



UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

Proyecto Dungeon Gunner

Curso: DISEÑO Y CREACIÓN DE VIDEOJUEGOS

Docente: Patrick José Cuadros Quiroga

Integrantes:

Oswaldo Jesus Chino Conde (2019063322)
Abraham Jesús Vela Vargas (2019063322)

**Tacna – Perú
2024**

VIDEOJUEGOS UNIDAD 01

I. OBJETIVOS

- Elaborar un videojuego de plataforma 2D aplicando todos los conocimientos aprendidos en el curso de diseño y creación de videojuegos.

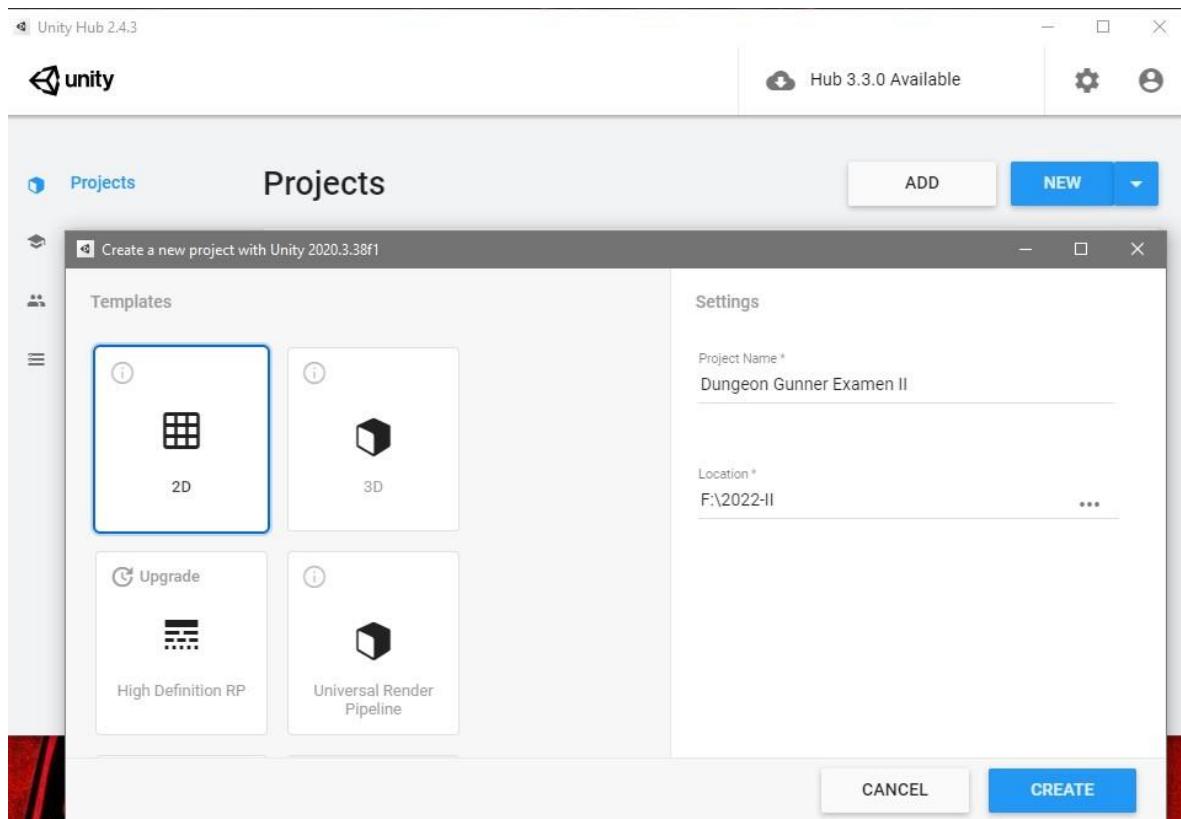
II. HERRAMIENTAS

- Unity Hub
- PixelArt

III. DESARROLLO

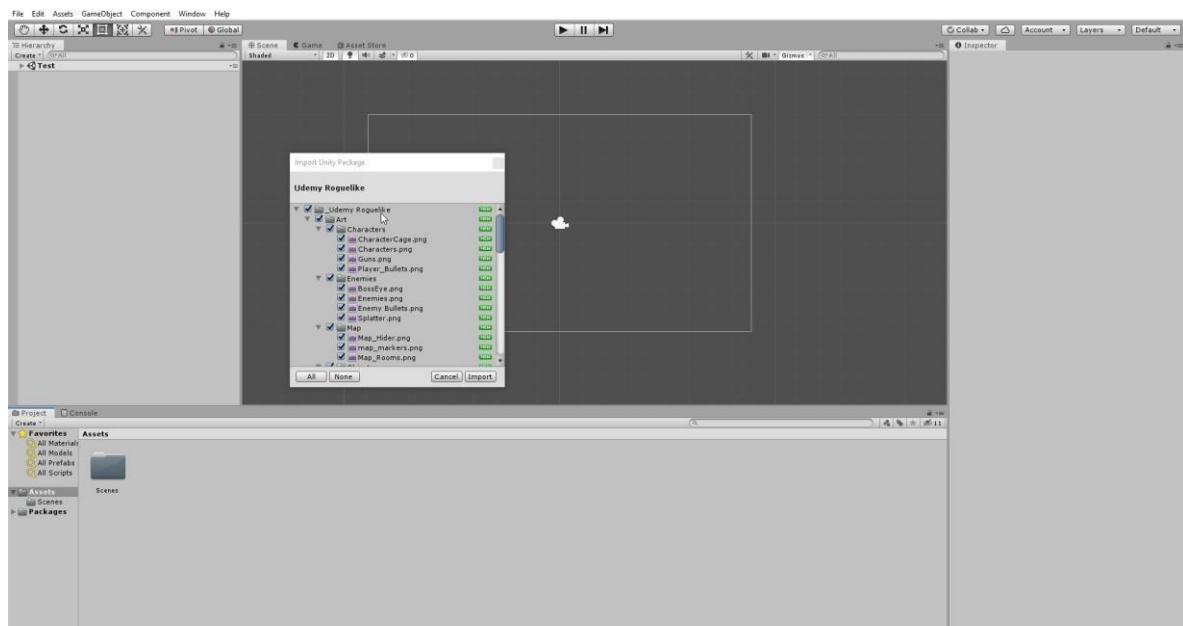
Sección 1: Creación del proyecto

Creamos nuestro proyecto en 2D.



Sección 2: Configuración de sprites y carpetas Importamos

nuestros sprites.

