNumber == 1 && (player1Lives == null || player1Lives.Length ==
0)) ||
            (playerNumber == 2 && (player2Lives == null || player2Lives.Length ==
0)))
        {
            Debug.Log("UpdateLivesUI: Intentando conseguir componentes UI para
Player " + playerNumber);
            EnsureUIComponentsExist();
        }

        Image[] targetArray = playerNumber == 1 ? player1Lives : player2Lives;

        // Me aseguro de que el array no sea nulo
        if (targetArray == null || targetArray.Length == 0)
        {
            Debug.LogError("Hearts array is null or empty for player " +
playerNumber);

            // Último intento de obtener las imágenes
            string uiName = "Player " + playerNumber + " Lives";
            GameObject playerLivesObj = GameObject.Find(uiName);

            if (playerLivesObj != null)
            {
                Debug.Log("UpdateLivesUI: Último intento - Encontrado objeto " +
uiName);
                Image[] foundImages =
playerLivesObj.GetComponentsInChildren<Image>(true);

                if (foundImages != null && foundImages.Length > 0)
                {
                    if (playerNumber == 1)
                        player1Lives = foundImages;
                    else
                        player2Lives = foundImages;

                    targetArray = foundImages;
                    Debug.Log("UpdateLivesUI: Recuperadas " + foundImages.Length +
" imágenes para Player " + playerNumber);
                }
                else
                {
                    Debug.LogError("UpdateLivesUI: No se encontraron imágenes en "
+ uiName);
                    return;
                }
            }
            else
            {
                Debug.LogError("UpdateLivesUI: Último intento fallido - No se
```

```
encuentra " + uiName);
                return;
            }
        }

        // Imprimo información de depuración
        Debug.Log("Updating UI for Player " + playerNumber + " with " +
livesRemaining + " lives remaining");

        for (int i = 0; i < targetArray.Length; i++)
        {
            // Activo o desactivo el corazón según si la vida está activa
            if (targetArray[i] != null)
            {
                // Me aseguro de que el GameObject esté activo
                if (!targetArray[i].gameObject.activeInHierarchy)
                {
                    targetArray[i].gameObject.SetActive(true);
                }

                // En lugar de desactivar el componente, cambio la transparencia
                Color imageColor = targetArray[i].color;
                imageColor.a = i < livesRemaining ? 1f : 0f; // Alpha 1 (visible)
o 0 (invisible)
                targetArray[i].color = imageColor;

                // Mantengo el componente activo para no afectar el layout
                targetArray[i].enabled = true;

                Debug.Log("Player " + playerNumber + " heart " + i + " alpha set
to " + (i < livesRemaining ? "1" : "0"));
            }
            else
            {
                Debug.LogWarning("Heart image at index " + i + " is null for
player " + playerNumber);
            }
        }
    }

    private void EnsureUIComponentsExist()
    {
        // Busco el Global State Manager y su Canvas
        GameObject gsm = GameObject.Find("Global State Manager");
        if (gsm != null)
        {
            Transform canvas = gsm.transform.Find("Canvas");
            if (canvas != null)
            {
                // --- PLAYER 1 ---
                if (player1Lives == null || player1Lives.Length == 0)
                {
                    Transform player1UI = canvas.Find("Player1UI");
                    if (player1UI != null)
```

```
                    {
                        player1UI.gameObject.SetActive(true);
                        Transform lives = player1UI.Find("Player1Lives");
                        if (lives != null)
                        {
                            player1Lives = lives.GetComponentsInChildren<Image>
(true);
                            Debug.Log("LivesUIManager: Asignados corazones de
Player1 desde Global State Manager");
                        }
                        else
                        {
                            Debug.LogError("LivesUIManager: No se encontró
Player1Lives bajo Player1UI en Global State Manager");
                        }
                    }
                    else
                    {
                        Debug.LogError("LivesUIManager: No se encontró Player1UI
bajo Canvas en Global State Manager");
                    }
                }
                // --- PLAYER 2 ---
                if (player2Lives == null || player2Lives.Length == 0)
                {
                    Transform player2UI = canvas.Find("Player2UI");
                    if (player2UI != null)
                    {
                        player2UI.gameObject.SetActive(true);
                        Transform lives = player2UI.Find("Player2Lives");
                        if (lives != null)
                        {
                            player2Lives = lives.GetComponentsInChildren<Image>
(true);
                            Debug.Log("LivesUIManager: Asignados corazones de
Player2 desde Global State Manager");
                        }
                        else
                        {
                            Debug.LogError("LivesUIManager: No se encontró
Player2Lives bajo Player2UI en Global State Manager");
                        }
                    }
                    else
                    {
                        Debug.LogError("LivesUIManager: No se encontró Player2UI
bajo Canvas en Global State Manager");
                    }
                }
                return; // Si encuentro el Canvas en GSM, no sigo buscando en
otros lados
            }
        }
```

