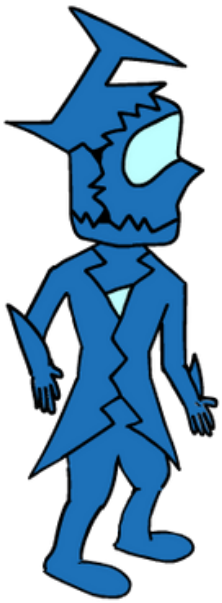
MEMORIA VIDEOJUEGO

Por: Piero Benjamín Felices Paulet

Uso de elementos gráficos realizados por ti

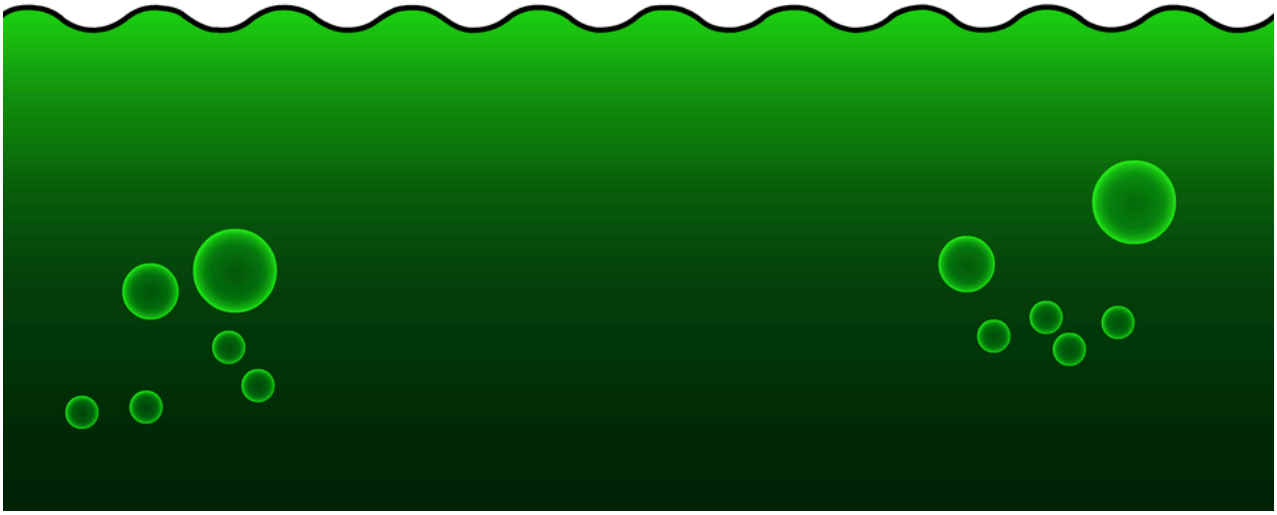
Todos los gráficos y tema visual del videojuego han sido hechos por mi. Para la creación de los dibujos he usado Adobe Illustrator y para la creación de las animaciones he usado Adobe Animate.

