Programación de Videojuego 3D en Unity

Nombre del videojuego: Daredevil

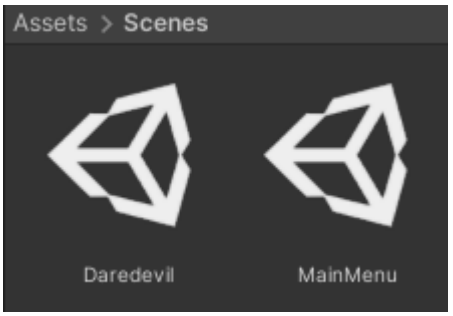
Juan David Lectamo Caicedo
Johann Andrei Ocampo Torres

Profesor: Javier Martin Moreno

Universidad de Huelva
2022

Preconfiguración

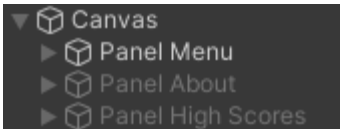
El primer paso que se realizó fue iniciar un proyecto en Unity 3D. Después se crearon dos Scenes, uno para el menú principal y otro para el juego en sí. Esta división se hizo con la intención de tener un mejor control de los componentes del juego, debido a que el menú principal sólo cuenta con componentes 2D.



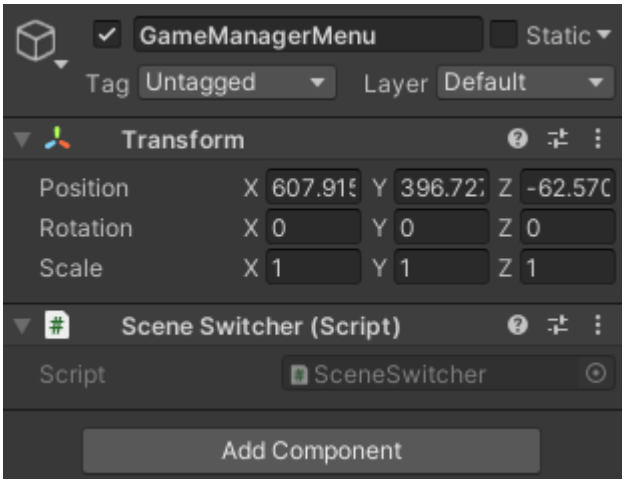
El siguiente paso fue crear el menú principal del juego.

Menú Principal

Para el menú principal se agregó un Canvas que contendría los paneles Menu, About y HighScores.



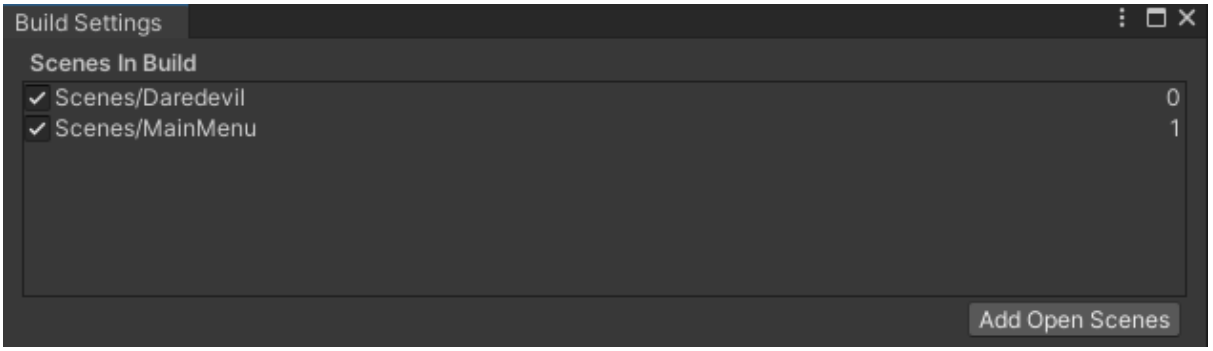
Además de esto, se creó un GameObject vacío con la intención de que este sirviera como objeto para navegar entre Scenes.



A este GameObject se le agregó un script llamado Scene Switcher que contiene la siguiente lógica:

```
1   using UnityEngine;
2   using UnityEngine.SceneManagement;
3
4   public class SceneSwitcher : MonoBehaviour {
5
6       public void OpenScene(int index) {
7           SceneManager.LoadScene(index);
8       }
9   }
```

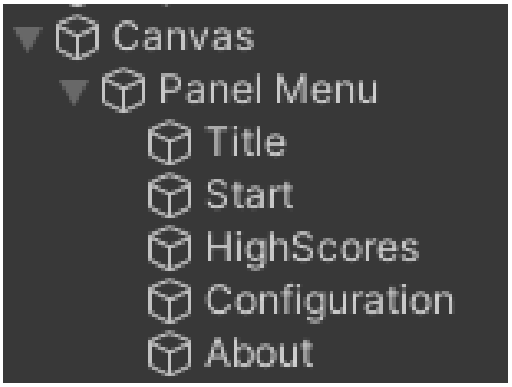
Su funcionamiento es sencillo, recibe un int index de una Scene y la carga. Para saber el index de cada Scene, se tiene que navegar al menú Build Settings y agregar la Scene que se quiere.



Una vez tenemos los índices de Cada Scene solo hace falta llamar al método e indicarle qué Scene se desea cargar.

Panel Principal

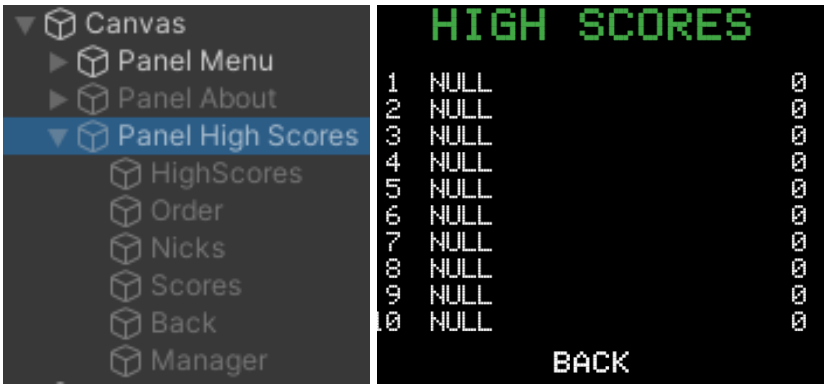
Este Panel contiene varios TextMesh Pro. El primero muestra el título del juego, los siguientes contienen un componente Button para navegar entre paneles.



Además, este es el panel que se muestra por defecto al iniciar el juego y el panel encargado de cargar la Scene del juego.

Panel HighScores

Este Panel también contiene varios TextMesh Pro. El primero es el título, Order muestra la numeración de 1 a 10, Nicks muestra los nicknames de los jugadores ordenados según su puntuación, y Scores muestra el puntaje de su respectivo jugador. Back cuenta con un componente Button que permite navegar al Panel Principal.



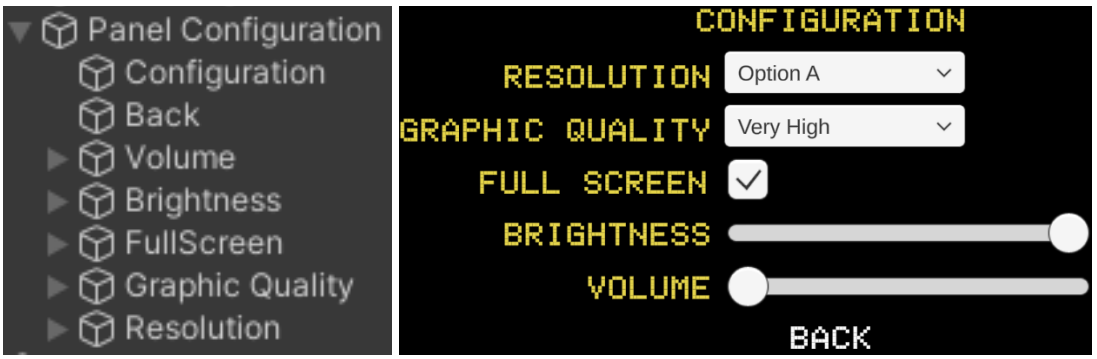
Finalmente, existe un GameObject llamado Manager que contiene un script encargado de rellenar la tabla de NickNames y Scores.

