

HELLGATE



ÍNDICE

1. Introducción

1.1 ¿Qué es HellGate?

1.2 ¿Cuál es el Objetivo?

2. Preproducción

2.1 Idea

2.2 Referencias

2.3 Tareas

2.4 Recursos y Herramientas

2.4.1 Software

2.4.2 Recursos Gráficos

2.4.3 Recursos Sonoros

2.5 Guión

2.6 GDD Técnico

2.6.1 Introducción

2.6.2 Mecánicas de Juego

2.6.3 Interfaz

3. Producción

3.1 Programación

3.1.1 Personaje

3.1.2 Enemigos

3.1.3 NPCs

3.1.4 Interfaz

3.1.5 Programación Menor

3.2 Diseño de Nivel

3.2.1 Ciudad

3.2.2 Cementerio

INTRODUCCIÓN

- **¿Qué es HellGate?**

HellGate es un videojuego con estética pixel art, de acción y aventura perteneciente al subgénero “Metroidvania”, con toques de plataformas.

- **¿Cuál es el objetivo?**

El objetivo principal de HellGate es ofrecer un reto a los jugadores más experimentados, además de una historia que atrapará al jugador, queriendo saber más y más sobre cómo se desarrolla la misma.

Mi objetivo es aplicar los conocimientos de programación que he adquirido durante estos dos años para crear una demo jugable con la que demostrar no solo los conocimientos de programación, sino también competencia en ciertos aspectos de diseño, como diseño de nivel y diseño de interfaz.

PREPRODUCCIÓN

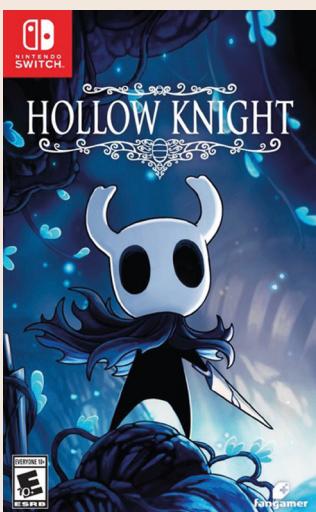
1. Idea

Desde que era pequeño siempre he estado enamorado de los videojuegos, mis primeros recuerdos relacionados con estos son de ser muy pequeño y sentarme con mi padre en su ordenador para verle jugar a juegos como “Diablo”, “Baldur’s Gate”, “Age of Empires”, “Medal of Honor” y otros tantos que ya no recuerdo. El siguiente recuerdo que tengo es de la Navidad en la que me regalaron la GameBoy Advance con el Pokemon Esmeralda, y a partir de ahí la mayoría de los recuerdos más felices que tengo tienen que ver con los videojuegos, por la diversión de jugar con amigos y por las grandes historias que son capaces de transmitir. Además de esto, siempre me han servido de refugio durante épocas difíciles.

Por todo lo que los videojuegos me han dado, quiero dedicarme profesionalmente al desarrollo de videojuegos, y dar el primer paso en esa dirección con un juego del género metroidvania, porque es mi género favorito y junto con los de Pokemon, con los juegos que más he disfrutado siempre.

2. Referencias

Para el desarrollo de HellGate he tenido “Hollow Knight” como principal referencia por ser uno de mis videojuegos favoritos, además de otros juegos como “Castlevania: Symphony of the Night” o “Castlevania: Dawn of Sorrow” por su estética oscura.



3. Tareas

- **GDD Técnico**

El documento donde se explican todos los detalles del juego, para que sirva como guía durante el desarrollo.

- **Selección de Assets**

Como el proyecto lo he querido centrar en la programación, todos los assets los he obtenido de Internet y siguen la estética oscura comentada en el apartado anterior.

- **Programación de control de Personaje**

Siendo un juego de acción-aventura con toques de plataformas, el control del personaje debe ser cómodo para el jugador.

- **Programación de IA de enemigos**

Los enemigos deben poder moverse y atacar al personaje con sus diferentes ataques.

- **Diseño de nivel**

Diseñar el camino que el personaje debe seguir para llegar al final del nivel, teniendo en cuenta la situación de los enemigos y los distintos elementos con los que el jugador podrá interactuar Selección de Assets.

- **Diseño de interfaz**

Estructurar la interfaz para que sea clara e intuitiva para el jugador.

4. Recursos y Herramientas

- **Sofware**

La herramienta principal que he usado es Unity 2018.3.13f1 junto con Visual Studio 2017, además de Adobe Illustrator y Adobe Photoshop de manera puntual.



- **Recursos Gráficos**

- Gothicvania Collection by Ansimuz
- Gothicvania Town by Ansimuz
- Gothicvania Cemetery by Anismuz
- Golden Coin, Rotate Sequence by VectorPixelStar
- 7Soul's RPG Graphics – UI Pack by 7Soul
- Pixel Font Pack by mattWalk

- **Recursos Sonoros**

- 8bit Tunes 8-pack by CactusBear

5. Guión

El protagonista de HellGate es Garm, un cazarrecompensas que ha sido contratado por la gente de la aldea de Ara para que mate a un demonio que está acechando en el cementerio de la aldea. En su camino a completar la misión que le ha sido encomendada, un fantasma llamado Corvus se le aparece para advertirle que no es una buena idea matar a ese demonio, pero Garm ignora el consejo del fantasma y sigue con su misión.

Una vez destruido el demonio, Corvus vuelve a aparecer para decirle a Garm que la vida de ese demonio estaba enlazada a las puertas del infierno y en consecuencia liberando a multitud de demonios y bestias infernales.

6. GDD Técnico

1. Introducción

- **Concepto del Juego**

HellGate es un videojuego en el que controlamos a un cazarrecompensas que ha sido contratado en una aldea para acabar con un demonio que acecha en el cementerio y perturba a los aldeanos.

- **Características Principales**

- **Historia Interesante:** La historia atrapará al jugador queriendo saber más de ésta a medida que va avanzando en el juego.

- **Nivel de dificultad:** HellGate va a ser un juego desafiante para el jugador, convirtiéndolo en un reto.

- **Mundo Extenso:** Como todo ‘Metroidvania’, HellGate tendrá lugar en un extenso mundo que el jugador podrá explorar a lo largo de la historia.

- **Género**

HellGate pertenece al subgénero “Metroidvania”, es un juego de acción-aventura con toques tanto de plataformas como de RPG, cuyo mundo es extenso y podrá ser explorado a medida que el jugador obtenga nuevas habilidades durante el transcurso de la historia.

- **Público Objetivo**

HellGate está dirigido a personas entre 16 y 30 años a la que le gusta jugar a juegos de acción-aventura con temática oscura que le ofrecen un reto de habilidad.