```
        // Si no tengo referencias a los corazones, los busco en la escena
        if (player1Lives == null || player1Lives.Length == 0)
        {
            GameObject player1UI = FindInAllScenes("Player1UI");
            if (player1UI != null)
            {
                player1UI.SetActive(true);

                // Intento primero obtener las imágenes directamente
                player1Lives = GetHeartImages(1);

                // Si no se encuentran por el método directo, busco por jerarquía
                if (player1Lives == null || player1Lives.Length == 0)
                {
                    // Intento con Player1Lives primero
                    Transform heartsParent =
player1UI.transform.Find("Player1Lives");

                    // Si no existe, intento con Hearts (nombre alternativo)
                    if (heartsParent == null) {
                        heartsParent = player1UI.transform.Find("Hearts");
                        if (heartsParent != null) {
                            Debug.Log("LivesUIManager: Usando fallback 'Hearts'
para Player 1");
                        }
                    }

                    if (heartsParent != null)
                    {
                        player1Lives = heartsParent.GetComponentsInChildren<Image>
(true);
                        Debug.Log("LivesUIManager: Encontrados " +
player1Lives.Length + " corazones para Player 1");
                    }
                    else
                    {
                        Debug.LogError("LivesUIManager: No se encontró el objeto
Player1Lives ni Hearts dentro de Player1UI");
                        // Intento listar los hijos para depuración
                        for (int i = 0; i < player1UI.transform.childCount; i++) {
                            Debug.Log("Player1UI child " + i + ": " +
player1UI.transform.GetChild(i).name);
                        }

                        // Como último recurso, creo imágenes de corazón
dinámicamente
                        if (player1Lives == null || player1Lives.Length == 0) {
                            CreateHeartImages(player1UI.transform, 1);
                        }
                    }
                }
            }
            else
            {
```

```
                Debug.LogError("LivesUIManager: No se encontró Player1UI en la
escena");
            }
        }

        if (player2Lives == null || player2Lives.Length == 0)
        {
            GameObject player2UI = FindInAllScenes("Player2UI");
            if (player2UI != null)
            {
                player2UI.SetActive(true);

                // Intentar primero obtener las imágenes directamente
                player2Lives = GetHeartImages(2);

                // Si no se encuentran por el método directo, buscar por jerarquía
                if (player2Lives == null || player2Lives.Length == 0)
                {
                    // Intento con Player2Lives primero
                    Transform heartsParent =
player2UI.transform.Find("Player2Lives");

                    // Si no existe, intento con Hearts (nombre alternativo)
                    if (heartsParent == null) {
                        heartsParent = player2UI.transform.Find("Hearts");
                        if (heartsParent != null) {
                            Debug.Log("LivesUIManager: Usando fallback 'Hearts'
para Player 2");
                        }
                    }

                    if (heartsParent != null)
                    {
                        player2Lives = heartsParent.GetComponentsInChildren<Image>
(true);
                        Debug.Log("LivesUIManager: Encontrados " +
player2Lives.Length + " corazones para Player 2");
                    }
                    else
                    {
                        Debug.LogError("LivesUIManager: No se encontró el objeto
Player2Lives ni Hearts dentro de Player2UI");
                        // Intento listar los hijos para depuración
                        for (int i = 0; i < player2UI.transform.childCount; i++) {
                            Debug.Log("Player2UI child " + i + ": " +
player2UI.transform.GetChild(i).name);
                        }

                        // Como último recurso, creo imágenes de corazón
dinámicamente
                        if (player2Lives == null || player2Lives.Length == 0) {
                            CreateHeartImages(player2UI.transform, 2);
                        }
                    }
```