Panel About

Este Panel también contiene varios TextMesh Pro. El primero muestra el título del Panel, el siguiente los nombres de los creadores del juego y finalmente Back que tiene agregado un componente Button que permite regresar al Panel Principal.

Panel Configuración

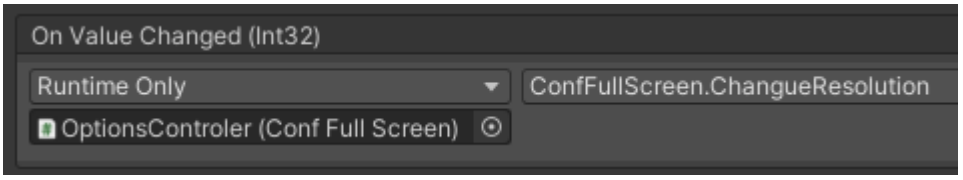
Este Panel es diferente a los anteriores, pues este contiene diferentes elementos, y cada uno de ellos permite realizar varias acciones. Lo primero son los TextMesh Pro que indican el título configuration, resolution, graphic quality, Full Screen, Brightness, volume y finalmente Back que contiene un componente Button que permite regresar.



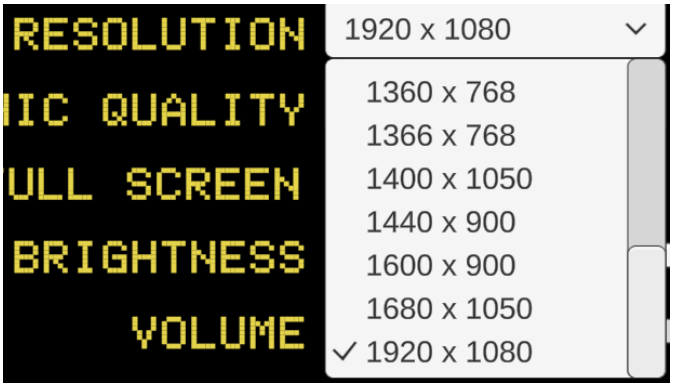
Resolución

Para poder elegir entre las resoluciones posibles se insertó un DropDown y se creó un Script que llena este DropDown.

```
public void CheckResolution() {
    resolutions = Screen.resolutions;
    resolution.ClearOptions();
    List<string> options = new List<string>();
    int currentResolution = 0;

    for (int i = 0; i < resolutions.Length; i++) {
        string option = resolutions[i].width + " x " + resolutions[i].height;
        options.Add(option);

        if (Screen.fullScreen && resolutions[i].width == Screen.currentResolution.width &&
            resolutions[i].height == Screen.currentResolution.height) {
            currentResolution = i;
        }
    }

    resolution.AddOptions(options);
    resolution.value = currentResolution;
    resolution.RefreshShownValue();

    resolution.value = PlayerPrefs.GetInt("numResolution", 0);
}
```

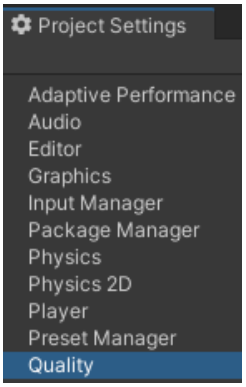


Y una vez se está corriendo el juego, el script ConfFullScreen rellena el DropDown y nos muestra una lista de resoluciones como la siguiente:

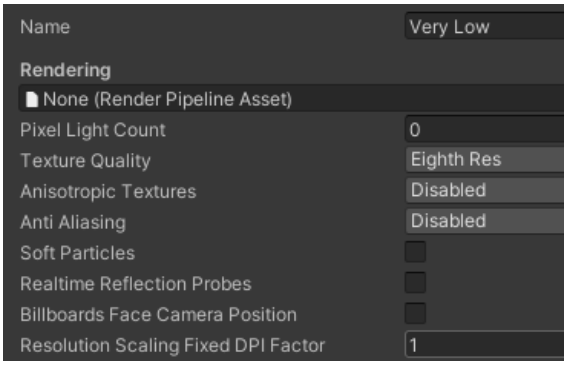


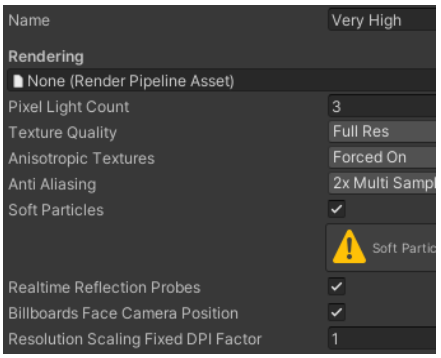
Calidad de Imagen

Para poder elegir entre las diferentes calidades gráficas se insertó un DropDown y se creó un Script que realiza estos cambios.
Además, en el menú Project Settings/Quality se modificaron los parámetros de las diferentes calidades por defecto con las que cuenta unity.



Por ejemplo, aquí vemos las diferencias entre las calidades Very Low and Very High.





El Script ConfQuality creado selecciona entre las opciones de calidad anteriores y guarda la opción seleccionada para la siguiente vez que el juego se inicie.

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

public class ConfQuality : MonoBehaviour {

    public TMP_Dropdown dropdown;
    public int quality;

    void Start() {
        quality = PlayerPrefs.GetInt("quality", 4);
        dropdown.value = quality;
        AdjustQuality();
    }

    void Update() {

    }

    public void AdjustQuality() {
        QualitySettings.SetQualityLevel(dropdown.value);
        PlayerPrefs.SetInt("quality", dropdown.value);
        quality = dropdown.value;
    }
}
```

En esta ocasión, sí se llenó de manera automática cada opción en el DropDown.

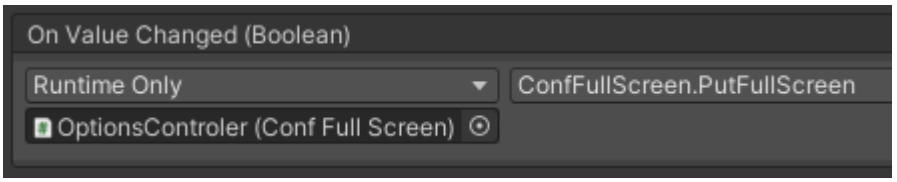
Pantalla Completa

Para seleccionar entre sí queremos el juego en modo ventana o en modo pantalla completa se insertó un toggle.

Este tiene asociado el mismo Script ConfFullScreen que busca las resoluciones, y la parte de comportamiento del toggle es la siguiente:

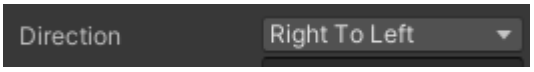
```
public class ConfFullScreen : MonoBehaviour {

    public Toggle toggle;
    public TMP_Dropdown resolution;
    Resolution[] resolutions;

    void Start() {
        if (Screen.fullScreen) {
            toggle.isOn = true;
        } else {
            toggle.isOn = false;
        }