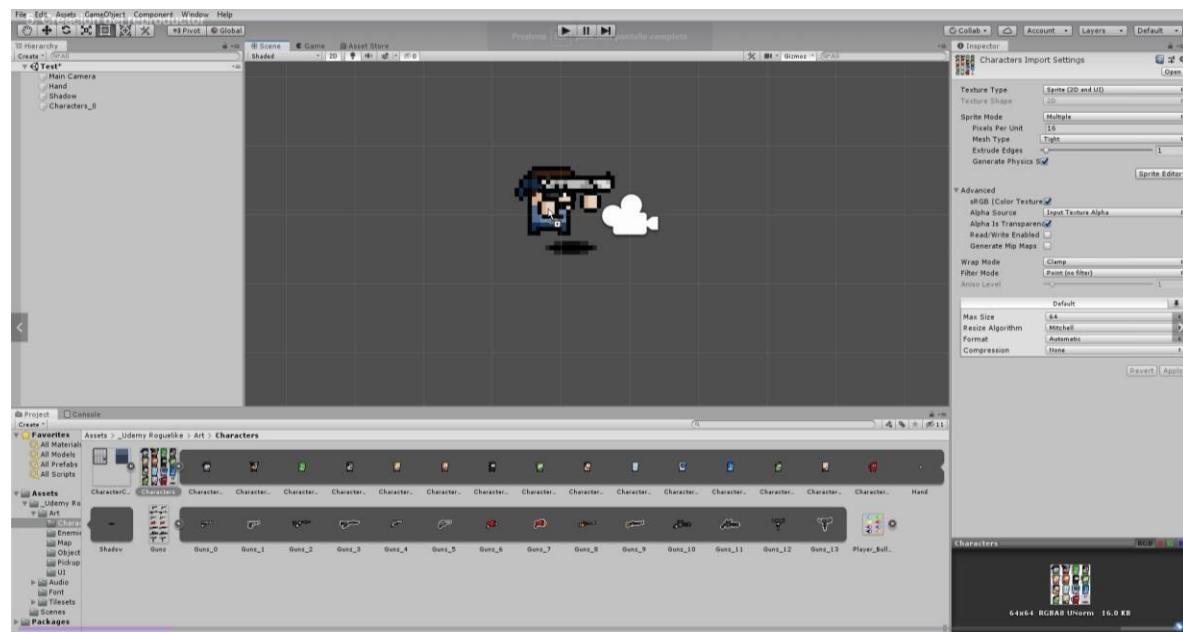


Luego recortamos cada personaje para que sea independiente y le asignamos un nombre y un tamaño de 16x16.



Sección 3: Control del jugador

Diseñamos el modelo de nuestro player.



En el script de Player hacemos la configuración para que nuestro personaje pueda moverse en todos los mapas.