```
                }
            }
            else
            {
                Debug.LogError("LivesUIManager: No se encontró Player2UI en la
escena");
            }
        }
    }

    // Para obtener imágenes de corazón de un jugador específico
    private Image[] GetHeartImages(int playerNumber)
    {
        string livesObjName = "Player " + playerNumber + " Lives";
        GameObject livesObj = GameObject.Find(livesObjName);

        if (livesObj != null)
        {
            Image[] hearts = livesObj.GetComponentsInChildren<Image>(true);
            if (hearts != null && hearts.Length > 0)
            {
                Debug.Log("LivesUIManager: Encontradas " + hearts.Length + "
imágenes para " + livesObjName);
                return hearts;
            }
        }

        Debug.Log("LivesUIManager: No se encontraron imágenes para " +
livesObjName + " por búsqueda directa");
        return null;
    }

    // Para crear imágenes de corazón dinámicamente si no se encuentran en la
escena
    private void CreateHeartImages(Transform parentTransform, int playerNumber)
    {
        Debug.Log("LivesUIManager: Creando imágenes de corazón para Player " +
playerNumber);

        // Creo un objeto padre para las imágenes de corazón
        GameObject livesHolder = new GameObject("Player" + playerNumber +
"Lives");
        livesHolder.transform.SetParent(parentTransform, false);

        // Creo tres imágenes de corazón (esto requiere que tengas un sprite de
corazón en tu proyecto)
        Image[] heartImages = new Image[3];
        for (int i = 0; i < 3; i++)
        {
            GameObject heartObj = new GameObject("Heart" + (i+1));
            heartObj.transform.SetParent(livesHolder.transform, false);

            // Añado componente de imagen
            Image heartImage = heartObj.AddComponent<Image>();
```

```
            // Intento cargar un sprite de corazón (esto depende de la estructura
de tu proyecto)
            // Si tienes una imagen de corazón en tu proyecto, reemplaza la ruta
            Sprite heartSprite = Resources.Load<Sprite>("Heart");
            if (heartSprite != null)
            {
                heartImage.sprite = heartSprite;
            }
            else
            {
                Debug.LogWarning("LivesUIManager: No se encontró un sprite de
corazón, usando un color sólido");
                heartImage.color = Color.red;
            }

            // Configuro el RectTransform para que tenga un tamaño adecuado
            RectTransform rectTransform = heartObj.GetComponent<RectTransform>();
            rectTransform.sizeDelta = new Vector2(30, 30);
            rectTransform.anchoredPosition = new Vector2(i * 35, 0);

            heartImages[i] = heartImage;
        }

        // Asigno las imágenes al array correspondiente
        if (playerNumber == 1)
            player1Lives = heartImages;
        else
            player2Lives = heartImages;

        Debug.Log("LivesUIManager: Creadas " + heartImages.Length + " imágenes de
corazón para Player " + playerNumber);
    }

    // Método auxiliar para buscar un objeto por nombre en toda la jerarquía,
incluyendo DontDestroyOnLoad
    private GameObject FindInAllScenes(string name)
    {
        Debug.Log("LivesUIManager: Buscando objeto UI '" + name + "' en todas las
escenas...");

        // Lista de nombres alternativos que podrían corresponder al objeto
        List<string> alternativeNames = new List<string>();
        if (name == "Player1UI")
        {
            alternativeNames.Add("Player 1 UI");
            alternativeNames.Add("UI Player 1");
            alternativeNames.Add("PlayerUI_1");
            alternativeNames.Add("P1UI");
        }
        else if (name == "Player2UI")
        {
            alternativeNames.Add("Player 2 UI");
            alternativeNames.Add("UI Player 2");
```

```
            alternativeNames.Add("PlayerUI_2");
            alternativeNames.Add("P2UI");
        }

        // 1. Intento búsqueda directa primero
        GameObject obj = GameObject.Find(name);
        if (obj != null) {
            Debug.Log("LivesUIManager: Encontrado " + name + " mediante búsqueda
directa");
            return obj;
        }

        // 1.5 Intento con nombres alternativos
        foreach (string altName in alternativeNames) {
            GameObject altObj = GameObject.Find(altName);
            if (altObj != null) {
                Debug.Log("LivesUIManager: Encontrado objeto alternativo '" +
altName + "' para '" + name + "'");
                return altObj;
            }
        }