        CheckResolution();
    }

    void Update() {

    }

    public void PutFullScreen (bool fullScreen) {
        Screen.fullScreen = fullScreen;
    }
```

Y finalmente, se asoció el Script al toggle:

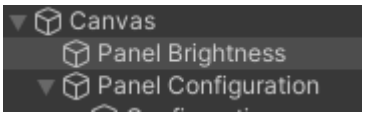


Brillo

Para controlar el Brillo se insertó un Slider con dirección Derecha a izquierda, de manera tal que entre menor fuera el valor del slider más brillo hubiese.



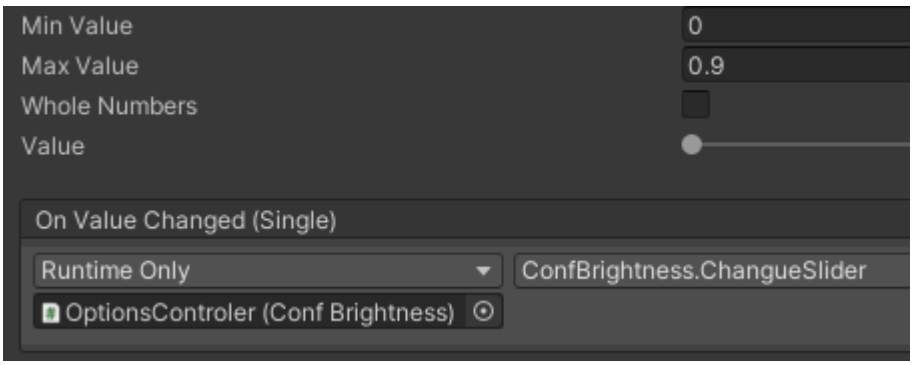
Después, se creó un nuevo Panel Brightness vacío que está por encima de los otros paneles y controlando el color de este panel se simula como si se estuviese aumentando o disminuyendo el brillo.



Además, se creó un script ConfBrightness que vincula el Slider con el color del Panel Brighness.

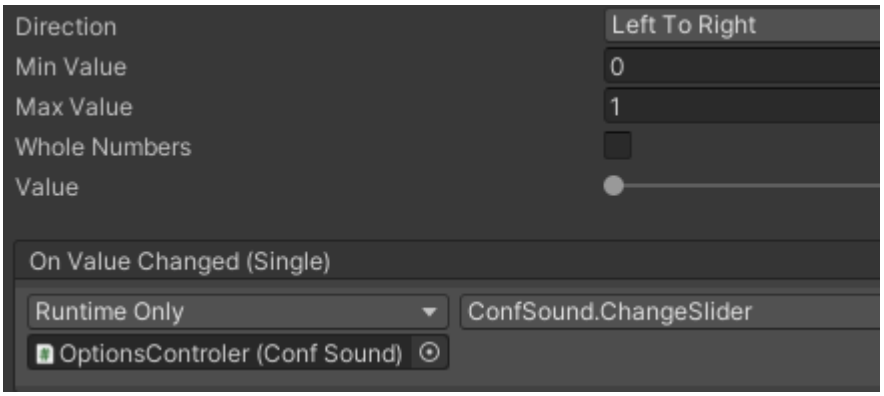
```using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ConfBrightness : MonoBehaviour  {
    public Slider slider;
    public float sliderValue;
    public Image brightness;

    void Start() {
        slider.value = PlayerPrefs.GetFloat("brightness", 0.5f);
        brightness.color = new Color(brightness.color.r, brightness.color.g, brightness.color.b, slider.value);
    }

    void Update() {
    }

    public void ChangueSlider(float value) {
        sliderValue = value;
        PlayerPrefs.SetFloat("brightness", sliderValue);
        brightness.color = new Color(brightness.color.r, brightness.color.g, brightness.color.b, slider.value);
    }
}
```

Finalmente, se impuso un valor máximo para el slider de 0.9 para que la pantalla no se quedara completamente en negro si se arrastraba al maximo el Slider y se asoció el Script al Slider Brighness.



Volumen

Para controlar el volumen se creó un Slider y un Script que lo controla. El Slider va de izquierda a derecha con valores entre 0 y 1, además tiene asociado el Script ConfSound.

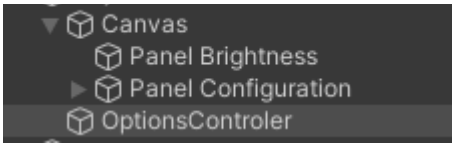


El Script ConfSound tiene la siguiente lógica:

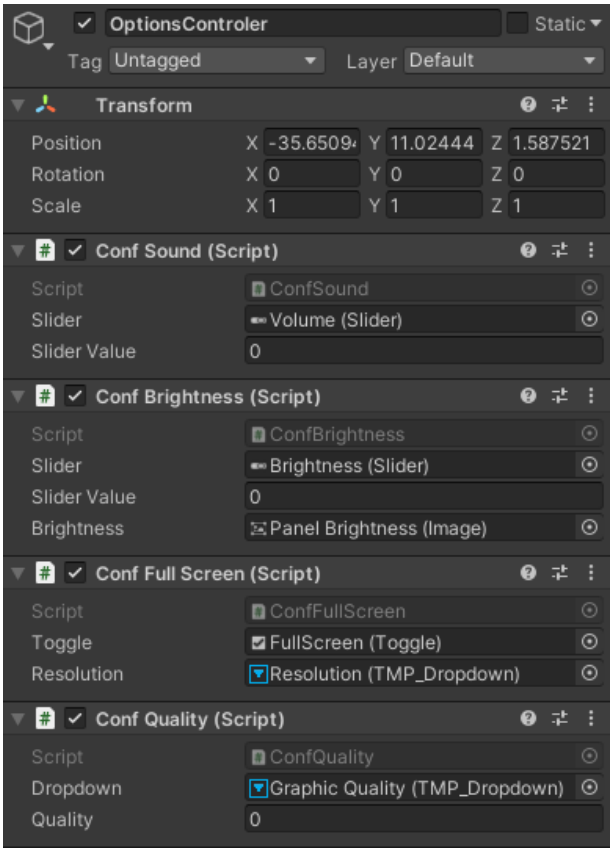
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ConfSound : MonoBehaviour {

    public Slider slider;
    public float sliderValue;

    void Start() {
        slider.value = PlayerPrefs.GetFloat("volume", 0.5f);
        AudioListener.volume = slider.value;
    }

    public void ChangeSlider(float value) {
        sliderValue = value;
        PlayerPrefs.SetFloat("volume", sliderValue);
        AudioListener.volume = slider.value;
    }
}
```

Options Controller

Para controlar todo el Panel de Configuración se creó un GameObject OptionsController vacío que tiene asociados todos los Scripts creados anteriormente.



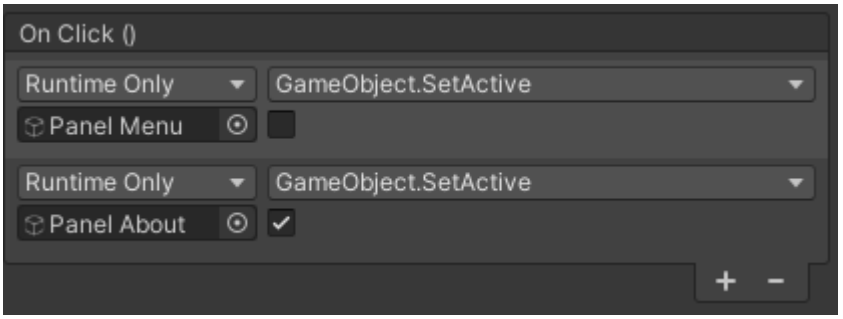
Este OptionsController además de tener asociados los Scripts, tiene asociados los elementos gráficos que cada uno de estos controla, como los Sliders, Toggle, Panel brightness y Quality.