- **Jugabilidad**

- **Movimiento:** El jugador tendrá el control del personaje para moverlo en 'scroll lateral', con la posibilidad de saltar, y de atacar para defenderse de los enemigos.

- **Habilidades:** Conforme el jugador avance en la historia y derrote a ciertos jefes, desbloqueará habilidades que le permitirán avanzar por zonas antes inaccesibles.

- **Estilo Visual**

- El estilo visual de HellGate es medieval y oscuro, a base de pixelart. La principal inspiración para haber adoptado este estilo son los juegos de la saga Castlevania, como Symphony of the Night de NES/PSX, o Dawn of Sorrow de NDS

- **Alcance**

- El alcance es bastante limitado, la única intención es llevar a cabo una demo jugable que tenga lugar en la primera zona de lo que sería el juego completo

2. Mecánicas de Juego

- **Flujo de Juego**

- El jugador deberá avanzar por el primer escenario, por los tejados de las casas para poder llegar al siguiente escenario. Una vez en el siguiente escenario, deberá enfrentarse a diversos enemigos hasta llegar al demonio del final del cementerio.

- **Personajes**

- **Garm:** Es un cazarrecompensas contratado en la aldea de Ara para matar al demonio que acecha el cementerio.



- **Aldeanos:** Son la gente de la aldea de Ara, viven desde hace un tiempo atemorizados por un demonio que merodea por el cementerio.



- **Corvus:** Es un fantasma que se le aparece a Garm para advertirle de que esta tarea que le han encomendado no es buena idea. Aparecerá a lo largo de la historia para aconsejar a Garm en su aventura.



- **Aries:** Es un duro demonio que atemoriza a la aldea y con el que se enfrentará Garm en el cementerio.



- **Perros Infernales:** Enemigo común que Garm encontrará en su camino.



- **Calaveras Infernales:** Enemigo común que Garm encontrará en su camino.



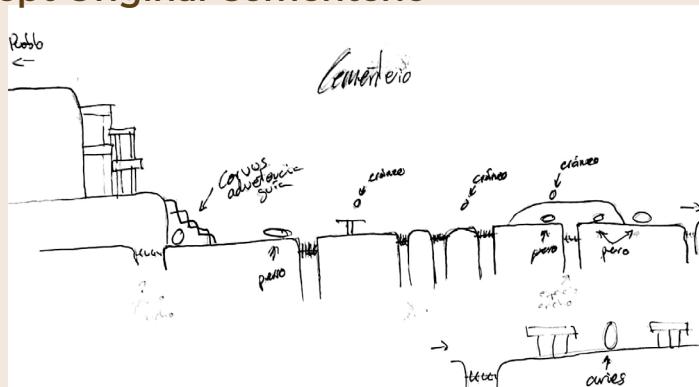
- **Movimiento y Físicas**
 - **Interacción entre elementos**
 - **Personaje - Escenario:** El personaje colisiona con el escenario de manera natural, dado que camina por el suelo.
 - **Personaje - Enemigo:** El personaje al entrar en contacto con un enemigo, recibirá daño, en cambio, los enemigos solo recibirán daño en caso de que el jugador ataque.
 - **Personaje - Hechizos:** El personaje recibirá daño al entrar en contacto con un hechizo, y el hechizo desaparecerá tras esto.
 - **Enemigo - Escenario:** Los enemigos colisionan con el escenario de manera natural, dado que caminan por el suelo.
 - **Hechizo - Escenario:** Al chocar con cualquier parte del escenario, el hechizo desaparecerá.
 - **Controles**
 - **Movimiento lateral:** Flechas Izquierda y Derecha
 - **Saltar:** Tecla Z
 - **Agacharse:** Flecha Abajo
 - **Atacar:** Tecla X
 - **Menú Pausa:** Tecla ESC
 - **Diseño de Nivel**

• Diseño de Nivel

• Cocept Original Pueblo

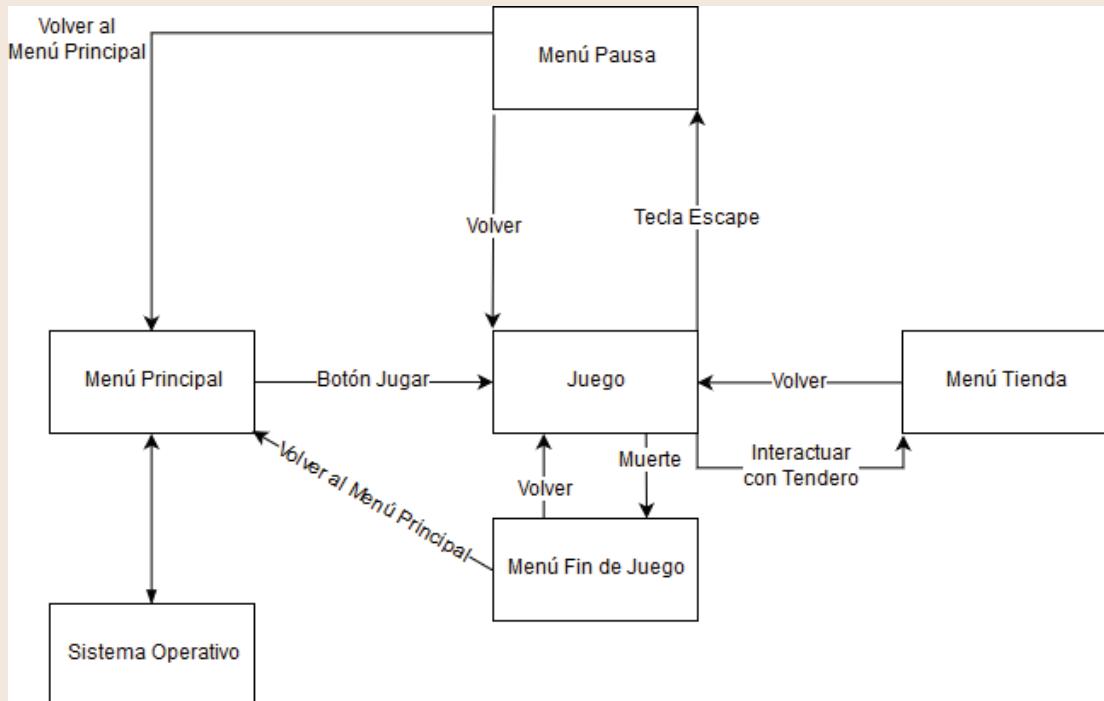


• Cocept Original Cementerio



3. Interfaz

- **Diagrama de Flujo**



- **Menú principal**



- **Jugar:** Se comienza una nueva partida.
- **Sair del Juego:** Se cierra el juego y se vuelve al escritorio.