        // 2. Busco en la escena activa
        Debug.Log("LivesUIManager: Buscando " + name + " en los objetos root de la
escena activa");
        foreach(GameObject rootObj in
UnityEngine.SceneManagement.SceneManager.GetActiveScene().GetRootGameObjects())
        {
            // Busco por nombre exacto
            if (rootObj.name == name) return rootObj;

            // Busco en los hijos profundos
            Transform found = rootObj.transform.FindDeepChild(name);
            if (found != null) {
                Debug.Log("LivesUIManager: Encontrado " + name + " como hijo
profundo de " + rootObj.name);
                return found.gameObject;
            }

            // Busco por nombres alternativos en hijos
            foreach (string altName in alternativeNames) {
                Transform altFound = rootObj.transform.FindDeepChild(altName);
                if (altFound != null) {
                    Debug.Log("LivesUIManager: Encontrado objeto alternativo '" +
altName + "' para '" + name + "'");
                    return altFound.gameObject;
                }
            }
        }

        // 3. Busco en objetos DontDestroyOnLoad
        Debug.Log("LivesUIManager: Buscando " + name + " en objetos
DontDestroyOnLoad");
        GameObject tempObj = new GameObject("__TempFinderObject");
```

```
        DontDestroyOnLoad(tempObj);
        Scene dontDestroyScene = tempObj.scene;
        DestroyImmediate(tempObj);

        foreach(GameObject rootObj in dontDestroyScene.GetRootGameObjects())
        {
            if (rootObj.name == name) {
                Debug.Log("LivesUIManager: Encontrado " + name + " en
DontDestroyOnLoad");
                return rootObj;
            }

            // Busco en hijos profundos
            Transform found = rootObj.transform.FindDeepChild(name);
            if (found != null) {
                Debug.Log("LivesUIManager: Encontrado " + name + " como hijo en
DontDestroyOnLoad");
                return found.gameObject;
            }

            // Busco por nombres alternativos
            foreach (string altName in alternativeNames) {
                if (rootObj.name == altName) {
                    Debug.Log("LivesUIManager: Encontrado objeto DontDestroyOnLoad
alternativo '" + altName + "' para '" + name + "'");
                    return rootObj;
                }

                Transform altFound = rootObj.transform.FindDeepChild(altName);
                if (altFound != null) {
                    Debug.Log("LivesUIManager: Encontrado objeto hijo alternativo
'" + altName + "' en DontDestroyOnLoad");
                    return altFound.gameObject;
                }
            }
        }

        // 4. Busco específicamente en Canvas y objetos UI
        Debug.Log("LivesUIManager: Buscando " + name + " en Canvas y objetos UI");
        Canvas[] allCanvases = GameObject.FindObjectsOfType(typeof(Canvas)) as
Canvas[];
        foreach(Canvas canvas in allCanvases)
        {
            Debug.Log("LivesUIManager: Examinando canvas: " + canvas.name);

            // Compruebo si el Canvas mismo es lo que busco
            if (canvas.name == name) {
                Debug.Log("LivesUIManager: El canvas mismo es " + name);
                return canvas.gameObject;
            }

            // Busco en los hijos del Canvas
            Transform found = canvas.transform.FindDeepChild(name);
            if (found != null)
```

```
            {
                Debug.Log("LivesUIManager: Encontrado " + name + " bajo el canvas:
" + canvas.name);
                return found.gameObject;
            }

            // Busco por nombres alternativos
            foreach (string altName in alternativeNames) {
                if (canvas.name == altName) {
                    Debug.Log("LivesUIManager: Canvas con nombre alternativo '" +
altName + "' para '" + name + "'");
                    return canvas.gameObject;
                }

                Transform altFound = canvas.transform.FindDeepChild(altName);
                if (altFound != null) {
                    Debug.Log("LivesUIManager: Encontrado objeto alternativo '" +
altName + "' bajo canvas " + canvas.name);
                    return altFound.gameObject;
                }
            }

            // Listo todos los hijos para depuración
            Debug.Log("LivesUIManager: Listando todos los hijos directos del
canvas " + canvas.name + ":");
            for (int i = 0; i < canvas.transform.childCount; i++) {
                Debug.Log("  - " + canvas.transform.GetChild(i).name);
            }
        }