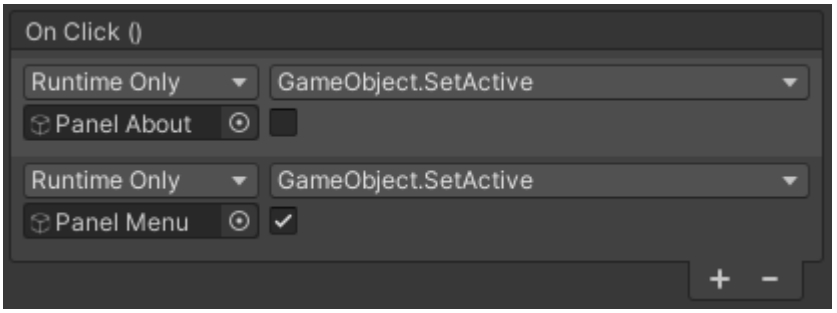
Navegación entre Paneles

Para navegar entre Paneles, se asoció un componente Button a cada TextMesh Pro que indica otro panel o de regreso. Por ejemplo, para este caso el TextMesh Pro About habilita la vista al Panel About. Su funcionamiento es el siguiente:

1. Se deshabilita el Panel Menú (GameObject.SetActive = false)
2. Se habilita el Panel About (GameObject.SetActive = true)



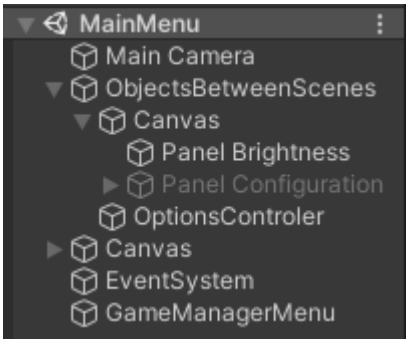
Y para regresar (TextMesh Pro Back en el Panel About) se hace lo contrario, se pone en false en Panel actual y en true el panel al que se quiere navegar.



Menú Pausa

El menú de pausa de llama desde la Scene Daredevil en la esquina superior derecha con un TextMesh Pro que indica pausar el juego,

Para lograr esto, se creó un GameObject ObjectBetweenScenes que contiene los objetos que se van a pasar de una Scene a otra. Estos son: El Panel Brightness, el Panel Configuration y el OptionsController de configuration.

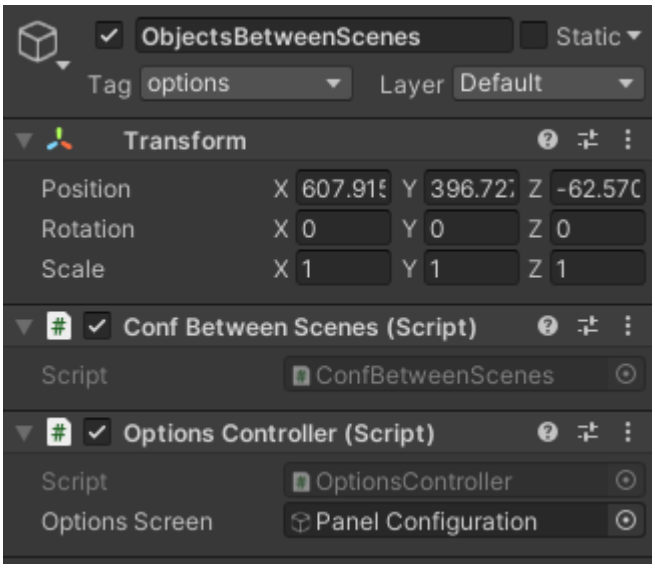


Además, se creó el Script ConfBetweenScenes que permite llamar el Objeto anterior en la Scene Daredevil. El contenido de este Script es el siguiente:

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

public class ConfBetweenScenes : MonoBehaviour {

    private void Awake() {
        var dontDestroyBetweenScenes = FindObjectsOfType<ConfBetweenScenes>();
        if (dontDestroyBetweenScenes.Length > 1) {
            Destroy(gameObject);
            return;
        }

        DontDestroyOnLoad(gameObject);
    }

    void Start() {

    }

    void Update() {

    }
}
```

Y se crea otro Script OptionsController vacío.

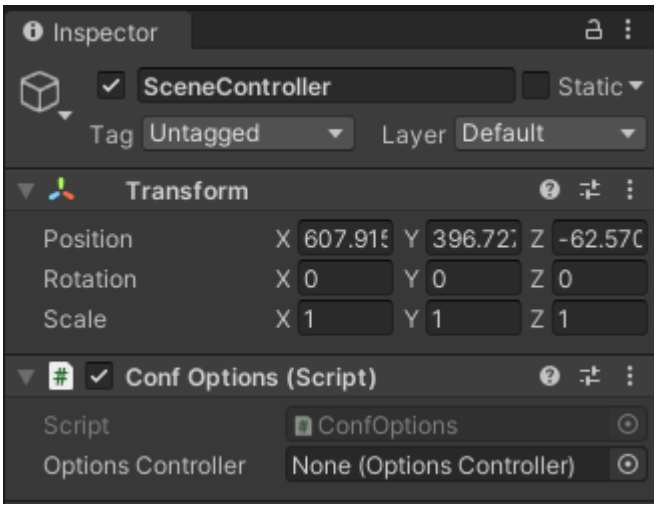
A continuación, en el ObjectBetweenScenes se agregan los siguientes Scripts y se asocia el Panel Configuration al Optionscontroller. Y se le agrega el Tag options para que en la siguiente Scene se pueda encontrar el objeto.



Luego se creó un GameObject SceneController en la Scene Daredevil y se le asoció el Script ConfOptions. El contenido del Script permite ubicar el objeto con el Tag options y mostrarlo, además de permitir pausar el juego con la tecla Esc.

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

public class ConfOptions : MonoBehaviour {

    public OptionsController optionsController;

    void Start() {
        optionsController = GameObject.FindGameObjectWithTag("options").GetComponent<OptionsController>();
    }

    void Update() {
        if (Input.GetKeyDown(KeyCode.Escape)) {
            ShowOptions();
        }
    }

    public void ShowOptions() {
        optionsController.optionsScreen.SetActive(true);
    }
}
```

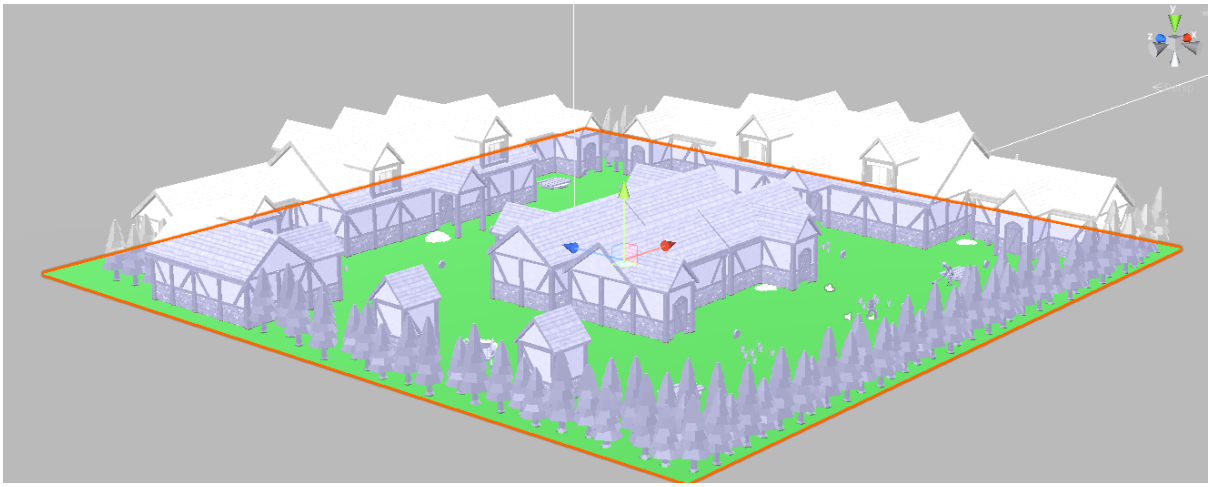
El SceneController tiene asociado un OptionsController que está Nulo pero que al ejecutar el Script se busca y encuentra a través del Tag.