- **Menú Pausa**



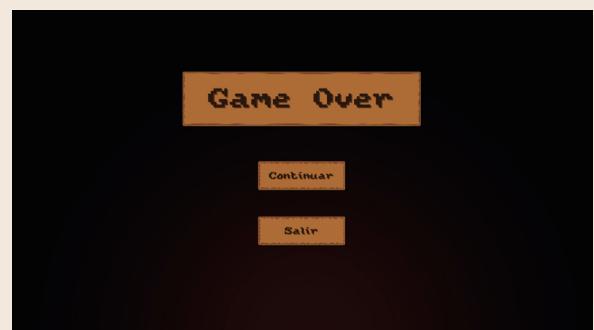
- **Reanudar:** Se reanuda el juego.
- **Salir al Menú:** Se sale de la partida y se vuelve al menú principal.

- **Juego**



- **Interfaz:** Se ve la vida del personaje, y el dinero que va acumulando.

- **Menú Fin de Partida**



- **Continuar:** El juego continuará desde el último guardado.
- **Salir:** Se saldrá al menú principal.

PRODUCCIÓN

1. Programación

- Personaje

- Animaciones:

- Correr: Se ejecuta cuando el jugador se mueve hacia los lados.



- Saltar: Se ejecuta cuando el jugador pulsa el botón de salto.



- Estático: Es la animación por defecto que se ejecuta mientras el personaje se mantenga sin moverse.



- Recibir Daño: Se ejecuta cuando el personaje recibe un golpe.



- Atacar: Se ejecuta cuando el jugador pulsa el botón de ataque.



- Agacharse: Se ejecuta cuando el jugador pulsa el botón abajo.



- Movimiento

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovimientoPersonaje : MonoBehaviour
{
    //VARIABLES

    Rigidbody2D rbPj;
    Animator animPj;

    public float fuerza = 10;
    public float velocidadMax = 7;
    public float fuerzaSalto = 300;
    public bool miraDerecha = true;

    Vector2 direction = Vector2.down;
    public LayerMask layerMask;

    bool tocaSuelo = false;

    public Vector2 mirando = Vector2.right;

    Estadisticas statsPj;

    // Funcion Start en la que se inicializan las variables
    // que dependen de componentes del objeto
    void Start()
    {
        rbPj = GetComponent<Rigidbody2D>();
        animPj = GetComponent<Animator>();
        statsPj = GetComponent<Estadisticas>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        //Si el personaje mira hacia un lado su sprite se gira
        //para que mire hacia ese lado
        if (!statsPj.invulnerable)
        {
            if (Input.GetKey(KeyCode.LeftArrow))
            {
                MueveIzquierda();
                if (!miraDerecha) { GiraSprite(); }
            }

            if (Input.GetKey(KeyCode.RightArrow))
            {
                MueveDerecha();
                if (!miraDerecha) { GiraSprite(); }
            }
        }

        //Si se pulsa la tecla asignada al salto y el personaje
        //está tocando el suelo, se aplica una fuerza vertical al personaje para que salte
        if (Input.GetButtonDown("Salto") && tocaSuelo)
        {
            rbPj.AddForce(Vector2.up * fuerzaSalto);
        }

        //Se llama a la función que gestiona las animaciones del personaje
        GestionAnimacion();

        //Raycast que detecta que el personaje esté tocando el suelo para evitar que el personaje pueda saltar infinitamente hacia arriba sin estar tocando el suelo
        RaycastHit2D hit = Physics2D.Raycast(new Vector3(transform.position.x, (transform.position.y -1.2f), transform.position.z), direction, 0.1f, layerMask);

        if (hit)
        {
            Debug.DrawRay(new Vector3(transform.position.x, (transform.position.y -1.2f), transform.position.z), direction * 0.1f, Color.yellow);
            Debug.Log("Did Hit " + hit.transform.name);
            tocaSuelo = true;
        }
        else
        {
            Debug.DrawRay(new Vector3(transform.position.x, (transform.position.y -1.2f), transform.position.z), direction * 0.1f, Color.white);
            Debug.Log("Did not Hit");
            tocaSuelo = false;
        }
    }

    //Funcion que hace que el personaje se desplace hacia la izquierda
    void MueveIzquierda()
    {
        mirando = Vector2.left;
        if (rbPj.velocity.x > -velocidadMax)
            rbPj.AddForce(Vector2.left * fuerza);
    }
}
```

```

//Función que hace que el personaje se desplace hacia la derecha
void MueveDerecha()
{
    mirando = Vector2.right;
    if (rbPj.velocity.x < velocidadMax)
        rbPj.AddForce(Vector2.right * fuerza);
}

//Función que invierte el eje de rotación Y del personaje para que esté orientado
void GiraSprite()
{
    transform.Rotate(0f, 180f, 0f);
    miraDerecha = !miraDerecha;
}

//Función que gestiona las animaciones
void GestionAnimacion()
{
    //Cuando el personaje se está moviendo o se está pulsando la tecla de movimiento se activa la animación de andar
    if (!statsPj.invulnerable)
    {
        if (Input.GetButton("Horizontal") || rbPj.velocity.x != 0)
        {
            animPj.SetBool("andando", true);

        }
        else
        {
            animPj.SetBool("andando", false);
        }
    }

    // Cuando se pulsa el botón den agacharse se ejecuta la animación de agacharse
    if (Input.GetButton("Agachar"))
    {
        animPj.SetBool("agachado", true);
    }
    else
    {
        animPj.SetBool("agachado", false);
    }

    //Cuando el jugador pulsa el botón de salto y el personaje está tocando el suelo, se ejecuta la animación de saltar
    if (Input.GetButtonDown("Salto") && tocaSuelo)
    {
        animPj.SetTrigger("salto");
    }

    animPj.SetBool("tocandoSuelo", tocaSuelo);
}
}

```

- Combate

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AtaqueHeroe : MonoBehaviour
{
    //Variables
    Animator animPj;
    MovimientoPersonaje mov;
    public LayerMask layerMaskAtaque;

    // Función Start que inicializa las variables que dependen de los componentes del objeto
    void Start()
    {
        animPj = GetComponent();
        mov = GetComponent<MovimientoPersonaje>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        //Si se pulsa el botón de atacar ejecuta la animación de atacar
        if (Input.GetButtonDown("Ataque"))
        {
            animPj.SetTrigger("ataque");
        }
    }
}