        // 5. Busco objetos con componentes UI que podrían ser lo que busco
        Debug.Log("LivesUIManager: Buscando en todos los componentes UI");
        GameObject[] allObjects = GameObject.FindObjectsOfType(typeof(GameObject))
as GameObject[];
        foreach (GameObject go in allObjects) {
            // Si el objeto tiene componentes UI y su nombre contiene indicadores
            if ((go.GetComponent<RectTransform>() != null ||
                go.GetComponent<Canvas>() != null ||
                go.GetComponent<CanvasGroup>() != null) &&
                (go.name.ToLower().Contains("player") ||
go.name.ToLower().Contains("ui"))) {

                Debug.Log("LivesUIManager: Posible candidato para " + name + ": "
+ go.name);

                // Si busco Player1UI y el nombre contiene Player y 1,
probablemente es lo que busco
                if (name == "Player1UI" && go.name.ToLower().Contains("player") &&
go.name.Contains("1")) {
                    Debug.Log("LivesUIManager: Encontrado probable Player1UI: " +
go.name);
                    return go;
                }
```

```
                // Si busco Player2UI y el nombre contiene Player y 2,
probablemente es lo que busco
                if (name == "Player2UI" && go.name.ToLower().Contains("player") &&
go.name.Contains("2")) {
                    Debug.Log("LivesUIManager: Encontrado probable Player2UI: " +
go.name);
                    return go;
                }
            }
        }

        // 6. Busco en el Global State Manager
        Debug.Log("LivesUIManager: Buscando " + name + " en el Global State
Manager");
        GameObject gsm = GameObject.Find("Global State Manager");
        if (gsm != null)
        {
            Transform found = gsm.transform.FindDeepChild(name);
            if (found != null) {
                Debug.Log("LivesUIManager: Encontrado " + name + " en Global State
Manager");
                return found.gameObject;
            }

            // Busco por nombres alternativos
            foreach (string altName in alternativeNames) {
                Transform altFound = gsm.transform.FindDeepChild(altName);
                if (altFound != null) {
                    Debug.Log("LivesUIManager: Encontrado objeto alternativo '" +
altName + "' en Global State Manager");
                    return altFound.gameObject;
                }
            }
        }

        Debug.LogWarning("[LivesUIManager] No se pudo encontrar " + name + " en
ninguna parte de la jerarquía");
        return null;
    }
}

// Extensión para búsqueda profunda en jerarquía
public static class TransformDeepChildExtension
{
    public static Transform FindDeepChild(this Transform aParent, string aName)
    {
        foreach(Transform child in aParent)
        {
            if(child.name == aName )
                return child;
            var result = child.FindDeepChild(aName);
            if (result != null)
                return result;
        }
```

```
            return null;
        }
    }
```

Scripts de Gestión de Juego

GlobalStateManager.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\GlobalStateManager
.cs
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class GlobalStateManager : MonoBehaviour
{
    public static GlobalStateManager Instance;
    private WinnerDisplay winnerDisplay;

    public int player1Wins = 0;
    public int player2Wins = 0;
    public System.Collections.Generic.List<int> roundWinners = new
System.Collections.Generic.List<int>();
    private bool roundOver = false;
    public static bool nextCampaignMode = false;
    [HideInInspector]
    public bool isCampaignMode = false; // true para modo campaña, false para modo
individual
    public bool player1Won = false; // Para seguir si el jugador 1 ha ganado

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            isCampaignMode = nextCampaignMode;
        }
        else if (Instance != this)  // Solo destruyo si no soy la instancia actual
        {
            Debug.Log("Destruyendo instancia duplicada de GlobalStateManager");
            Destroy(gameObject);
            return;
        }

        // Busco el WinnerDisplay en la escena al inicio
        winnerDisplay = FindObjectOfType<WinnerDisplay>();
    }

    // Para actualizar la referencia al WinnerDisplay cuando cambia la escena
```

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

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

    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        // Actualizo la referencia al WinnerDisplay en la nueva escena
        winnerDisplay = FindObjectOfType<WinnerDisplay>();