Assets

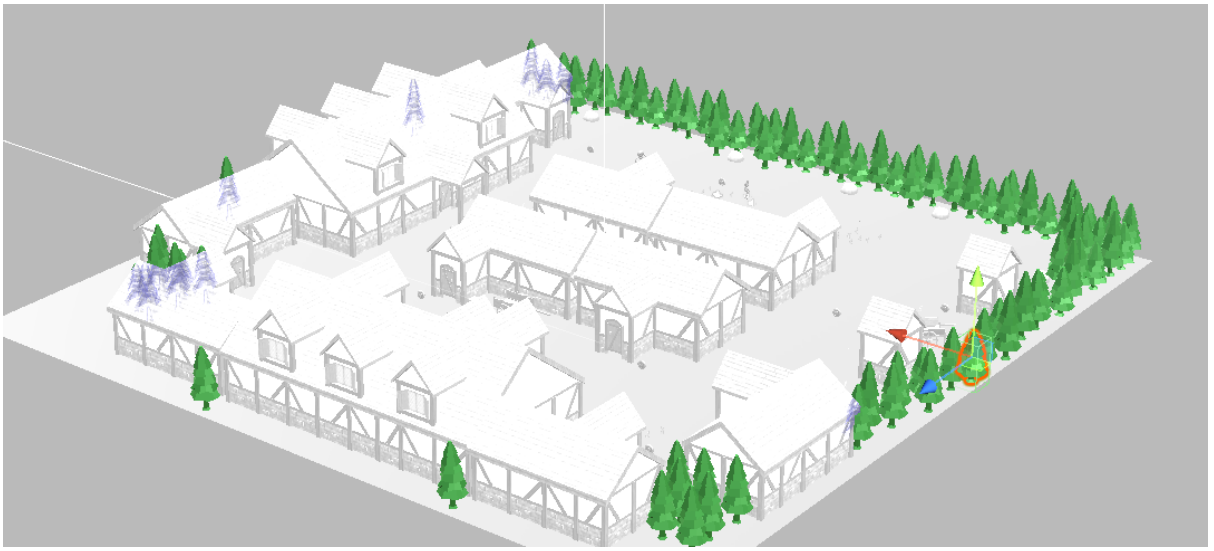
Mapa

El primer paso para el diseño del mapa fue crear un plano 3D en el cual todos los elementos se sostienen. Después de esto se realizó una búsqueda de assets gratis para los elementos que dan vida al mapa. Para esto se utilizó un arte medieval que combinara tanto con la skin de nuestro player como con la de los enemigos.



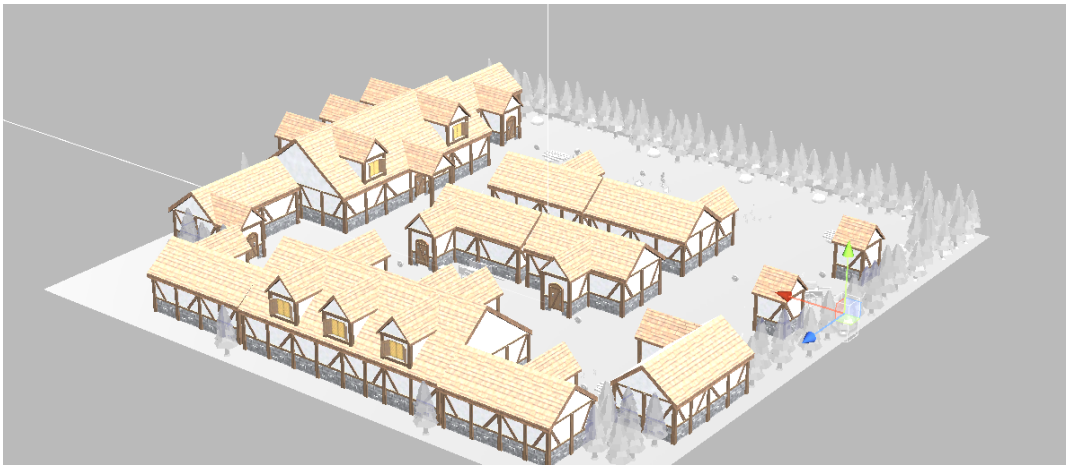
- **Árboles**

  Los árboles además de ser un elemento decorativo también tienen la función de limitar el mapa de tal manera que el jugador no pueda acceder a zonas no deseadas. Estos elementos provienen del asset llamado “Nature Pack - low polly trees & bushes”



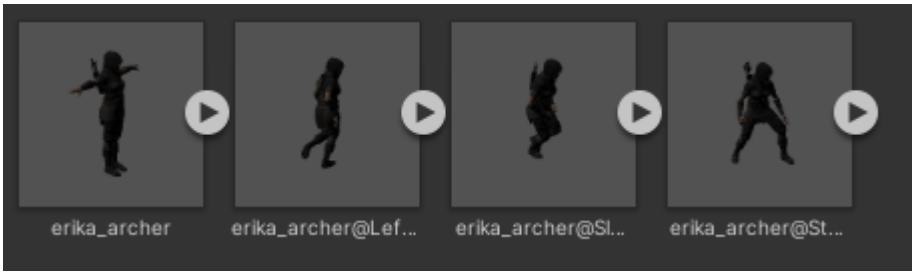
- **Casas**

  Las casas componen la distribución principal del mapa siendo estas las que definen los caminos accesibles por el jugador. Estos elementos provienen del asset llamado “MedievalTownExteriors”.



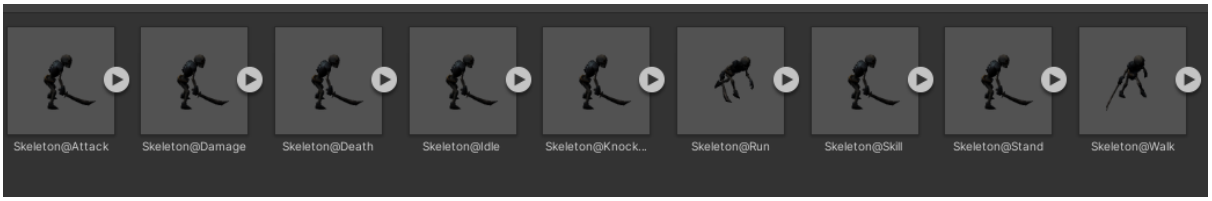
Skin jugador

La skin usada para el jugador fue descargada de la página tienda oficial de assets de Unity. Esta skin llamó nuestra atención ya que tenía diferentes modelos de animación de desplazamientos laterales y horizontales bastante pulidos y por esto la elegimos.



Skin enemigo

La skin usada para el jugador fue descargada de la página tienda oficial de assets de Unity. Esta skin al igual que la del jugador, llamó nuestra atención ya que tenía diferentes modelos de animación de acciones tales como atacar, deambular, desplazarse, etc.



Erinys

Erinys es el personaje principal, no puede atacar. Este personaje debe tomar todas las monedas en el mapa antes que termine el tiempo y su alma este maldita por siempre. Pero debe tener cuidado de los Skeleton que cuidan de su tesoro azteca muy bien.