```

```

//Esta es la función de ataque, que se ejecuta cada frame de la animación de ataque
public void AnimAtaque()
{
    //Raycast que detecta si se toca a un enemigo
    RaycastHit2D ataque = Physics2D.Raycast(transform.position, mov.mirando, 2f, layerMaskAtaque);

    //en caso de que se toque un enemigo, se comprueba que tipo de enemigo es y se ejecuta su función de recibir daño
    if (ataque)
    {
        Debug.DrawRay(transform.position, mov.mirando * 2f, Color.red);
        Debug.Log("Ataque a " + ataque.transform.name);
        if (ataque.transform.gameObject.GetComponent<CombatePerro>())
        {
            ataque.transform.gameObject.GetComponent<CombatePerro>().RecibeDano();
        }
        else if (ataque.transform.gameObject.GetComponent<CombateCraneo>())
        {
            ataque.transform.gameObject.GetComponent<CombateCraneo>().RecibeDano();
        }
        else if (ataque.transform.gameObject.GetComponent<CombateAries>())
        {
            ataque.transform.gameObject.GetComponent<CombateAries>().RecibeDano();
        }
    }
    else
    {
        Debug.DrawRay(transform.position, mov.mirando * 2f, Color.white);
        Debug.Log("Ataque Fallado");
    }
}
}

```

- Estadísticas

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Estadisticas : MonoBehaviour
{
    //Variables
    public int vidaPj;

    public Sprite fullHeart;
    public Sprite halfHeart;
    public Sprite voidHeart;

    public Image corazon1;
    public Image corazon2;
    public Image corazon3;
    public Image corazon4;
    public Image corazon5;

    public int dineroPj;

    public bool invulnerable;

    Animator animPj;
    Rigidbody2D rb;

    public Animator animMuerte;

    //Al aparecer, los corazones de vida de la interfaz se llenan
    void Awake()
    {
        corazon1.sprite = fullHeart;
        corazon2.sprite = fullHeart;
        corazon3.sprite = fullHeart;
        corazon4.sprite = fullHeart;
        corazon5.sprite = fullHeart;
    }

    //Al empezar se inicializan las variables que dependen de componentes del objeto
    private void Start()
    {
        animPj = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }

    //Función que vuelve al personaje invulnerable al recibir un golpe, y activa la animación de recibir daño del personaje
    public void TiempoInvulnerable()
    {
        invulnerable = true;
        animPj.SetTrigger("golpe");
    }

    //Función que hace que el personaje deje de ser invulnerable, que se ejecuta al terminar la animación de recibir daño
    public void Mortal()
    {
        invulnerable = false;
    }
}

```

```

//Funcion de recibir daño del personaje, los enemigos acceden a la función y le dicen el daño que hacen y la posición del enemigo
public void RecibeDano(int a, Vector3 enemigo)
{
    //Si el personaje no es invulnerable
    if (invulnerable == false)
    {
        if (enemigo.x - transform.position.x < 0)
        {
            rb.AddForce(new Vector2(200f, 200f));
            Debug.Log(transform.name + " Vuela Derecha");
        }
        //se lanza al personaje hacia un lado o hacia otro
        //en función de donde se encuentre el enemigo
    }
    else
    {
        rb.AddForce(new Vector2(-200f, 200f));
        Debug.Log(transform.name + " Vuela Izquierda");
    }
    //Al recibir daño se comprueba la vida del personaje y si es mayor que cero, se resta vida
    //se actualiza la interfaz para que la cantidad de vida esté acorde con los corazones que la representan en la interfaz
    if (vidaPj > 0)
    {
        vidaPj = vidaPj - a;
        if (vidaPj <= 1)
        {
            corazon1.sprite = halfHeart;
            corazon2.sprite = voidHeart;
            corazon3.sprite = voidHeart;
            corazon4.sprite = voidHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 2)
        {
            corazon2.sprite = voidHeart;
            corazon3.sprite = voidHeart;
            corazon4.sprite = voidHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 3)
        {
            corazon2.sprite = halfHeart;
            corazon3.sprite = voidHeart;
            corazon4.sprite = voidHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 4)
        {
            corazon3.sprite = voidHeart;
            corazon4.sprite = voidHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 5)
        {
            corazon3.sprite = halfHeart;
            corazon4.sprite = voidHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 6)
        {
            corazon4.sprite = voidHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 7)
        {
            corazon4.sprite = halfHeart;
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 8)
        {
            corazon5.sprite = voidHeart;
        }
        else if (vidaPj <= 9)
        {
            corazon5.sprite = halfHeart;
        }
        //Tras cambiar la interfaz, se ejecuta la función comentada antes
        TiempoInvulnerable();
    }
    //Si al recibir el golpe el personaje tiene menos de 1 de vida,
    //se cambian los corazones de la interfaz a corazones vacíos y se ejecuta la animación de muerte
    if (vidaPj < 1)
    {
        corazon1.sprite = voidHeart;
        corazon2.sprite = voidHeart;
        corazon3.sprite = voidHeart;
        corazon4.sprite = voidHeart;
        corazon5.sprite = voidHeart;