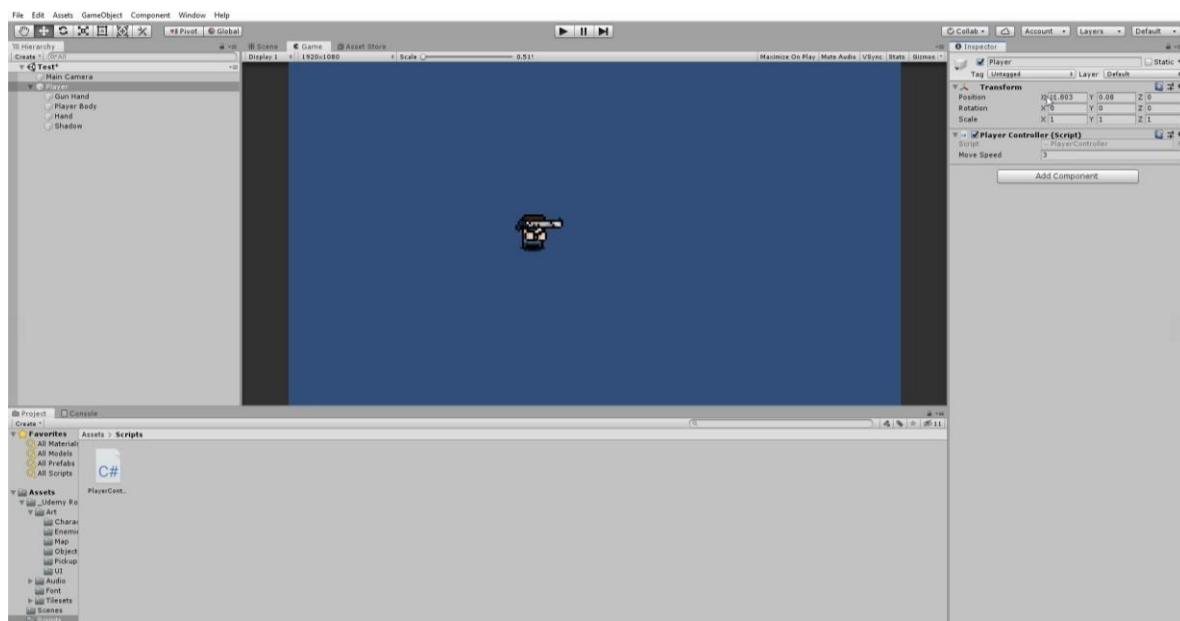
```
public class PlayerController : MonoBehaviour
{
    public float moveSpeed;
    private Vector2 moveInput;

    // Start is called before the first frame update
    void Start()
    {
    }

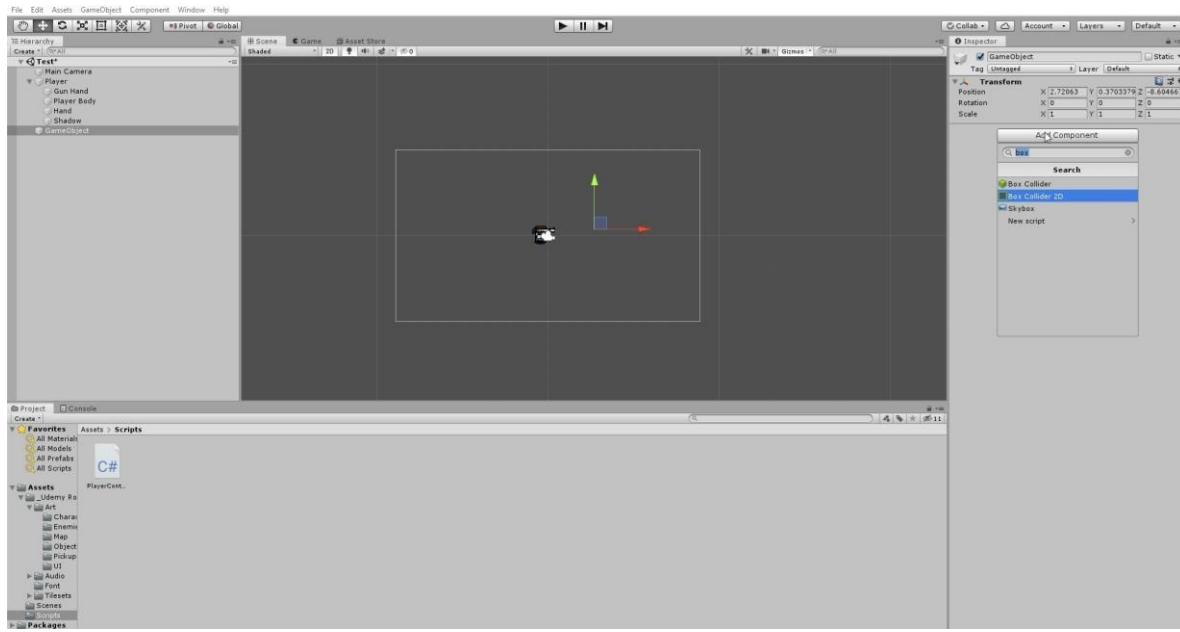
    // Update is called once per frame
    void Update()
    {
        moveInput.x = Input.GetAxisRaw("Horizontal");
        moveInput.y = Input.GetAxisRaw("Vertical");

        transform.position += new Vector3(moveInput.x * Time.deltaTime , moveInput.y * Time.deltaTime, 0f);
    }
}
```

Arrastramos el script hacia nuestro objeto.



Agregamos una colisión para controlar el espacio del mapa con el Player.