Personaje

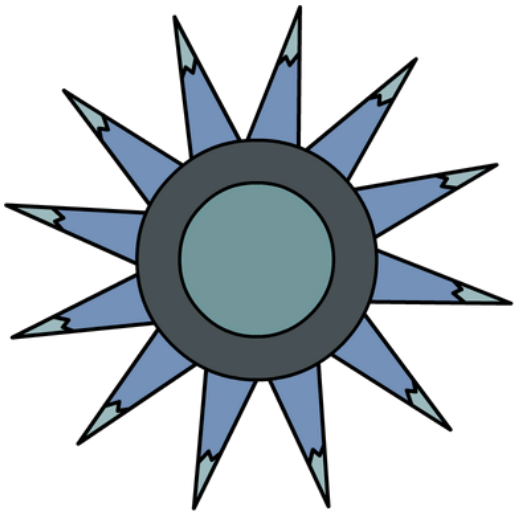


Enemigo

Ácido

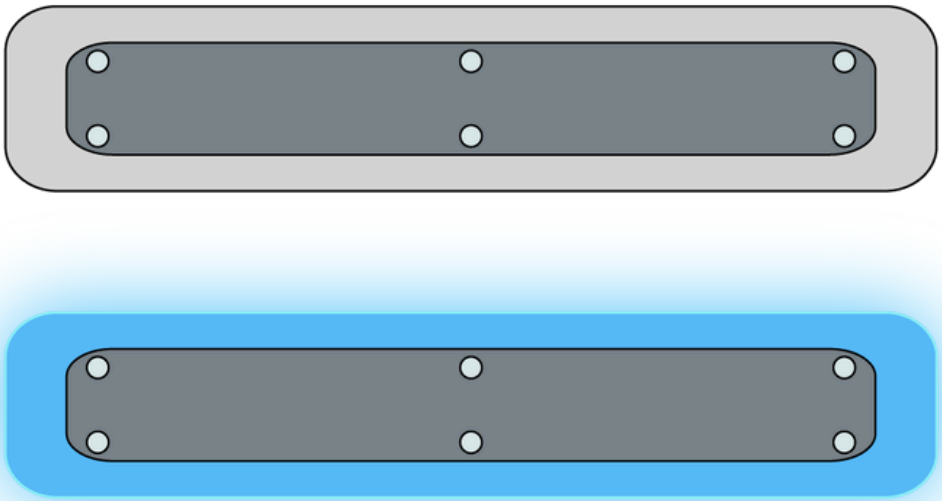


Pinchos



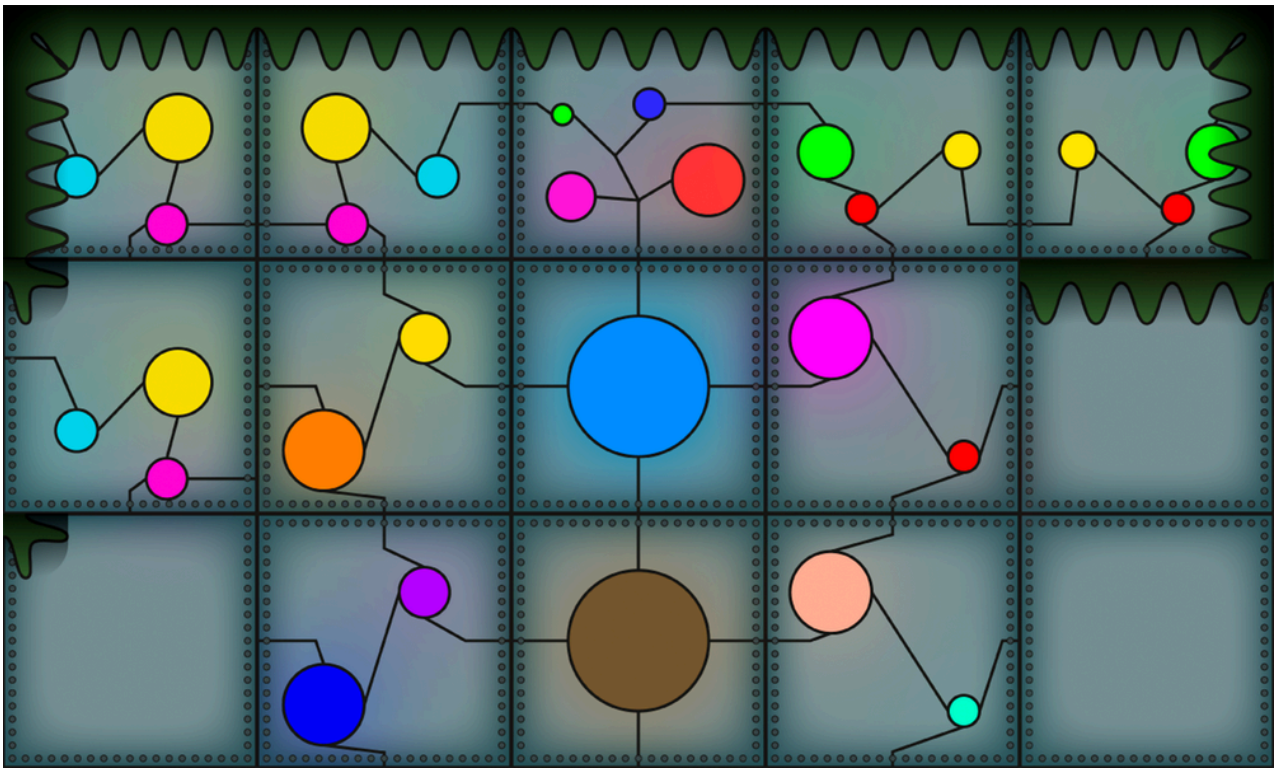
Fuego

Plataformas

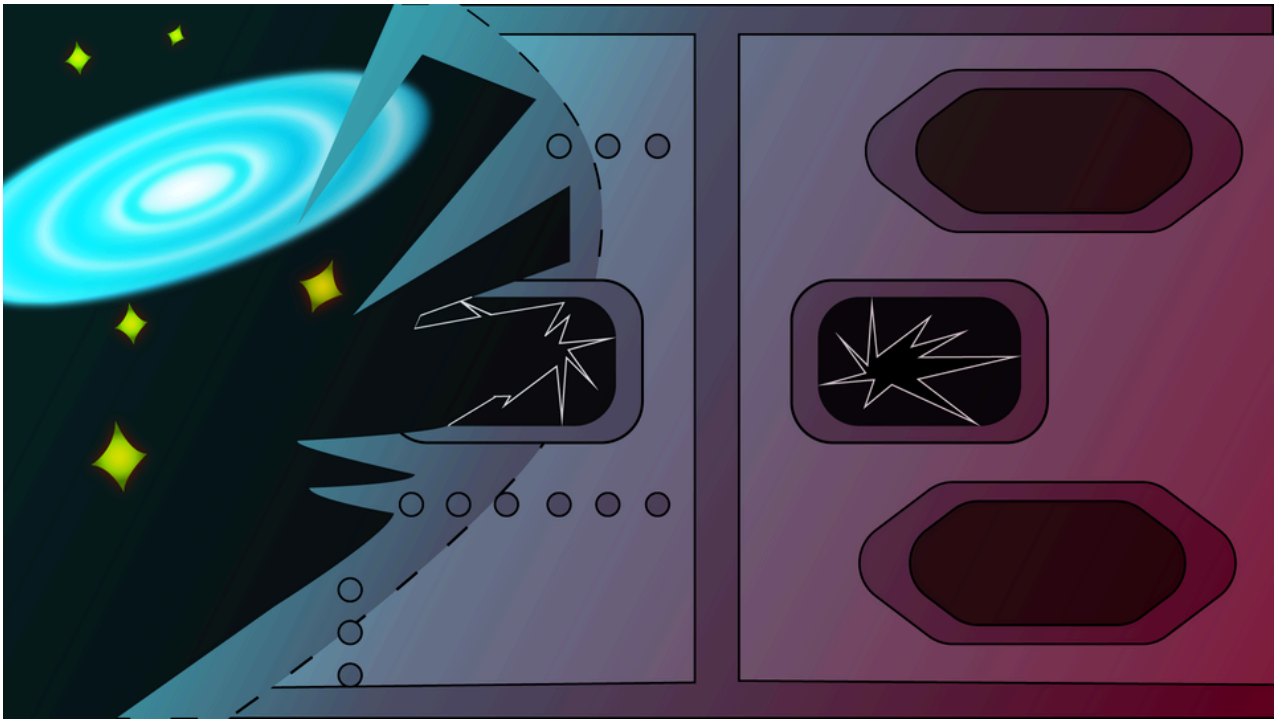


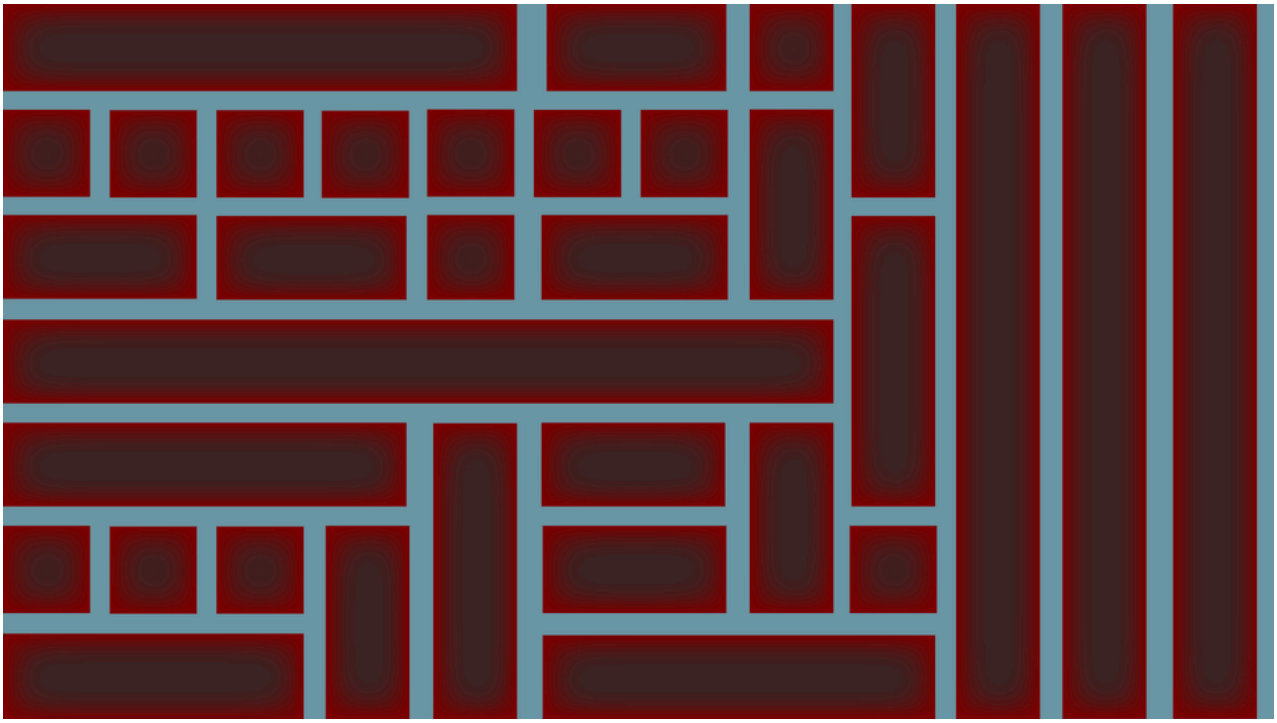
TileSet Tierra

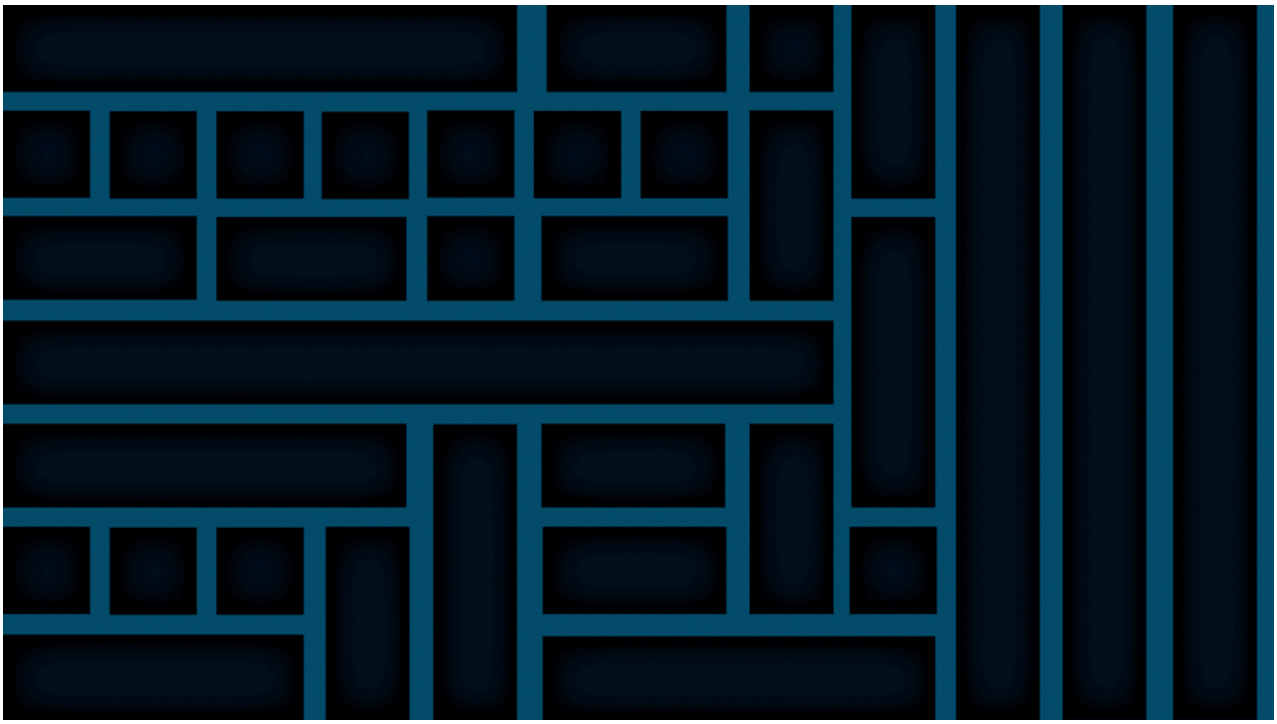
TileSet Metal



Fondos



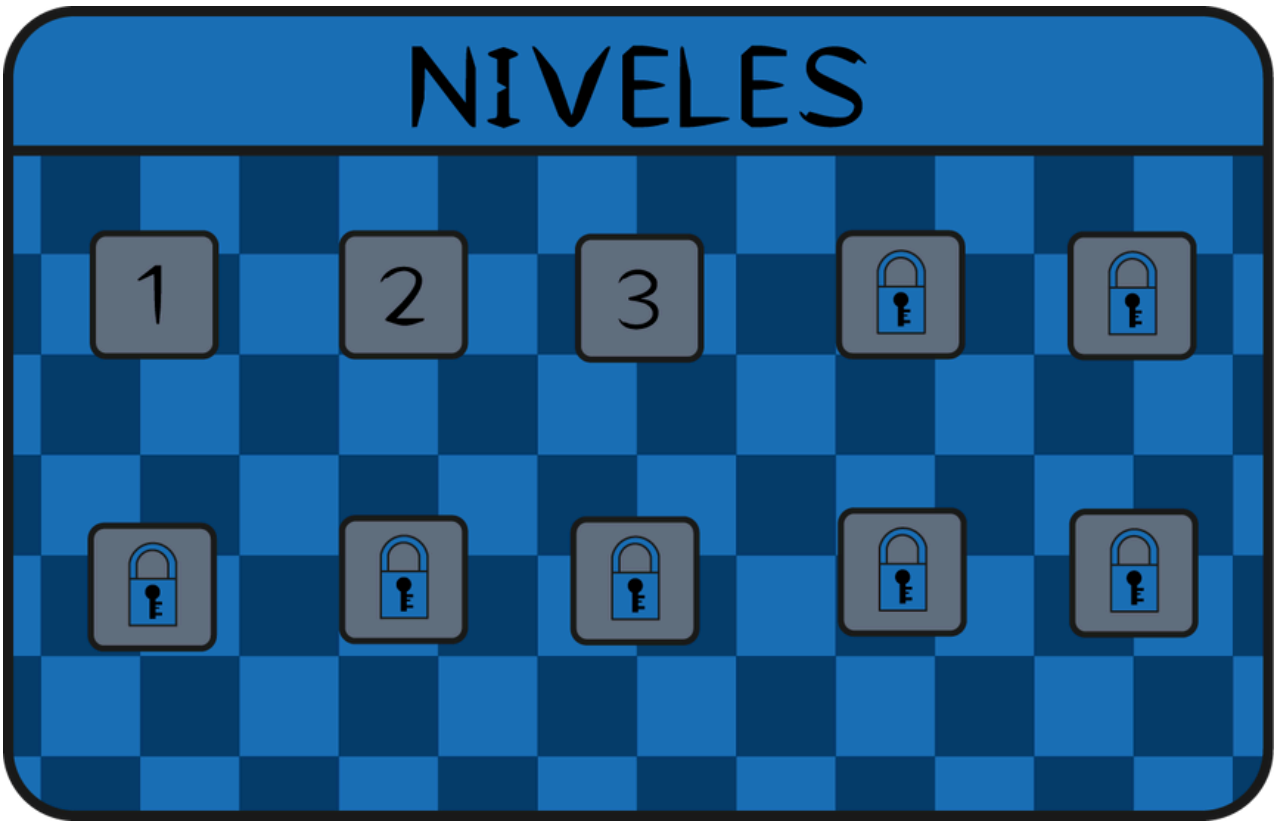




HUD


Vesselbound
START
MADE BY ARNIXIN PRODUCTIONS


Vessel
bound
PLAY
LEVEL
CREDITS
QUIT
MADE BY ARNIXIN PRODUCTIONS


NIVELES
1
2
3


CREADOR
Piero Benjamín
Felices Paulet

Listado de los scripts utilizados explicando sus funcionalidades y mostrando

Empecé por los HUD, aquí se me dificultó hacer que los menús o fondos se mostrarán o se ocultarán al pulsar los botones debido