        animPj.SetTrigger("Muerte");
    }
}

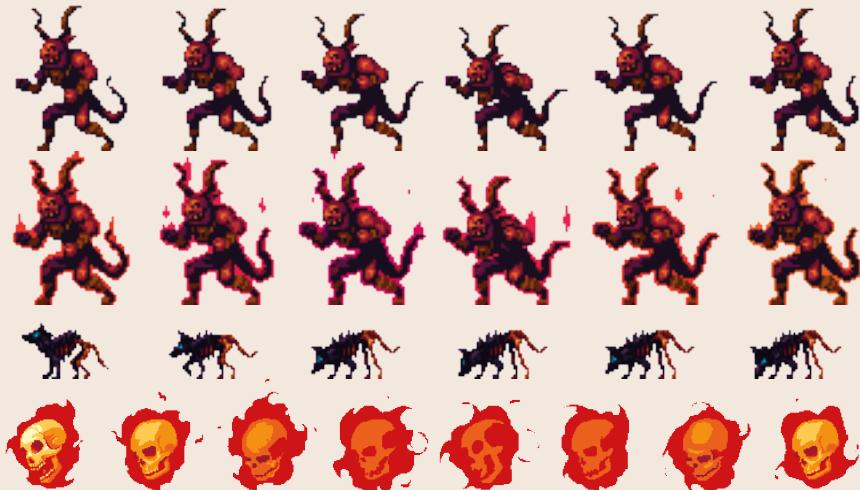
//Cuando termina la animación de muerte se ejecuta esta función que destruye el objeto y hace aparecer la pantalla de muerte
void Muerte()
{
    Destroy(gameObject);
    animMuerte.SetTrigger("activa");
    Time.timeScale = 0;
}

```

- **Enemigos**

- **Animaciones:**

- **Estático :** Se ejecuta mientras los enemigos no se muevan.



- **Correr:** Cuando el enemigo se desplaza hacia el personaje se ejecuta esta animación.



- **Disparo:** Cuando el enemigo va a lanzar una bola de fuego, activa esta animación.



- **Modo Furia:** Cuando el enemigo está por debajo del 50% de vida, se ejecuta esta animación y el enemigo entra en modo furia.



- **Muerte:** Al perder toda la vida un enemigo , se ejecuta la animación.



- Combate Aries

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CombateAries : MonoBehaviour
{
    //Variables
    int vida = 10;
    int dano = 1;
    bool invulnerable = false;

    bool rageMode;

    GameObject player;
    bool persiguePlayer;

    SpriteRenderer spr;
    Rigidbody2D rb;
    Animator anim;

    float orientacion;

    public GameObject prefabBolaFuego;
    GameObject saleFuego;
    GameObject puntoIzq;
    GameObject puntoDch;

    public GameObject corvus;

    public BoxCollider2D borde;

    //Función Start en la que se inicializan las variables que dependan de componentes de objetos
    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();

        //estas variables se inicializan con los objetos de el nombre indicado que existan en la escena
        puntoIzq = GameObject.Find("PuntoFuegoIzq");
        puntoDch = GameObject.Find("PuntoFuegoDch");
    }

    //Cuando el enemigo entra en contacto con el jugador activa la función de recibir daño del mismo
    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            col.gameObject.GetComponent<Estadisticas>().RecibeDano(dano, transform.position);
        }
    }

    //Detecta cuando el personaje entra en el área de visión representada por un collider marcado como trigger
    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            borde.enabled = true;           //Activa un borde que evita que el personaje salga del área de combate
            player = col.gameObject;
            persiguePlayer = true;
            StartCoroutine("PersiguePlayer"); //Activa una función que se ejecuta en paralelo que hace que el enemigo vaya hacia el personaje
            StartCoroutine("LanzaBolas");    //Activa una función que se ejecuta en paralelo que hace que el enemigo dispare bolas de fuego
            Debug.Log("Voy");
        }
    }

    //Cuando el personaje abandona el área comentada antes, detiene las funciones de seguimiento y de disparo
    private void OnTriggerExit2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            persiguePlayer = false;
            StopAllCoroutines();
        }
    }

    //Función que se ejecuta cada vez que el personaje alcanza con su arma al enemigo
    public void RecibeDano()
    {
        //si el enemigo no es invulnerable, entonces se le resta un punto de vida
        //se le hace invulnerable durante un pequeño intervalo de tiempo para evitar que un solo ataque interactúe varias veces
        //se cambia el color a rojo para que el jugador sepa que ha conectado el golpe
        if (invulnerable == false)
        {
            vida--;
            invulnerable = true;
            spr.color = Color.red;
            Debug.Log("Golpe Recibido");
            //Si la vida baja de 0, entonces se ejecuta la animación de muerte
            if (vida <= 0)
            {
                anim.SetTrigger("Muerte");
            }
        }
    }
}
```

```

        //Si la vida del enemigo baja hasta 5, activa el modo furia del enemigo
        //aumentando el daño que inflinge a 2 puntos, y activando las animaciones del modo furia
        if (vida <= 5 && !rageMode)
        {
            rageMode = true;
            dano = 2;
            anim.SetBool("RageMode", true);
        }
        Invoke("Mortal", 0.6f);
    }

}

//Cuando se termina la animación de muerte, se deshabilita el borde
//que evita que el personaje salga, activa la posibilidad de interactuar con Corvus,
//y destruye el objeto
void Muerte()
{
    borde.enabled = false;
    corvus.GetComponent<BoxCollider2D>().enabled = true;
    Destroy(gameObject);
}

//Función que devuelve a la normalidad el enemigo
//Vuelve a ser vulnerable y el sprite recupera sus colores originales
public void Mortal()
{
    spr.color = Color.white;
    invulnerable = false;
}

//Función que activa la animación de disparo del enemigo
void AnimLanzaBolas()
{
    anim.SetTrigger("Disparo");
}

//Función que se ejecuta al final de la animación de disparo
//que crea una bola de fuego que se desplaza horizontalmente en la dirección hacia la que esté el enemigo mirando
public void DisparaBolas()
{
    GameObject bolaFuego;
    DanoBolaFuego bolaScript;
    bolaFuego = Instantiate(prefabBolaFuego, saleFuego.transform.position, new Quaternion(0,0,0,0));
    bolaScript = bolaFuego.GetComponent<DanoBolaFuego>();
    bolaScript.dano = dano;
    bolaScript.orientacion = orientacion;

    Destroy(bolaFuego, 2f);
}

//Corutina que hace que el enemigo persiga al personaje teniendo en cuenta la posición del mismo
//para ir en una dirección u otra
IEnumerator PersiguePlayer()
{
    while (true)
    {
        while (!invulnerable)
        {
            Debug.Log("en camino");
            if (player.transform.position.x - transform.position.x > 0)
            {
                orientacion = 1;
                saleFuego = puntoDch;
                Debug.Log("dch");
                if (rb.velocity.x < 0.8f)
                {
                    rb.AddForce(Vector2.right * 9f);
                    spr.flipX = true;
                }
            }
            else if (player.transform.position.x - transform.position.x < 0)
            {
                Debug.Log("izq");

                if (rb.velocity.x > -0.8f)
                {
                    orientacion = -1;
                    saleFuego = puntoIzq;
                    Debug.Log("fuerza izq");
                    rb.AddForce(Vector2.left * 9f);
                    spr.flipX = false;
                }
            }
            yield return null;
        }
        yield return null;
    }
}

```

```

//Corutina que cada intervalo de 3 a 6 segundos activa la animación de disparar
IEnumerator LanzaBolas()
{
    while (true)
    {
        AnimLanzaBolas();
        yield return new WaitForSeconds(Random.Range(3, 6));
    }
}

```

- Combate Perro Infernal

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CombatePerro : MonoBehaviour
{
    //Variables
    public int vida;
    public int dano;
    bool invulnerable = false;

    GameObject player;
    SpriteRenderer spr;

    Rigidbody2D rb;
    Animator anim;
    //Función Start en la que se inicializan las variables que dependan de componentes del objeto
    void Start()
    {
        spr = GetComponent<SpriteRenderer>();
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
    }