Movimiento

Para controlar a Erinys, se creó el Script CharacterLogic y se asoció al objeto del personaje.

```
public float movementSpeed = 5.0f;
public float rotationSpeed = 200.0f;
```

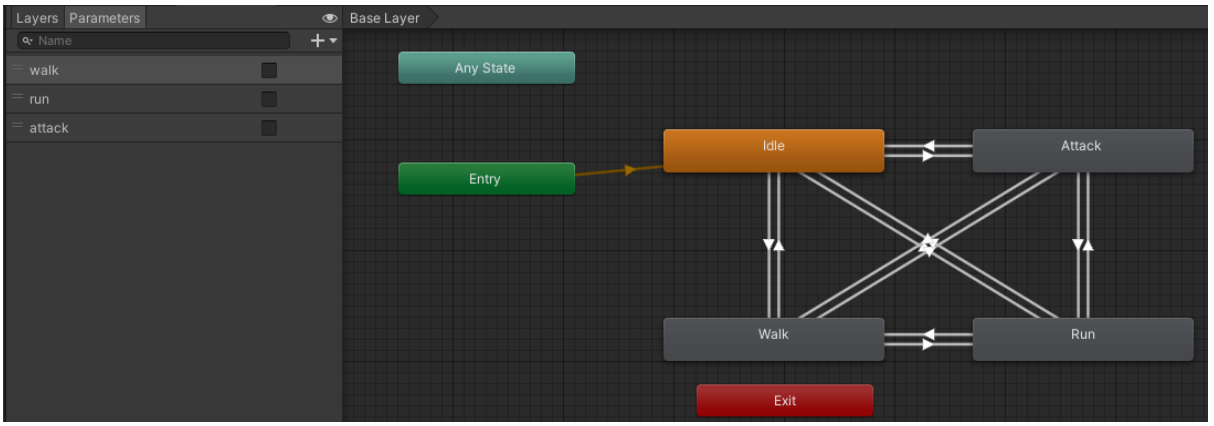
```
void Start() {
    animator = GetComponent<Animator>();
}


void Update() {
    x = Input.GetAxis("Horizontal");
    y = Input.GetAxis("Vertical");

    transform.Rotate(0, x * Time.deltaTime * rotationSpeed, 0);
    transform.Translate(0, 0, y * Time.deltaTime * movementSpeed);

    animator.SetFloat("VelX", x);
    animator.SetFloat("VelY", y);
}

void OnTriggerEnter(Collider collider) {
    if (collider.CompareTag("blade")) {
        print("Damage");
        Debug.Log("damage");
    }
}
```

Skeleton

Este es el enemigo del videojuego. Este enemigo es más grande y lento que el personaje principal y tiene un daño del 100%, es decir que de un solo golpe con su espada nos mata.

Movimiento

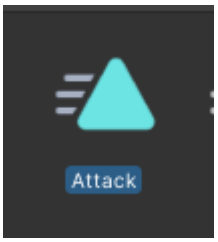
El Skeleton cuenta con varios estados, Idle, walk, run y attack. Y alterna entre ellos según su distancia con el jugador.



Como se muestra en este script, el Skeleton permanece caminando de manera aleatoria en el mapa hasta que el jugador entra en su rango y entonces corre tras él .

```
public void EnemyBehavior() {
    if (Vector3.Distance(transform.position, target.transform.position) > 10) {
        animator.SetBool("run", false);
        chronometer += 1 * Time.deltaTime;
        if (chronometer >= 4) {
            rutine = Random.Range(0, 2);
            chronometer = 0;
        }

        switch (rutine) {
            case 0:
                animator.SetBool("walk", false);
                break;
            case 1:
                degree = Random.Range(0, 360);
                angle = Quaternion.Euler(0, degree, 0);
                rutine++;
                break;
            case 2:
                transform.rotation = Quaternion.RotateTowards(transform.rotation, angle, 0.5f);
                transform.Translate(Vector3.forward * 1 * Time.deltaTime);
                animator.SetBool("walk", true);
                break;
        }
    } else {
        if (Vector3.Distance(transform.position, target.transform.position) >= 1  && !attacking) {
            var lookPosition = target.transform.position - transform.position;
            lookPosition.y = 0;
            var rotation = Quaternion.LookRotation(lookPosition);
            transform.rotation = Quaternion.RotateTowards(transform.rotation, rotation, 3);
            animator.SetBool("walk", false);
            animator.SetBool("run", true);
            transform.Translate(Vector3.forward * 2 * Time.deltaTime);

            animator.SetBool("attack", false);
        } else {
            animator.SetBool("walk", false);
            animator.SetBool("run", false);
            animator.SetBool("attack", true);
            attacking = false;
        }
    }
```

Cuando el jugador está a una distancia corta del Skeleton, este cambia a estado de ataque y ejecuta la animación de atacar.



Colisiones

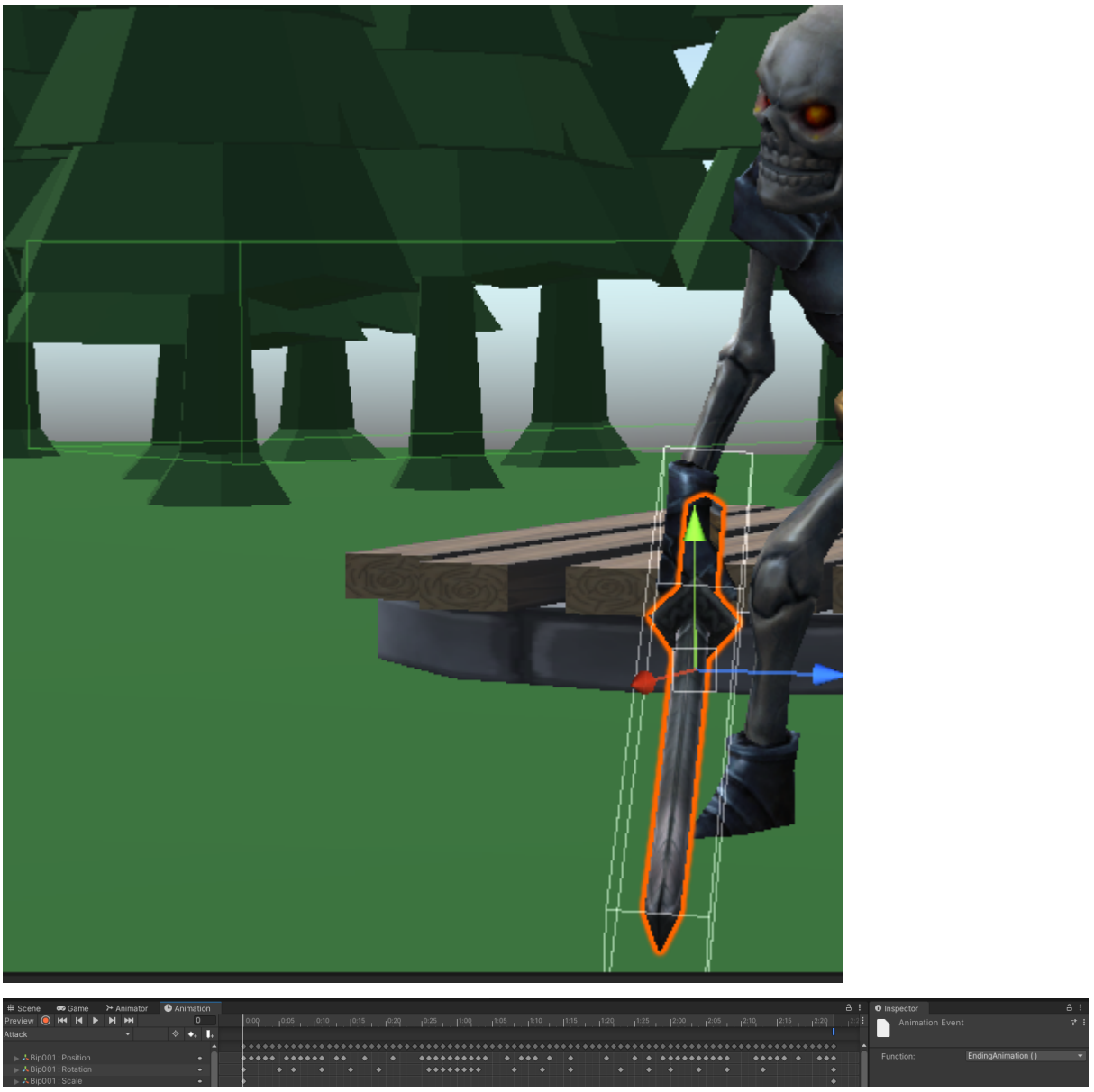
El Skeleton tiene un capsule collider y un rigidbody. Sin embargo, sólo es la espada la que colisiona con el jugador y la que nos muestra el mensaje de GameOver.