Ahora para que el arma tenga movimiento creamos una función en el cual capturemos en ángulo donde se encuentra nuestro mouse.

```
// Update is called once per frame
void Update()
{
    moveInput.x = Input.GetAxisRaw("Horizontal");
    moveInput.y = Input.GetAxisRaw("Vertical");

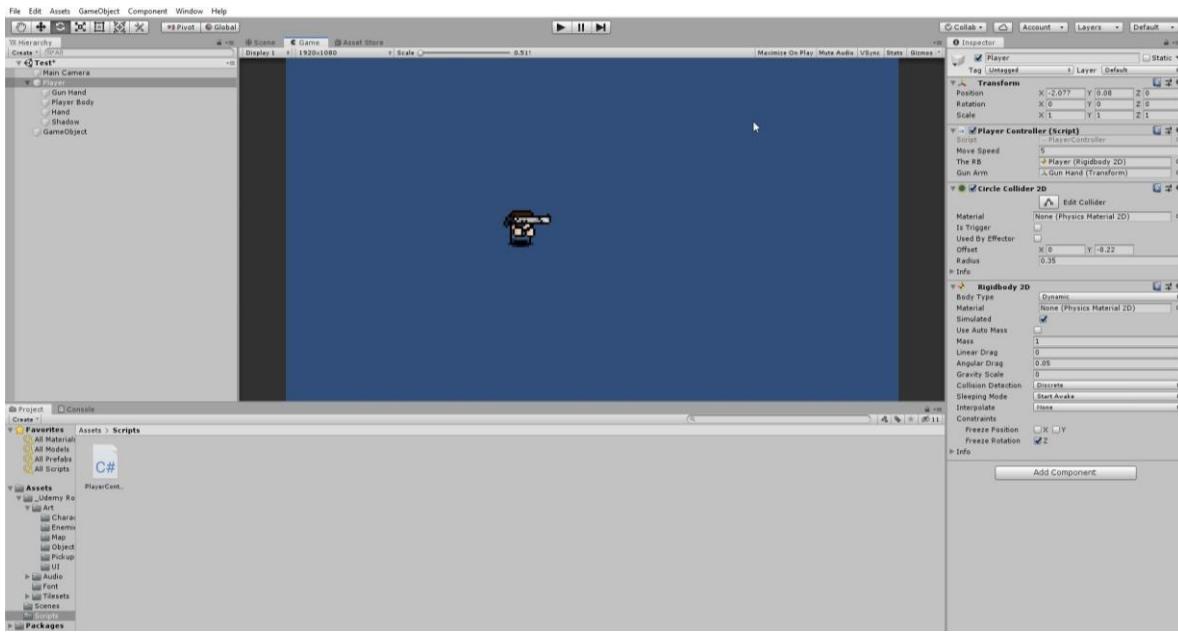
    //transform.position += new Vector3(moveInput.x * Time.deltaTime * moveSpeed, moveInput.y * Time.deltaTime * moveSpeed, 0f);

    theRB.velocity = moveInput * moveSpeed;

    Vector3 mousePos = Input.mousePosition;
    Vector3 screenPoint = Camera.main.WorldToScreenPoint(transform.localPosition);

    //rotate gun arm
    Vector2 offset = new Vector2(mousePos.x - screenPoint.x, mousePos.y - screenPoint.y);
    float angle = Mathf.Atan2(offset.y, offset.x) * Mathf.Rad2Deg;
    gunArm.rotation = Quaternion.Euler(0, 0, angle);
}
```

Instanciamos nuestros objetos creados.