    //Cada frame, se pasa el valor absoluto de la velocidad del enemigo a su animator
    //y en caso de que sea superior a 0, ejecuta la animación de correr
    void FixedUpdate()
    {
        anim.SetFloat("velocidad", Mathf.Abs(rb.velocity.x));
    }

    //Cuando el enemigo entra en contacto con el jugador activa la función de recibir daño del mismo
    void OnCollisionEnter2D(Collision2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            col.gameObject.GetComponent<Estadisticas>().RecibeDano(dano, transform.position);
        }
    }

    //Detecta cuando el personaje entra en el área de visión representada por un collider marcado como trigger
    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            player = col.gameObject;
            StartCoroutine("PersiguePlayer");           ///Activa una función que se ejecuta en paralelo que hace que el enemigo vaya hacia el personaje
            Debug.Log("Voy");
        }
    }

    //Cuando el personaje abandona el área comentada antes, detiene la función de seguimiento
    private void OnTriggerExit2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            StopCoroutine("PersiguePlayer");
        }
    }

    //Función que se ejecuta cada vez que el personaje alcanza con su arma al enemigo
    public void RecibeDano()
    {
        //si el enemigo no es invulnerable, entonces se le resta un punto de vida
        //se hace invulnerable durante un pequeño intervalo de tiempo para evitar que un solo ataque interactúe varias veces
        //se le aplica una fuerza que empuja al enemigo en dirección contraria al personaje
        if (invulnerable == false)
        {
            vida--;
            Debug.Log("Golpe Recibido");
            //Cuando la vida baja de 0, activa la animación de muerte
            if (vida <= 0)
            {
                anim.SetTrigger("Muerte");
            }
        }
    }
}

```

```

        if (player.transform.position.x - transform.position.x < 0)
        {
            rb.AddForce(new Vector2(300f,200f));
            Debug.Log(transform.name + " Vuela Derecha");
        }
        else
        {
            rb.AddForce(new Vector2(-300f, 200f));
            Debug.Log(transform.name + " Vuela Izquierda");
        }
    }
    Invoke("Mortal",0.6f);
}

}

//Función que destruye el enemigo al terminar la animación de muerte
void Muerte()
{
    Destroy(gameObject);
}

//Función que hace volver a ser vulnerable al enemigo
public void Mortal()
{
    invulnerable = false;
}

//Corrutina que hace que el enemigo persiga al personaje teniendo en cuenta la posición del mismo
//para ir en una dirección u otra
IEnumerator PersiguePlayer()
{
    while (true)
    {
        while (!invulnerable)
        {
            Debug.Log("en camino");
            if (player.transform.position.x - transform.position.x > 0)
            {
                Debug.Log("dch");
                if (rb.velocity.x < 4f)
                {
                    rb.AddForce(Vector2.right * 9f);
                    spr.flipX = true;
                }

            }
            else if (player.transform.position.x - transform.position.x < 0)
            {
                Debug.Log("izq");

                if (rb.velocity.x > -4f)
                {
                    Debug.Log("fuerza izq");
                    rb.AddForce(Vector2.left * 9f);
                    spr.flipX = false;
                }
            }
            yield return null;
        }
        yield return null;
    }
}
}

```

- Combate Calavera Infernal

El script de combate de la calavera es exactamente igual que el del perro, salvo que al tener solo un punto de vida, al recibir un golpe muere de forma instantánea y no recibe fuerza que lo empuje como sí ocurre con el perro.

- **NPCs**

- **Animaciones:**

- **Estático :** Es la animación por defecto de los NPCs.



- **Aparición y desaparición:** Corvus tiene estas dos animaciones que se ejecutan al aparecer y desaparecer frente al jugador.



- **Gestor de Conversaciones**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GestorConversacion : MonoBehaviour
{
    //Variables
    public Animator panel;
    Animator anim;
    public Animator finJuego;

    //Función Start en la que se inicializan las variables que dependan de componentes de objetos
    void Start()
    {
        //Esto solo ocurre en el caso de Corvus, ya que los aldeanos no tienen un componente Animator
        anim = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    //Al entrar el personaje en el área definida con un trigger
    //se activa la animación que hace aparecer la caja de texto correspondiente a cada personaje
    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            //En caso de Corvus también se activa su animación de aparición
            if (anim != null)
            {
                anim.SetBool("Aparece", true);
            }
            panel.SetBool("activo", true);
        }
    }
}
```

```

//Al salir el personaje del área definida con un trigger
//se activa la animación que hace desaparecer la caja de texto correspondiente a cada personaje
void OnTriggerExit2D(Collider2D col)
{
    if (col.gameObject.CompareTag("Player"))
    {
        //En caso de Corvus también se activa su animación de desaparición
        if (anim != null)
        {
            anim.SetBool("Aparece", false);
        }
        panel.SetBool("activo", false);
    }
}

//En el caso de Corvus, al terminar su animación de desaparición
//se activa esta función que borra el objeto para que no vuelva a aparecer
void Destruye()
{
    //En caso de que sea el último Corvus, que aparece una vez el personaje ha derrotado a Aries
    //Se para el juego y aparece la pantalla de agradecimientos
    if (finJuego != null)
    {
        Time.timeScale = 0;
        finJuego.SetTrigger("activa");
    }
    Destroy(gameObject);
}
}

```

- **Interfaz**
 - Menú Pausa

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class UIControl : MonoBehaviour
{
    bool pausa = false;
    public Animator menuPausa;

    //Se detecta cuando se pulsa la tecla "Escape"
    //y activa la función Pausa, que detiene el tiempo
    //y hace aparecer el menú de pausa en pantalla
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            Pausa();
        }
    }
    public void Pausa()
    {
        pausa = !pausa;
        menuPausa.SetBool("activo", pausa);
        if (pausa)
        {
            Time.timeScale = 0f;
        }
        else if (!pausa)
        {
            Time.timeScale = 1f;
        }
    }
}

```

El botón de reanudar ejecuta también la función Pausa() de este script para que se reanude el juego tanto pulsando el botón como volviendo a pulsar la tecla “Escape”

- Interfaz InGame



La interfaz InGame está gestionada por el script de las estadísticas del personaje, como se explica en su apartado correspondiente.

• Programación Menor

- Control de Cámara

```
using UnityEngine;
using System.Collections;

public class CompleteCameraController : MonoBehaviour
{
    public GameObject player;           //Variable para guardar el gameObject del personaje

    private Vector3 offset;            //Variable para guardar la distancia entre la cámara y el jugador

    // Se inicializa el valor
    void Start()
    {
        //Calcula y guarda la distancia entre el jugador y la cámara
        offset = transform.position - player.transform.position;
    }