Esta colisión se ejecuta en la clase de Erinys.

```
void OnTriggerEnter(Collider collider) {
    if (collider.CompareTag("blade")) {
        gameOver.SetActive(true);
        Other();
    }
}
```

Espada

Como se mencionó anteriormente, la espada tiene su propio boxcollider que se activa cuando se ejecuta la animación de atacar.



Sistema puntaje

Monedas

Antes de implementar la lógica de ítems que al recolectarlos nos sumarán un puntaje, lo primero que se hizo fue buscar un asset que se adaptara a la estética del juego y una vez encontrado se empezó con el proceso de implementación.

Cuando importamos los modelos de la moneda desde la tienda de assets de Unity, cargamos uno de estos modelos en el mapa y le añadimos un box collider con la opción de “is trigger” activada.
Creamos 1 script llamado CollectCoin:

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

public class collectCoin : MonoBehaviour
{
    public AudioSource collectSound;

    void OnTriggerEnter(Collider other)
    {
        collectSound.Play();
        scoringSystem.theScore += 50;
        gameObject.SetActive(false);
    }

}
```

En este lo que se hace es ejecutar un sonido en el momento que el jugador colisione con la moneda y suma 50 puntos a un contador de puntaje global.

Score

Lo primero que se realizó para esta funcionalidad fue agregar un canvas a la cámara del jugador y en este se agregó a su vez un elemento UI de texto en el cual se va ver nuestro puntaje.

Una vez tenemos el campo de texto, creamos un script llamado scoring System. En este script hacemos referencia a él campo de texto para que cuando recolectemos la moneda, el script de collectCoin mande el puntaje recolectado al script de scoringSystem y este lo muestre en la pantalla.

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

public class scoringSystem : MonoBehaviour
{

    public GameObject scoreText;
    public static int theScore;

    void Update()
    {
        scoreText.GetComponent<Text>().text = "SCORE: " + theScore;

    }

}
```

Contrarreloj

Para que el juego tenga mayor dificultad se añade un temporizador en cual al ser agotado se termina el juego. De esta manera los puntajes serán basados en quién recolecta la mayor cantidad de monedas en 1 minuto con 25 segundos.

Lo primero que se hizo fue agregar 2 elementos al canvas de la cámara del jugador, el primer elemento tiene el símbolo del tiempo y el segundo es un campo de texto donde aparecerán el tiempo; se agregó un estilo de texto tipo digital para aportar mejor estética al conteo.

Se crea un script en él que se llevará el conteo del tiempo y se verá en la interfaz gráfica.

Miscellaneous Files — TimeController

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

public class TimeController : MonoBehaviour
{

    [SerializeField] int min, seg;
    //[SerializeField] Text tiempo;

    public GameObject tiempo;
    private float restante;
    private bool enMarcha;


    private void Awake()
    {
        restante = (min * 60) + seg;
        enMarcha = true;
    }

    // Update is called once per frame
    void Update()
    {
        if (enMarcha)
        {
            restante -= Time.deltaTime;
            if (restante < 1)
            {
                //pasar pantalla de game over
                enMarcha = false;
            }
            int tempMin = Mathf.FloorToInt(restante / 60);
            int tempSeg = Mathf.FloorToInt(restante % 60);
            tiempo.GetComponent<Text>().text = string.Format("{00:00}:{01:00}", tempMin, tempSeg);
        }
    }
}

```

HighScore

El sistema de puntaje más alto no funciona con puntos, sino con tiempo. Pues el juego es a contrarreloj. Entre menos tiempo gastemos recolectando todas las monedas más alto estaremos en el ranking.

Funcionamiento

La clase Hall Of Fame utiliza PlayerPrefs para guardar los tiempos del jugador, además de su nickname.

Para saber en qué posición debe guardar el tiempo, se realiza una comparación entre los tiempos del jugador y los de la tabla HighScores y se guarda. Las comparaciones entre Strings se hacen de la siguiente manera.

```
/*
 *    0 = both the numbers are equal
 *    1 = second number is smaller || first number is bigger
 *   -1 = first number is smaller  || second number is bigger
 *
 * El que tenga mayor tiempo restante es mas rapido
 */

//Comparador pa ver donde lo guarda
// I compare my time with the first
if (1 == currentTime.CompareTo(PlayerPrefs.GetString("1T"))) {
```

Guardar Nombre del jugador

Para el jugador, una vez realiza un tiempo que esté entre los 10 mejores, se lanza una Scene que le pide su nombre y que guarda su tiempo en la tabla HighScores.