a que si un botón le daba visibilidad a un menú, el siguiente botón por alguna razón no podía ocultarlo. Al final, siguiendo una organización pude solventarlo y comprendí el funcionamiento detrás de llamar a un gameObject en un script, además de comprender cuando se hace private class y public class.

Separé los scripts en dos carpetas: “FUNCION” e “INTERACCION”

Cabe recalcar que en un principio, estos scripts estarían asociados a gameObjects vacíos. Para llamar a los gameObjects de cada menú usé varios “public GameObject nombreClave;” para luego referenciar a cada uno arrastrándolo en el inspector.

En la carpeta FUNCION estarían los scripts que definirían las funciones de cada gameObject y les pondría un nombre a cada uno, por ejemplo pongo la función “public void Mostrar()” y le digo que el gameObject del menú se visualice, hice esto con todos los menús.

```
using UnityEngine;

public class FUNCION_PANEL_MENU : MonoBehaviour
{
    public GameObject plano; // El plano al que se refiere este script

    public void Ocultar()
    {
        if (plano != null)
        {
            plano.SetActive(false);
            Debug.Log("PANEL_MENU ocultado.");
        }
        else
        {
            Debug.LogWarning("PANEL_MENU no esta asignado.");
        }
    }

    public void Mostrar()
    {
        if (plano != null)
        {
            plano.SetActive(true);
            Debug.Log("PANEL_MENU mostrado.");
        }
        else
        {
            Debug.LogWarning("PANEL_MENU no esta asignado.");
        }
    }
}
```