    //Función que se repite cada frame
    void LateUpdate()
    {
        //La cámara sigue al personaje, respetando siempre la distancia guardada con anterioridad
        if (player != null)
        {
            transform.position = player.transform.position + offset;
        }
    }
}
```

- Control de Escenas

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

public class ControlEscenas : MonoBehaviour
{
    //Estas funciones están asignadas a los botones
    //que aparecen en los menús

    //Función que carga la escena de Juego
    public void ComienzaPartida()
    {
        SceneManager.LoadScene(1);
        if (Time.timeScale == 0f)
        {
            Time.timeScale = 1f;
        }
    }

    //Función que carga la escena del Menú Principal
    public void MenuPrincipal()
    {
        SceneManager.LoadScene(0);
        if (Time.timeScale == 0f)
        {
            Time.timeScale = 1f;
        }
    }

    //Función que cierra el juego
    public void CerrarJuego()
    {
        Application.Quit();
    }
}
```

- Control de Escenarios

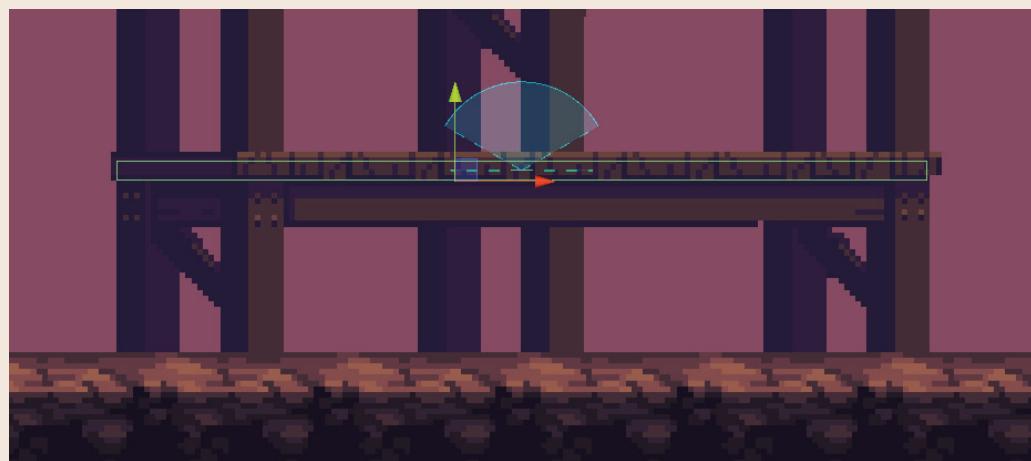
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ControlFondos : MonoBehaviour
{
    public Animator animFondo;

    //Cuando el personaje entra en el área del cementerio
    //delimitada por un trigger, se activa la animación
    //que hace aparecer los fondos del cementerio
    //y desaparecer los de la ciudad. Al salir del área
    //ocurre justo lo contrario desaparecen los del cementerio
    //y aparecen los de la ciudad
    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            animFondo.SetBool("cementerio", true);
        }
    }