Para que ahora el Player mire en la posición donde esta el arma creamos la siguiente condicional.

```
// Update is called once per frame
void Update()
{
    moveInput.x = Input.GetAxisRaw("Horizontal");
    moveInput.y = Input.GetAxisRaw("Vertical");

    //transform.position += new Vector3(moveInput.x * Time.deltaTime * moveSpeed, moveInput.y * Time.deltaTime * moveSpeed, 0f);

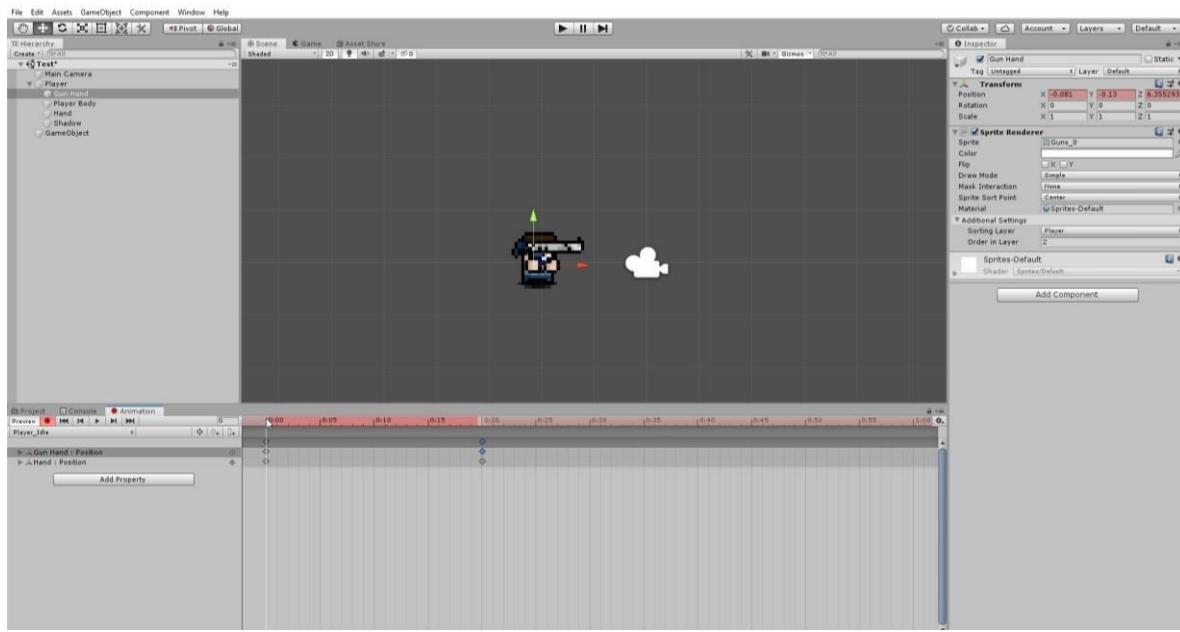
    theRB.velocity = moveInput * moveSpeed;

    Vector3 mousePos = Input.mousePosition;
    Vector3 screenPoint = Camera.main.WorldToScreenPoint(transform.localPosition);

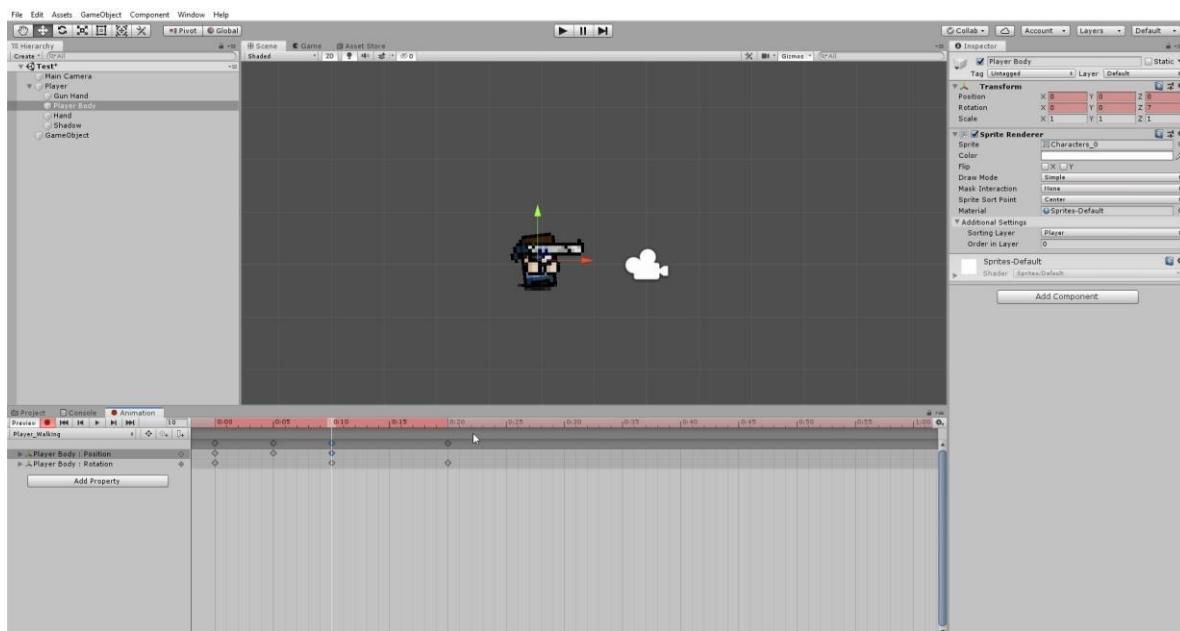
    if(mousePos.x < screenPoint.x)
    {
        transform.localScale = new Vector3(-1f, 1f, 1f);
        gunArm.localScale = new Vector3(-1f, -1f, 1f);
    }
    else
    {
        transform.localScale = Vector3.one;
        gunArm.localScale = Vector3.one;
    }

    //rotate gun arm
    Vector2 offset = new Vector2(mousePos.x - screenPoint.x, mousePos.y - screenPoint.y);
    float angle = Mathf.Atan2(offset.y, offset.x) * Mathf.Rad2Deg;
    gunArm.rotation = Quaternion.Euler(0, 0, angle);
}
```