En la carpeta INTERACCION estarían los scripts que establecerían la activación de las funciones de los scripts FUNCION. Aquí llamé al gameObject de los botones y llame a los scripts FUNCION con “public FUNCION nombreClave;”

```
using UnityEngine;

public class INTERACCION_PANEL_MENU_BOTONES_NIVELES : MonoBehaviour
{
    public FUNCION_PANEL_NIVEL funcion3;
    public FUNCION_PANEL_NIVEL_ANIMACION funcion3_1;
    public FUNCION_PANEL_NIVEL_ANIMACION_1 funcion3_1_1;
    public FUNCION_PANEL_NIVEL_ANIMACION_2 funcion3_1_2;
    public FUNCION_PANEL_NIVEL_ANIMACION_3 funcion3_1_3;
    public FUNCION_PANEL_NIVEL_INTERACTUAR funcion3_2;
    public GameObject personaje;

    void Start()
    {
        GameObject personaje = GameObject.FindWithTag("PERSONAJE");
    }

    public void AlPulsarBoton()
    {
        if (funcion3 != null)
        {
            funcion3.Mostrar();
        }

        if (funcion3_1 != null)
        {
            funcion3_1.Mostrar();
        }

        if (funcion3_1_1 != null)
        {
            funcion3_1_1.Mostrar();
        }

        if (funcion3_1_2 != null)
        {
            funcion3_1_2.Ocultar();
        }
```

Con esto, ya tenía los botones funcionales

Para hacer las transiciones entre menús, puse un evento en el último fotograma de cada animación. En el void Start() puse que los gameObjects de las animaciones estarían ocultas. Cuando pulsará el botón, el gameObject de la animación se mostraría por encima del menú usando Sorting Layers. Cuando llegara al evento del último fotograma, el gameObject de la animación se volvería a ocultar, dejando ver otro menú diferente al anterior con botones funcionales. En este caso hice que cambiará de escena.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class INTERACCION_PANEL_INICIO_ANIMACION : MonoBehaviour
{
    public void EventoFin()
    {
        SceneManager.LoadScene(0);
    }
}
```

Seguimos con el cambio entre escenas, cuando le daba al botón play también tendría que cambiar de escena. En el builder scene añadí las escenas que agregué y use su número de referencia para nombrarlas en el script y decir a cuál de ellas me quiero cambiar.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class INTERACCION_PANEL_MENU_BOTONES_JUGAR : MonoBehaviour
{
    public void CargarEscena()
    {
        SceneManager.LoadScene(1); // Cambia a la escena de indice 1
        Debug.Log("Cargando escena: 1");
    }

    public void Update()
    {
        // Verifica si la escena activa es la de indice 1
        if (SceneManager.GetActiveScene().buildIndex == 1)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
            Debug.Log("Reiniciando la escena actual");
        }
    }
}
```

En las demás escenas empecé a diseñar como serían los niveles, poniendo los TileMaps, suelos, paredes y puse varios BoxCollider2D

Ahora vamos con el movimiento del personaje, primero puse un cubo con su RigidBody2D, un BoxCollider2D y empecé a darle movimientos en x y saltos.

```
using UnityEngine;

public class INTERACCION_PERSONAJE_MOVIMIENTO : MonoBehaviour
{
    private Rigidbody2D _rb;
    private INTERACCION_PERSONAJE_MOVIMIENTO_DASH _Dash;
    private Animator animator;

    [Header("Movement")]
    [SerializeField] private float _speed = 45f;
    private float direction;
    public float Direction => direction;

    [Header("Jump")]
    [SerializeField] private float _jumpForce = 21f;
    [SerializeField] private Transform _checkGround;
    [SerializeField] private float _raycastLength;
    [SerializeField] private LayerMask _groundLayer;
    private bool _isGrounded; // Si el personaje esta tocando el suelo
    private bool _isJumping;  // Si el personaje esta en el aire

    private void Awake()
    {
        _rb = GetComponent<Rigidbody2D>();
        if (_rb == null)
        {
            Debug.LogError("El personaje no tiene un Rigidbody2D asignado.");
        }
        _Dash = GetComponent<INTERACCION_PERSONAJE_MOVIMIENTO_DASH>();
    }

    void Start()
    {
        animator = GetComponent<Animator>();
    }
```

Mencioné a su Rigidbody2D y a su Animator para las animaciones del personaje. También mencioné al script del Dash que indicará que el personaje solo pueda saltar o correr cuando no esté hacienda Dash.

Siguiendo, puse un Header donde iría las físicas de movimiento. Hice un float _speed y direction.

En el Jump, puse un _JumpForce, un _checkGround para verificar si el personaje ha tocado el suelo, un _raycastLenght que es un rayo hacia abajo que emitirá un BoxCollider2D para evitar que el personaje haga un doble salto. Creo un _groundLayer para poner un layer de Ground a todos los gameObjects que contengan los BoxCollider2D del mapa. Por ultimo, una booleana para verificar si el personaje toca el suelo y una booleana para verificar si el personaje está saltando.

```
void Update()
{
    direction = Input.GetAxisRaw("Horizontal");

    // Actualizar estado de salto y suelo
    _isGrounded = Physics2D.Raycast(_checkGround.position, Vector2.down, _raycastLength, _groundLayer);

    if (!_Dash.IsDashing)
    {
        Jump();
    }
}

private void FixedUpdate()
{
    if (!_Dash.IsDashing)
    {
        Move();
    }
}

private void Move()
{
    _rb.velocity = new Vector2(direction * _speed, _rb.velocity.y);

    // Verificar si el personaje está en el suelo antes de permitir la animación de correr
    if (_isGrounded && Mathf.Abs(direction) > 0)
    {
        animator.SetBool("CORRER", true);
        animator.SetBool("ESTANDAR", false);
    }
    else if (_isGrounded) // Si está en el suelo pero no se mueve
    {
        animator.SetBool("CORRER", false);
        animator.SetBool("ESTANDAR", true);
    }

    // Girar al personaje dependiendo de la dirección
    if (direction > 0)
    {
        transform.localScale = new Vector3(1, 1, 1); // Orientación normal (derecha)
    }
    else if (direction < 0)
    {
        transform.localScale = new Vector3(-1, 1, 1); // Invertir la escala X para girar a la izquierda
    }
}
```

En las funciones, le digo que mientras el personaje este no esté dasheando, puede saltar y moverse

Creo un update en el que le digo que el RayCast envié un rayo hacia abajo y verifique si en todo momento si esta en contacto con un groundLayer, es decir con los BoxCollider2D del mapa. Este resultado definirá el estado del _isGrounded.