        // Reseteo el estado roundOver cuando se carga una nueva escena de juego
en modo campaña
        if (isCampaignMode && (scene.name == "Game" || scene.name == "Game 1" ||
scene.name == "Game 2"))
        {
            Debug.Log("Nueva escena de campaña cargada: " + scene.name + ",
reseteando roundOver");
            roundOver = false;
        }

        // Si estamos en la escena de ganador, muestro el resultado
        if (scene.name == "WinnerScene" && winnerDisplay != null)
        {
            int winner = GetWinner();
            string winnerStr = winner == 1 ? "Player 1" : (winner == 2 ? "Player
2" : "");

            Debug.Log("WinnerScene cargada - Mostrando ganador: " + winnerStr +
                      " (P1: " + player1Wins + ", P2: " + player2Wins +
                      ", Último ganador en roundWinners: " + (roundWinners.Count >
0 ? roundWinners[roundWinners.Count - 1].ToString() : "ninguno") + ")");

            winnerDisplay.DisplayWinner(winnerStr, player1Wins, player2Wins);
        }
    }

    public void PlayerDied(int playerNumber)
    {
        if (roundOver) return;
        roundOver = true;

        int winner = (playerNumber == 1) ? 2 : 1;
        player1Won = (winner == 1);

        // Apunto el ganador de esta ronda
        roundWinners.Add(winner);

        if (winner == 1)
        {
```

```
            player1Wins++;
            Debug.Log("Jugador 1 gana esta ronda. Victorias: " + player1Wins);
        }
        else
        {
            player2Wins++;
            Debug.Log("Jugador 2 gana esta ronda. Victorias: " + player2Wins);
        }

        string currentScene = SceneManager.GetActiveScene().name;
        if (isCampaignMode)
        {
            Debug.Log("Modo Campaña - " + (currentScene == "Game" ? "Nivel 1" :
                                           currentScene == "Game 1" ? "Nivel 2" :
                                           currentScene == "Game 2" ? "Nivel Final"
: "Nivel Desconocido"));
        }

        StartCoroutine(LoadNextRound());
    }

    private IEnumerator LoadNextRound()
    {
        yield return new WaitForSeconds(2f);

        string currentScene = SceneManager.GetActiveScene().name;
        Debug.Log("[DEBUG] LoadNextRound() - Escena actual: '" + currentScene +
"', isCampaignMode: " + isCampaignMode);

        if (isCampaignMode)
        {
            if (currentScene == "Game")
            {
                Debug.Log("[DEBUG] Avanzando a Game 1...");
                SceneManager.LoadScene("Game 1");
            }
            else if (currentScene == "Game 1")
            {
                Debug.Log("[DEBUG] Avanzando a Game 2...");
                SceneManager.LoadScene("Game 2");
            }
            else if (currentScene == "Game 2")
            {
                Debug.Log("[DEBUG] Último nivel completado, mostrando resultados
finales...");
                SceneManager.LoadScene("WinnerScene");
                yield return new WaitForSeconds(0.1f);
                StartCoroutine(ReturnToMenuWithDelay());
                yield break;
            }
            else
            {
                Debug.LogWarning("[DEBUG] Escena inesperada en campaña: '" +
currentScene + "'");
```

```
            }
            // Reseteo el estado para el siguiente nivel
            yield return new WaitForSeconds(0.1f);
            roundOver = false;
            yield break;
        }
        else
        {
            Debug.Log("[DEBUG] FastGame - Estado antes de WinnerScene - P1: " +
player1Wins + ", P2: " + player2Wins + ", Último ganador: " + (roundWinners.Count
> 0 ? roundWinners[roundWinners.Count - 1].ToString() : "ninguno"));
            SceneManager.LoadScene("WinnerScene");
            yield return new WaitForSeconds(0.1f);
            StartCoroutine(ReturnToMenuWithDelay());
        }

        if (LightingManager.Instance != null)
        {

StartCoroutine(EnsureLightingAfterSceneLoad(LightingManager.Instance.ambientIntens
ity,

LightingManager.Instance.directionalLightIntensity));
        }
    }

    // Corrutina para asegurar que la iluminación se aplique bien después de
cargar la escena
    private IEnumerator EnsureLightingAfterSceneLoad(float ambientIntensity, float
directionalIntensity)
    {
        // Espero a que la escena esté completamente cargada
        yield return new WaitForSeconds(0.5f); // Aumentado de 0.2f a 0.5f para
dar más tiempo a la carga