    private void OnTriggerExit2D(Collider2D col)
    {
        if (col.gameObject.CompareTag("Player"))
        {
            animFondo.SetBool("cementerio", false);
        }
    }
}
```

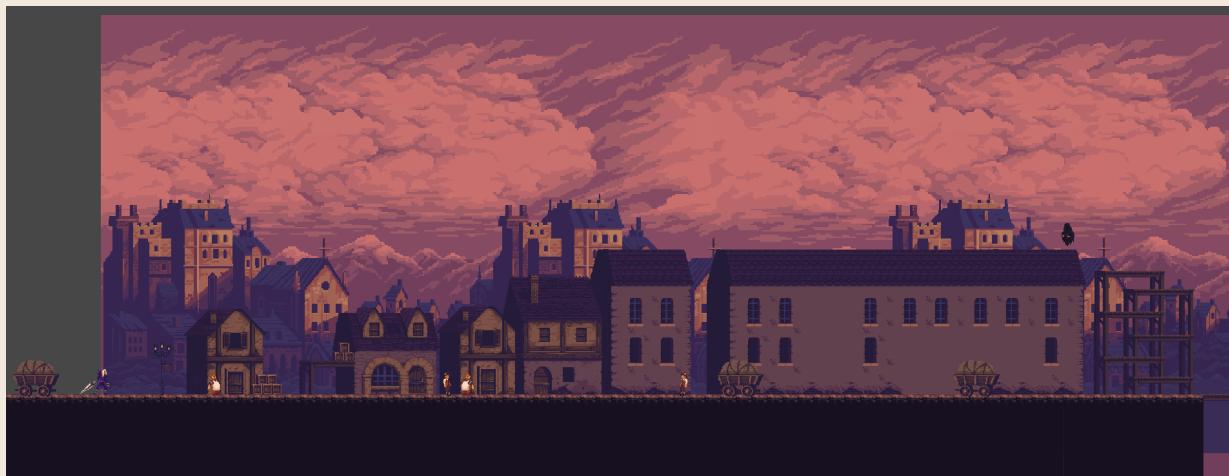
- Plataformas



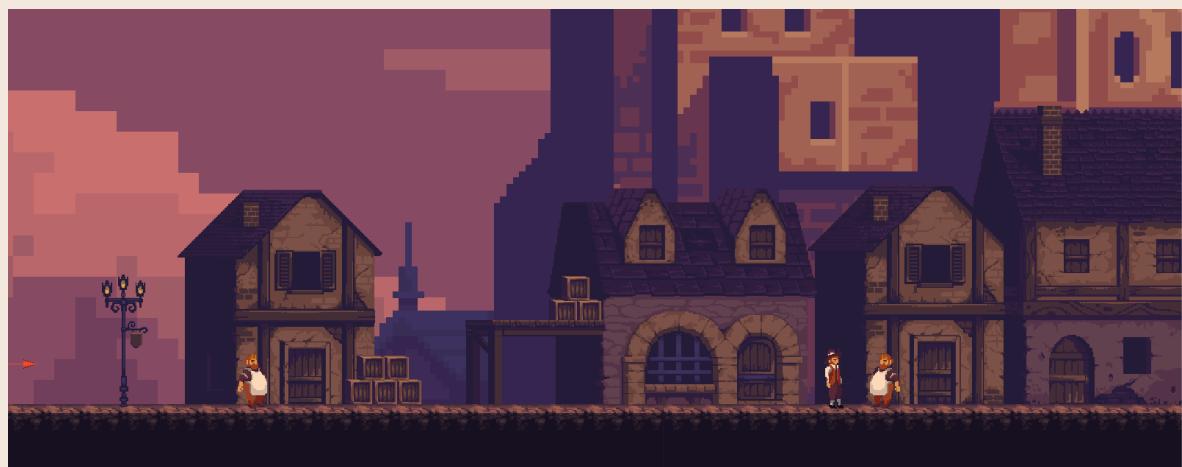
Para crear las plataformas flotantes he usado el componente Platform Effector para que los colliders solo sean sólidos si el personaje lo impacta desde el ángulo indicado, y los pueda atravesar libremente por los lados.

2. Diseño de Nivel

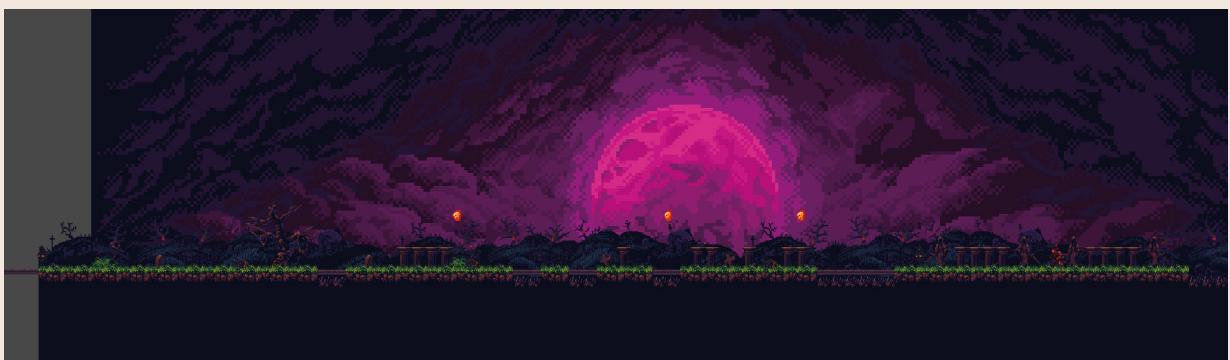
- Ciudad



El personaje aparece inicialmente en la parte izquierda, si sigue avanzando por el suelo hacia la derecha, se encontrará los NPCs de la aldea. El primero le dirá que siga adelante para llegar al cementerio, pero si sigue encontrará que el camino está cortado y un NPC le dirá que tiene que subir por los tejados. Si vuelve al principio, el jugador encontrará una caja donde se puede subir para subir a un andamio y así acceder a los tejados. Siguiendo por los tejados, al final aparecerá Corvus por primera vez, tras esto, si se baja y se sigue recto, se llegará al cementerio.



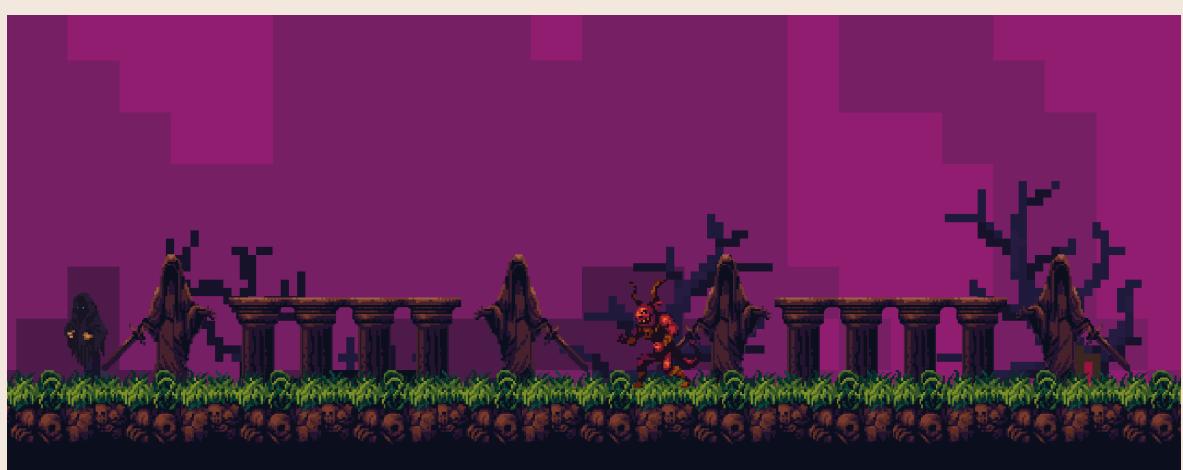
• Cementerio



Tras cruzar el puente, el jugador llegará al cementerio, justo al cruzar el puente volverá a aparecer Corvus. A partir de aquí comienzan a aparecer los enemigos. Siguiendo un poco hacia adelante el jugador encontrará un perro infernal, en el siguiente tramo habrá una plataforma en el aire para poder atacar al cráneo que sobrevuela al personaje, pero deberá tener cuidado porque hay otro perro más en el suelo. Si sigue avanzando verá que hay un cráneo solo esta vez, pero más adelante habrá un perro y otro cráneo más, para que si el jugador trata de ignorar el segundo cráneo se tenga que enfrentar a una amenaza mayor de la esperada.

Al final de la zona se encuentra Aries, el jefe final, y al empezar la pelea el jugador no podrá volver atrás hasta haberlo derrotado. A ambos lados del terreno hay plataformas a la altura perfecta para que si el jugador se siente acorralado pueda subir encima y cruzar al otro lado sin riesgo de recibir daño.

Una vez derrotado aparecerá Corvus y al desaparecer terminaría la partida.



CONCLUSIONES

No creo que el resultado del proyecto haya sido el deseable por diversos factores, pero la falta de organización es el principal. Al no haberme podido dedicar exclusivamente a este proyecto durante el curso, por el cansancio y la desmotivación y no haberme organizado bien, he ido posponiendo este trabajo hasta que ha quedado poco tiempo. Otro de los factores sería el que la parte gráfica haya dependido de assets que he buscado en internet, ya que he tenido que adaptarme a las posibilidades que estos me han ofrecido y amoldar el trabajo a los recursos disponibles (un claro ejemplo es que el jefe final no tiene animación de andar cuando se desplaza hacia los lados).

A pesar de todo esto, estoy contento con el proyecto porque creo que el resultado es más que aceptable, he cumplido con el objetivo que tenía en mente, y además he ampliado mis conocimientos de programación y Unity mientras trabajaba, ya que he tenido que investigar sobre algunos temas para poder llevar a cabo el proyecto. Por supuesto también he aprendido de los errores comentados anteriormente y ahora tengo mucho más claro que para poder sacar adelante un proyecto de la manera correcta, la organización y la constancia son vitales.