En el void Move, agarro el parámetro del RigidBody2D para ajustarle las físicas y en su velocidad le digo que el valor en x será igual a su direction *_speed.

Si el personaje está tocando el suelo y la direccion es mayor a cero se esta moviendo por lo que ajusto la animación a CORRER. Si el personaje solo está en el suelo sin movimiento, su animación será la ESTANDAR

Luego le digo que si la dirección es mayor que cero, cambie la escala para que el personaje Flipee o haga la acción de girar para dirigirse a dicha dirección

```
private void Jump()
{
    // Dibuja el raycast en la ventana de escena para verificar su posición
    Debug.DrawRay(_checkGround.position, Vector2.down * _raycastLength, Color.red);

    // Lanza el raycast hacia abajo para verificar si el personaje está en el suelo
    _isGrounded = Physics2D.Raycast(_checkGround.position, Vector2.down, _raycastLength, _groundLayer);

    print("Está en el suelo: " + _isGrounded); // Para depurar

    // Si se presiona la tecla de salto y el personaje está en el suelo
    if (Input.GetButtonDown("Jump") && _isGrounded)
    {
        _rb.velocity = Vector2.up * _jumpForce; // Añadir fuerza hacia arriba
        animator.SetBool("SALTAR", true);       // Activar la animación de salto
        animator.SetBool("CORRER", false);      // Detener otras animaciones
        animator.SetBool("ESTANDAR", false);
    }

    // Si el personaje está en el suelo
    if (_isGrounded)
    {
        animator.SetBool("SALTAR", false); // Detener la animación de salto al aterrizar
    }
}

private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.tag == "PLATAFORMA")
    {
        transform.parent = collision.transform;
    }
}

private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.gameObject.tag == "PLATAFORMA")
    {
        transform.parent = null;
    }
}
```

En la función Jump le digo que si pulsa la tecla para saltar y está en el suelo, modifique la física del RigidBody en el eje y Vector2.up. Además le ajusto su animación correspondiente.

En las últimas funciones referencio a los gameObjects que tengan un Tag llamado PLATAFORMA, las cuales corresponden a las plataformas móviles que hay encima del liquido de ácido del juego. Hago que el gameObject del personaje se haga hijo de las plataformas mientras este esté en contacto con ellas, para que sigan su posición estando quieto. Cuando salte, dejará de ser hijo de las plataformas.

Este sería el código para que el personaje haga Dash

```
using UnityEngine;
using System.Collections;

public class INTERACCION_PERSONAJE_MOVIMIENTO_DASH : MonoBehaviour
{
    private Rigidbody2D _rb;
    private INTERACCION_PERSONAJE_MOVIMIENTO _movimientoBase;
    private float _baseGravity;

    [Header("Dash")]
    [SerializeField] private float _dashingTime = 0.3f;
    [SerializeField] private float _dashForce = 90f;
    [SerializeField] private float _timeCanDash = 0.5f;
    private bool _isDashing;
    private bool _canDash = true;
    public bool IsDashing => _isDashing;

    private void Awake()
    {
        _rb = GetComponent<Rigidbody2D>();
        _movimientoBase = GetComponent<INTERACCION_PERSONAJE_MOVIMIENTO>();
        _baseGravity = _rb.gravityScale;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.W))
        {
            StartCoroutine(Dash());
        }
    }
```

Aquí le pongo las variables que determinan las carácteristicas del Dash. Cabe señalar que como el _isDashing esta en un private bool, habrá que crear un public bool IsDashing y decir que es igual a esa variable para poder llevar esta referencia al script del movimiento del personaje.

Cuando pulso la W, hago uso de un StartCoroutine que me permite poner una funcion dentro de otra. Al usar esto, tengo que poner un using System.Collections; en el script para que funcione.