```
void Update() {
    if (ScoringSystem.theScore == 800) {
        PlayerPrefs.SetString("currentTime", time.text);
        saved = true;
        // I compare my time with the Tenth
        if (1 == time.text.CompareTo(PlayerPrefs.GetString("10T"))) {
            //Im Bigger, So I can save my Nick
            SceneSwitcher.OpenScene(2);
        } else {
            //Im Lower, So I can't save my Nick
            SceneSwitcher.OpenScene(1);
        }
    }
```

Esta acción se realiza desde el GameManager, quien realiza la lógica de si debe pedir el nombre del jugador o si se regresa de inmediato al menú principal. Esto a través de SceneSwitcher.

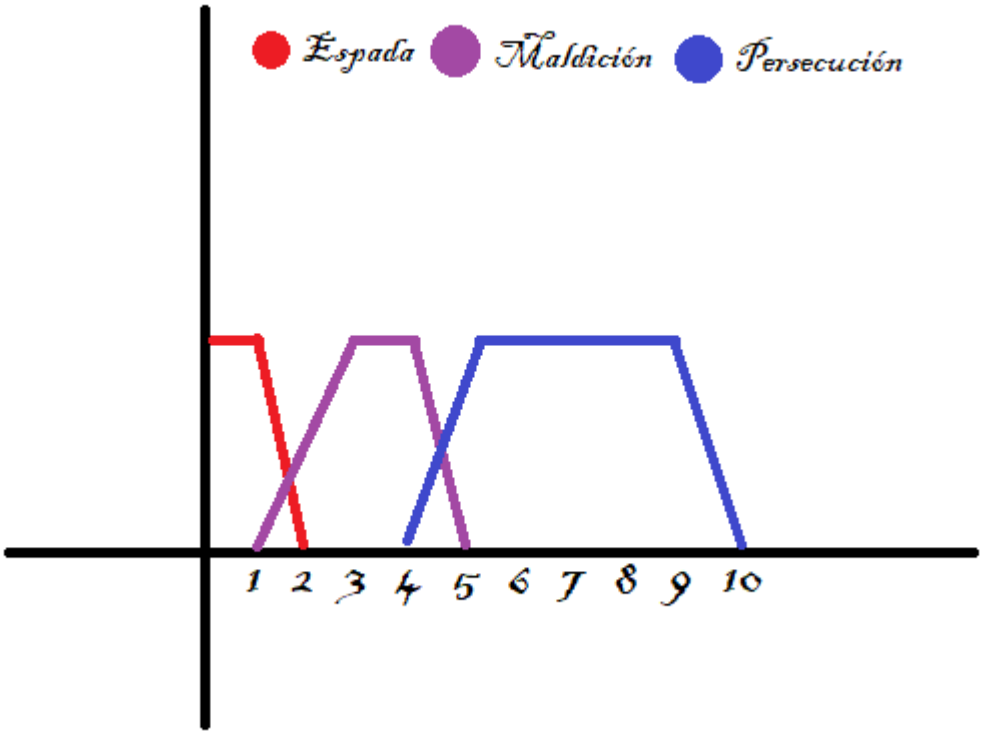
Audio del juego

El audio del juego está controlado por un GameObject llamado Sound que tiene un AudioSource son el audio del juego. Esta es una canción de otro videojuego en internet llamado Territorio War. El nombre de la canción es: The Graveyard Spook.

Lógica difusa

El planteamiento inicial de la lógica difusa sería atribuido a el enemigo del juego, ya que consideramos que este encaja en el caso ideal para el debido funcionamiento.
En la fuzzificación empleamos como datos de entrada la distancia entre el jugador y el enemigo, de esta manera ya tenemos las variables que van a ser usadas para encontrar el grado de membresía. Las funciones de membresía que establecimos fueron 3, la primera es de función inversa, la segunda y tercera son función de trapezoide. La primera función va representar que cuando el jugador esté en una distancia de 0 a 2 el enemigo utiliza su espada para hacer daño y abatir de forma instantánea al jugador. La segunda función va de 1 a 5 y en este rango el enemigo desplegará una “Zona de maldición” la cual hace que el jugador se desplaza más lento, esto a fin de hacerlo perder más tiempo e incluso permitirle tenerlo en el rango

de golpe. Siendo la tercera función la última, está lo que hace es acercarse al jugador cuando este en el rango de 4 a 10.



Cuando tenemos el conjunto definimos una regla difusa la cual es:
$REGLA\ 1\ IF\ persecución\ AND\ Maldición\ THEN\ Espada$
$REGLA\ 2\ IF\ NOT(Espada)\ AND\ Persecución\ THEN\ Maldición$
$REGLA3\ IF\ NOT(Espada\ AND\ maldición)\ THEN\ Persecución$

Una vez tenemos las reglas de membresía que nos dan los grados de membresía, procedemos a hacer una defuzzificación mediante el método singleton.

$$o = \frac{\sum_{i=1}^{n} MiXi}{\sum_{i=1}^{n} Mi}$$

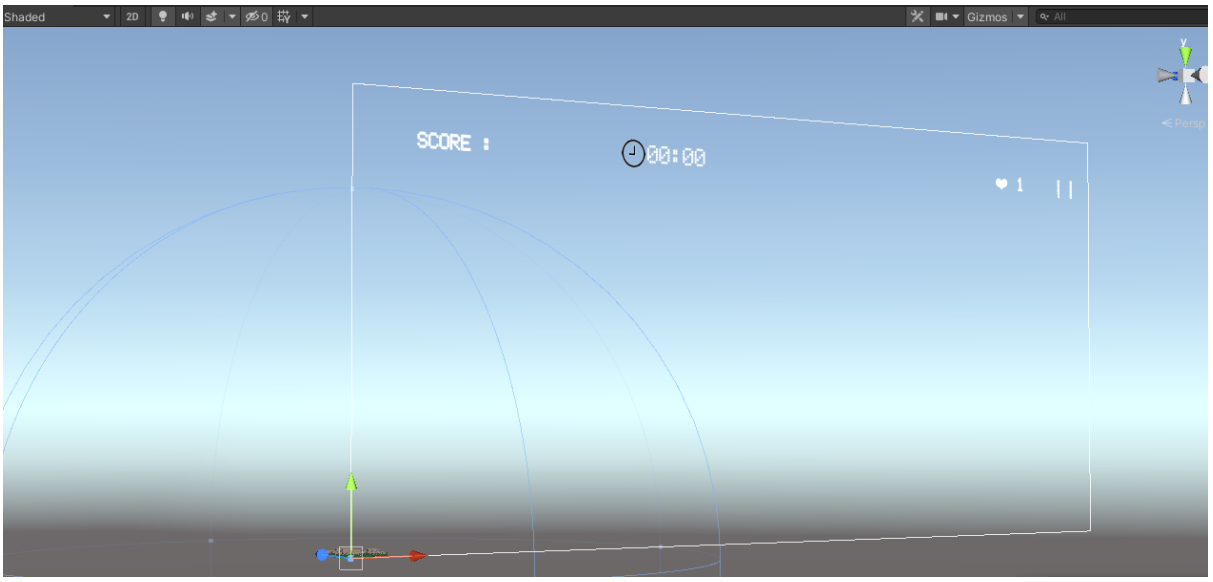
Donde **Mi** son los grados de membresía y **Xi** son los factores de membresía.
Con el resultado de esta defuzificación el enemigo es capaz de tomar decisiones de sus acciones con respecto a la distancia del jugador.

Problemas durante la implementación

Durante el desarrollo de este videojuego encontramos muchos problemas, empezando con la importación de assets, los cuales no importaban todos sus paquetes de manera correcta, por lo que se tomó la decisión de cambiar los assets por otros en la página de unity.

Por otro lado, tuvimos problemas con Erinys, puesto que al inicio no funcionaba cuando se incorporaron las animaciones de caminar, correr, etc. Razón por la cual, nos vimos forzados a crear un nuevo proyecto e implementar a Erinys desde cero. Solución que fue efectiva, puesto que ya no tuvimos más problemas con ella.

Finalmente, un error o mejor dicho un inconveniente que nos encontramos hasta la etapa final de desarrollo es que en la pestaña de edición, al crear un canvas para mostrar el botón de pausa, el score y el tiempo se nos bloqueó por completo la navegación 3D que teníamos con el mouse, por lo que desde ese momento tuvimos que navegar con las letras WASD pulsando el botón derecho del mouse.



Bibliografía

- https://www.youtube.com/watch?v=aIPMc92LsL0
- https://assetstore.unity.com/packages/3d/characters/animals/dog-knight-pbr-polyart-135227
- https://assetstore.unity.com/packages/3d/props/too-many-items-kitchen-props-127635
- https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/rpg-hero-pbr-hp-polyart-121480
- https://assetstore.unity.com/packages/3d/characters/animals/meshtint-free-chicken-mega-toon-series-151842
- https://assetstore.unity.com/packages/3d/characters/toony-tiny-people-demo-113188
- https://assetstore.unity.com/packages/3d/characters/humanoids/barbarian-warrior-75519#reviews
- https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy-monster-skeleton-35635
- https://assetstore.unity.com/packages/3d/props/bridges-pack-212950
- https://assetstore.unity.com/packages/3d/props/stylized-country-house-205553
- https://assetstore.unity.com/packages/3d/props/exterior/low-poly-boat-yard-128856
- https://assetstore.unity.com/packages/3d/environments/fantasy/medieval-town-exteriors-27026
- https://assetstore.unity.com/packages/3d/vegetation/nature-pack-low-poly-trees-bushes-210184
- https://assetstore.unity.com/packages/3d/props/exterior/crates-and-barrels-pack-volume-1-free-version-128829
- https://assetstore.unity.com/?category=3d&price=0-0&orderBy=1&page=17
- https://youtu.be/_5pxcUykXcA
- https://youtu.be/Bi4g2cnN0Go
- https://www.youtube.com/watch?v=_5pxcUykXcA&t=401s
- https://www.youtube.com/watch?v=Ay_oy6GXC-s