        // Aplico la configuración de iluminación guardada
        if (LightingManager.Instance != null)
        {
            LightingManager.Instance.SetLightingIntensity(ambientIntensity,
directionalIntensity);
            Debug.Log("Iluminación restaurada después del cambio de escena");

            // Fuerzo una actualización adicional después de un ratito para
asegurar que se aplique bien
            StartCoroutine(ForceAdditionalLightingUpdate(ambientIntensity,
directionalIntensity));
        }
    }

    // Corrutina adicional para forzar una segunda actualización de iluminación
    private IEnumerator ForceAdditionalLightingUpdate(float ambientIntensity,
float directionalIntensity)
    {
        yield return new WaitForSeconds(0.3f);
```

```
        if (LightingManager.Instance != null)
        {
            LightingManager.Instance.SetLightingIntensity(ambientIntensity,
directionalIntensity);
            Debug.Log("Actualización adicional de iluminación aplicada");
        }
    }

    private void ShowFinalResults()
    {
        // Este método ya no se usa directamente, la lógica se ha movido a
LoadNextRound
        Debug.Log("ShowFinalResults está obsoleto, usar LoadNextRound");
    }

    private IEnumerator ReturnToMenuWithDelay()
    {
        // Espero lo suficiente para que el jugador vea el resultado
        yield return new WaitForSeconds(3f);

        // Guardo el estado final antes de resetear
        int finalWinner = GetWinner();
        int finalP1Wins = player1Wins;
        int finalP2Wins = player2Wins;
        Debug.Log("Estado final antes de volver al menú - Ganador: " + finalWinner
+ ", P1: " + finalP1Wins + ", P2: " + finalP2Wins);

        // Primero cargo el menú
        SceneManager.LoadScene("MainMenu");

        // Espero a que la escena se cargue completamente
        yield return new WaitForSeconds(0.1f);

        // Ahora sí, reseteo el estado
        ResetGameState();
    }

    private IEnumerator ReturnToMenu()
    {
        yield return new WaitForSeconds(3f);
        SceneManager.LoadScene("MainMenu");
        ResetGameState();
    }

    public void ResetGameState()
    {
        player1Wins = 0;
        player2Wins = 0;
        roundOver = false;
        player1Won = false;
        roundWinners.Clear();
    }
```

```
    public void ResetMatch()
    {
        roundOver = false;
        player1Won = false;
    }

    public int GetWinner()
    {
        // Para el modo FastGame, donde solo hay una ronda, uso el último ganador
        if (!isCampaignMode && roundWinners.Count > 0)
        {
            return roundWinners[roundWinners.Count - 1];
        }

        // Para el modo campaña y otros casos, comparo las victorias totales
        if (player1Wins > player2Wins)
            return 1;
        else if (player2Wins > player1Wins)
            return 2;
        else
            return 0; // Empate
    }