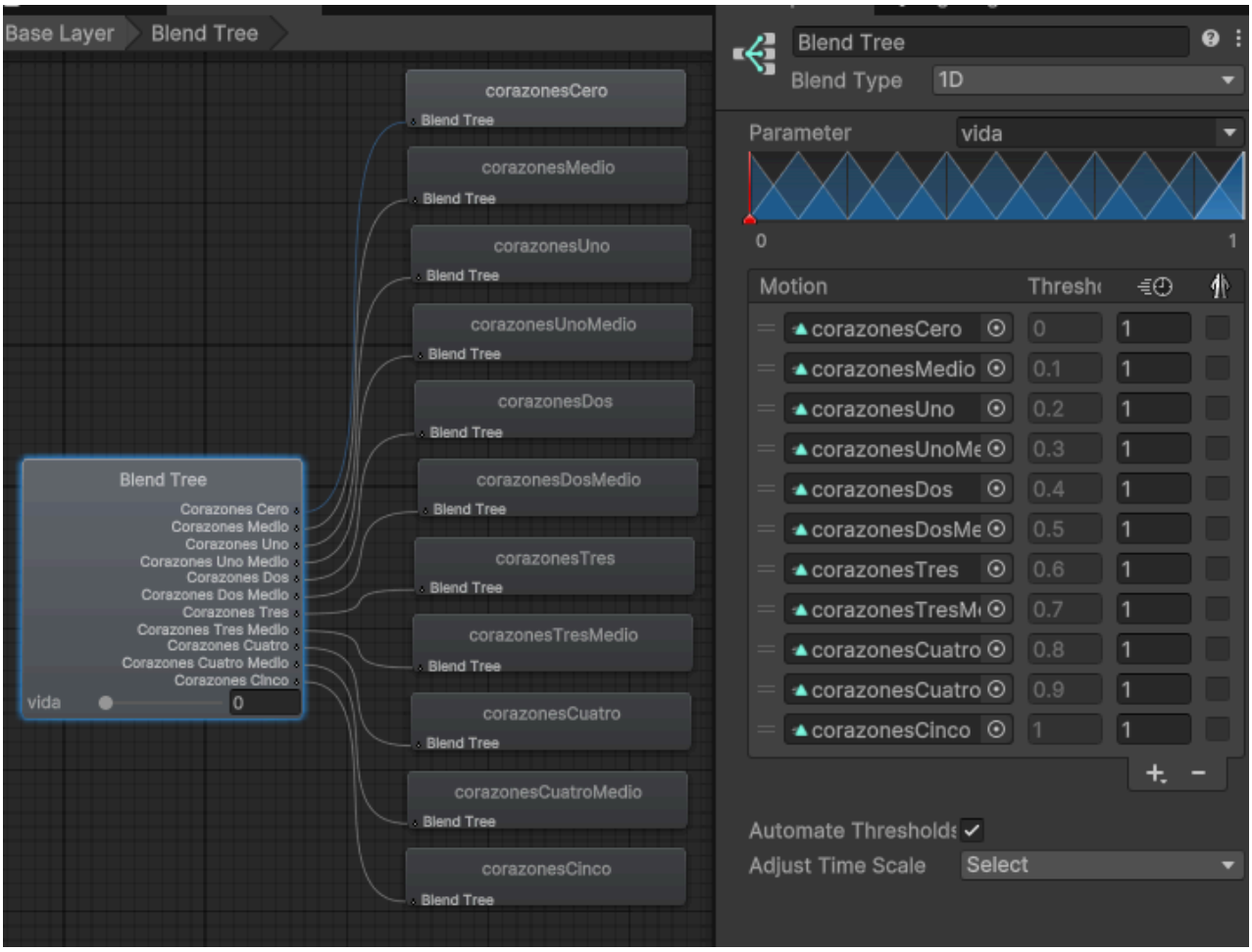
```
private IEnumerator Dash()
{
    if (_movimientoBase.Direction != 0 && _canDash)
    {
        //Si esta dasheando
        _isDashing = true;
        //No puede dashear otra vez mientras dashea
        _canDash = false;
        //Gravedad cero para que el Dash sea recto
        _rb.gravityScale = 0f;
        _rb.velocity = new Vector2(_movimientoBase.Direction * _dashForce, 0f);
        //Le pedimos a unity que espere el tiempo que dura el Dash para seguir leyendo los proximos codigos
        yield return new WaitForSeconds(_dashingTime);
        //Despues del dashingTime ya no esta dasheando
        _isDashing = false;
        //La gravedad vuelve a su estado normal
        _rb.gravityScale = _baseGravity;
        //Para darle cooldown al Dash
        yield return new WaitForSeconds(_timeCanDash);
        _canDash = true;
    }

}
```

En este sitio esta la función Dash, que para poder mencionarla en el StartCoroutine debo de poner un IEnumerator. En la función describo el orden en el que ocurriran las cosas. Señalar hago uso del yield return new WaitForSeconds() que me define el tiempo que espera el ordenador para seguir leyendo la próxima línea de código. La primera corresponde al tiempo que se pasa el jugador dasheando para que luego diga que el _isDashing está en false. La segunda es para ponerle un cooldown o tiempo de espera al dash para evitar que el jugador pueda hacer varios dashes seguidos.

Este es el código para la vida del personaje en el que los corazones de vida se van cambiando a medida que el personaje va perdiendo vida por haber estado en contacto con el BoxCollider2D de las trampas o enemigos con Tag DAÑO o con Tag MUERTE

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class INTERACCION_PERSONAJE_VIDA : MonoBehaviour
{
    public Animator corazonesAnimator; // Referencia al Animator del HUD (pantalla)
    public float vida = 1f;            // Vida inicial del personaje

    void Start()
    {
        vida = 1f; // Restablece la vida al máximo

        // Si no se ha asignado el Animator, buscarlo por Tag
        if (corazonesAnimator == null)
        {
            GameObject corazones = GameObject.FindGameObjectWithTag("VIDA");
            if (corazones != null)
            {
                corazonesAnimator = corazones.GetComponent<Animator>();
            }
            else
            {
                Debug.LogError("No se ha encontrado un GameObject con el Tag 'VIDA'. Aseg
            }
        }

        if (corazonesAnimator != null)
        {
            corazonesAnimator.SetFloat("vida", vida);  // Actualizamos el valor de vida a
        }
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        // Si colisiona con un objeto de tag "DAÑO", reduce la vida
        if (collision.CompareTag("DAÑO"))
        {
            ReducirVida(0.1f);
        }

        if (collision.CompareTag("MUERTE"))
        {
            ReducirVida(1f);
        }
    }

    private void ReducirVida(float cantidad)
    {
        // Reducir la vida y asegurarse de que no sea menor que 0
        vida = Mathf.Max(vida - cantidad, 0);

        // Actualizar el valor de vida en el Animator (el parámetro "vida")
        if (corazonesAnimator != null)
        {
            corazonesAnimator.SetFloat("vida", vida);
        }

        // Mostrar la vida actual en la consola (opcional para depuración)
        Debug.Log($"Vida actual: {vida}");

        // Comprobar si el personaje ha muerto (si la vida es 0)
        if (vida <= 0)
        {
            Destroy(gameObject);
            SceneManager.LoadScene(0);
            Debug.Log("¡El personaje ha muerto!");
            // Aquí podrías llamar a una función para manejar la muerte del personaje, co
        }
    }
}
```

Aquí llamo al Animator del gameObject que contiene a los corazones. Y le digo que si no encuentra al gameObject que lo busque por su Tag VIDA. Le digo que si el personaje recibe daño del Tag DAÑO reduzca la vida 0.1f. Si el personaje recibe daño del Tag MERTE reduzca la vida 1f.



Estos valores es debido a que cree un Blend Tree en la animación de los corazones y cree un parámetro bool llamado vida. Cada clip de animación es un fotograma del corazón en los diferentes estados, aunque esto también sirve para crear transiciones entre varias animaciones.

Al final, le pongo que si la vida del personaje es menor igual a a cero que se destruya el objeto y cargue la escena número 0 que corresponde al menú. Es importante que el personaje se destruya antes de cambiar de escena por esto que voy a mencionar a continuación.

El personaje al pasar por el BoxCollider2D de la luz azul, que supone el final del mapa, cambiaría de escena a otro mapa del juego, sin embargo, este personaje no se mantiene en la otra escena porque no está. Pensé que copiar y pegar el personaje en las demás escenas me llevaría a errores futuros y mucha complicación con los scripts, así que mediante un script hice que el personaje no se destruyera si cambiaba de escena.

```
using UnityEngine;

public class INTERACCION_PERSONAJE_MANTENER_ENTRE_ESCENA : MonoBehaviour
{
    private static INTERACCION_PERSONAJE_MANTENER_ENTRE_ESCENA instancia;

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

Por esto es importante que el personaje al tener una vida igual o menor a 0 o cruce por la puerta final de un nivel, se destruya antes de cambiar de escena.

El gameObject de los corazones y el gameObject de la cámara también tienen este código y también tengo referenciado al gameObject del jugador para decir que si el gameObject de este se destruye, que tanto el gameObject de la cámara como el gameObject de los corazones también se destruyan.

En este script hago que las plataformas móviles se muevan de un lado al otro. Hice dos gameObjects vacíos y cada uno lo posicione al principio y al final hacia donde llegaría la plataforma.

```
using UnityEngine;

public class INTERACCION_PLATAFORMA_MOVIMIENTO_1 : MonoBehaviour
{
    public GameObject plataforma;
    public Transform startPoint;
    public Transform endPoint;

    public float velocidad;

    private Vector3 direccion;

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        direccion = endPoint.position;
    }

    // Update is called once per frame
    void Update()
    {
        plataforma.transform.position = Vector3.MoveTowards(plataforma.transform.position, direccion, velocidad * Time.deltaTime);

        if(plataforma.transform.position == endPoint.position)
        {
            direccion = startPoint.position;
        }

        if(plataforma.transform.position == startPoint.position)
        {
            direccion = endPoint.position;
        }
    }
}
```

Mediante el script hice que la plataforma se mueva desde la posición del gameObject del principio hasta la posición del gameObject del final con una velocidad marcada.

Para el fuego, necesitaba que la animación se parará una vez llegado al fotograma final donde el fuego deja de propagarse y luego esperar un tiempo hasta volver a reanudarse la animación.

```
using UnityEngine;
using System.Collections;

public class INTERACCION_FUEGO_COLLIDER : MonoBehaviour
{
    public float delay = 3f; // Tiempo de espera entre el fin de la animación y su reanudación
    private Animator animator; // Referencia al Animator
    private BoxCollider2D boxCollider; // Referencia al BoxCollider2D

    private void Start()
    {
        // Obtener referencia al componente Animator
        animator = GetComponent<Animator>();
        if (animator == null)
        {
            Debug.LogError("No se encontró un Animator en este GameObject.");
        }

        // Obtener referencia al componente BoxCollider2D
        boxCollider = GetComponent<BoxCollider2D>();
        if (boxCollider == null)
        {
            Debug.LogError("No se encontró un BoxCollider2D en este GameObject.");
        }
    }

    // Evento llamado al final de la animación (configurado desde el evento en el Animator)
    public void endEvent()
    {
        if (animator != null)
        {
            Debug.Log("Evento 'endEvent' ejecutado. Pausando animación y desactivando el BoxCollider...");
            StartCoroutine(PausarYReanudar());
        }
    }

    // Corrutina para pausar la animación, desactivar el BoxCollider, esperar y luego reanudar
    private IEnumerator PausarYReanudar()
    {
        animator.speed = 0; // Pausa la animación
        if (boxCollider != null)
        {
            boxCollider.enabled = false; // Desactiva el BoxCollider
        }

        yield return new WaitForSeconds(delay); // Espera el tiempo definido

        if (boxCollider != null)
        {
            boxCollider.enabled = true; // Reactiva el BoxCollider
        }
        animator.speed = 1; // Reanuda la animación
```

Para ello usé la línea de código “yield return new WaitSecond (tiempoClave)” como mencioné anteriormente. También necesitaba que mientras el fuego estuviera inactivo, desactivar su BoxCollider2D y lo hice mediante la línea de código boxCollider.enabled = true or false.

Si has incluido más escenas de las seis mínimas exigidas

Sobre las escenas, he hecho una para el menú de Inicio y otra para los demás menús del principio. El nivel 1 constituye de un total de 4 escenas y el nivel 2 constituye de un total de 2 escenas.

Cabe señalar que en un principio el panel de Inicio lo puse en una escena con los demás menús, pero el personaje al morir, lo primero que vería el jugador sería el panel de inicio, por ello, lo cambié y lo separe de escena. Sin embargo se me dificultó ya que hasta ahora, los cambios de escena han sido mencionados con su número int del Build Profile. Lo conseguí mediar estableciendo en un script que la escena número 4 (la del panel de Inicio) era la principal del juego.