    public void SetCampaignMode(bool isCampaign)
    {
        isCampaignMode = isCampaign;
        nextCampaignMode = isCampaign;
    }
}
```

LightingManager.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\LightingManager.cs
using UnityEngine;
using UnityEngine.SceneManagement;

public class LightingManager : MonoBehaviour
{
    public static LightingManager Instance;

    // Configuración de iluminación para mantener consistencia entre escenas
    [Header("Configuración de Iluminación")]
    public Color ambientLightColor = new Color(0.5f, 0.5f, 0.5f, 1.0f); // Color
gris claro por defecto
    public float ambientIntensity = 1.0f;
    public float directionalLightIntensity = 1.0f;

    [Tooltip("Tiempo de espera en segundos antes de aplicar la iluminación")]
    public float lightingApplyDelay = 0.5f;
```

```
    private void Awake()
    {
        // Singleton para acceso global
        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject); // Mantengo entre escenas
        }
        else
        {
            Destroy(gameObject);
        }

        // Me apunto al evento para cuando cambie la escena
        SceneManager.sceneLoaded += OnSceneLoaded;

        // Aplico configuración inicial
        ApplyLightingSettings();
    }

    private void OnDestroy()
    {
        // Quito el registro del evento al destruir
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    // Se llama cuando se carga una nueva escena
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        // Aplico la misma configuración de iluminación en cada escena
        ApplyLightingSettings();
        Debug.Log("Configuración de iluminación aplicada a la escena: " +
scene.name);
    }

    // Permite establecer los valores de iluminación ANTES de aplicar la
configuración en la nueva escena
    public void SetLightingValues(float ambientValue, float directionalValue)
    {
        ambientIntensity = ambientValue;
        directionalLightIntensity = directionalValue;
    }
    // Aplica la configuración de iluminación a la escena actual
    public void ApplyLightingSettings()
    {
        // Configuro iluminación ambiental
        RenderSettings.ambientLight = ambientLightColor;
        RenderSettings.ambientIntensity = ambientIntensity;

        // Espero un ratito para asegurar que todas las luces estén cargadas
        StopAllCoroutines(); // Paro cualquier corrutina anterior para evitar
conflictos
        StartCoroutine(ConfigureLightsAfterDelay());
    }
```

```
    // Corrutina para configurar las luces después de un poquito de delay
    private System.Collections.IEnumerator ConfigureLightsAfterDelay()
    {
        // Espero un tiempo específico para asegurar que todos los objetos estén
inicializados
        yield return new WaitForSeconds(lightingApplyDelay);

        // Busco y configuro la luz direccional principal si existe
        Light[] lights = FindObjectsOfType<Light>();
        bool foundDirectionalLight = false;

        foreach (Light light in lights)
        {
            if (light.type == LightType.Directional)
            {
                light.intensity = directionalLightIntensity;
                foundDirectionalLight = true;
                Debug.Log("Luz direccional configurada con intensidad: " +
directionalLightIntensity + " en objeto: " + light.gameObject.name);
            }
        }

        if (!foundDirectionalLight)
        {
            Debug.LogWarning("No se encontró ninguna luz direccional en la escena
actual. Creando una luz direccional temporal.");
            CreateTemporaryDirectionalLight();
        }

        // Fuerzo una segunda actualización después de un poquito más de retraso
        yield return new WaitForSeconds(0.2f);

        // Verifico nuevamente las luces direccionales (por si alguna se creó
después)
        lights = FindObjectsOfType<Light>();
        foreach (Light light in lights)
        {
            if (light.type == LightType.Directional)
            {
                light.intensity = directionalLightIntensity;
                Debug.Log("Luz direccional actualizada en segunda pasada con
intensidad: " + directionalLightIntensity);
            }
        }
    }

    // Creo una luz direccional temporal si no encuentro ninguna en la escena
    private void CreateTemporaryDirectionalLight()
    {
        GameObject lightObj = new GameObject("Temporary Directional Light");
        Light light = lightObj.AddComponent<Light>();
        light.type = LightType.Directional;
        light.intensity = directionalLightIntensity;
```

```
        light.color = Color.white;
        lightObj.transform.rotation = Quaternion.Euler(50f, -30f, 0f); // Rotación
estándar para luz direccional
        Debug.Log("Luz direccional temporal creada con intensidad: " +
directionalLightIntensity);
    }

    // Método público para ajustar la intensidad de la iluminación en tiempo de
ejecución
    public void SetLightingIntensity(float ambientValue, float directionalValue)
    {
        ambientIntensity = ambientValue;
        directionalLightIntensity = directionalValue;
        ApplyLightingSettings();
    }
}
```

Otros Scripts

DisableTriggerOnPlayerExit.cs

```
// filepath:
c:\Users\fumop\Documents\DAM\BombermanProyecto3D\Assets\Scripts\DisableTriggerOnPl
ayerExit.cs
using UnityEngine;
using System.Collections;

/// <summary>
/// Este script se asegura de que se pueda poner una bomba a los pies del jugador
sin que cause movimiento raro cuando el jugador se aleje.
/// Desactiva el trigger del collider, haciendo que el objeto sea sólido.
/// </summary>
public class DisableTriggerOnPlayerExit : MonoBehaviour
{

    public void OnTriggerExit (Collider other)
    {
        if (other.gameObject.CompareTag ("Player"))
        { // Cuando el jugador sale del área del trigger
            GetComponent<Collider> ().isTrigger = false; // Desactivo el trigger
        }
    }
}
```