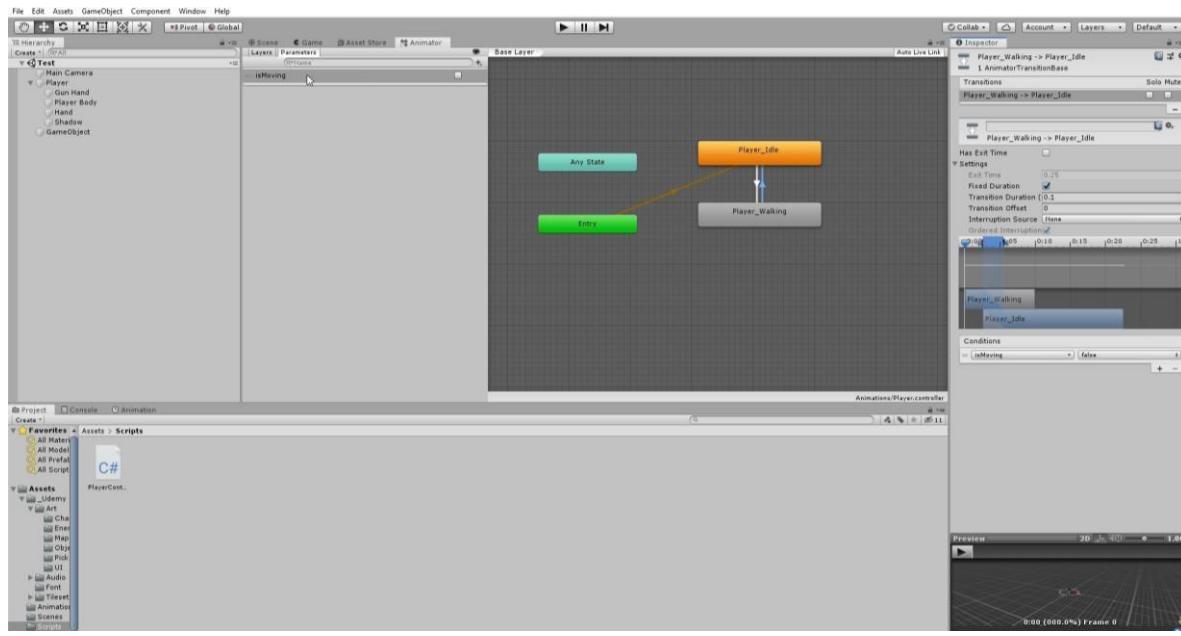
Creamos la primera animación para Player, esta será para que el arma se mueva.



Creamos una segunda animación para cuando el Player este en movimiento.



Creamos la jerarquía de animaciones.



Y una condicional para cuando suceda esto.

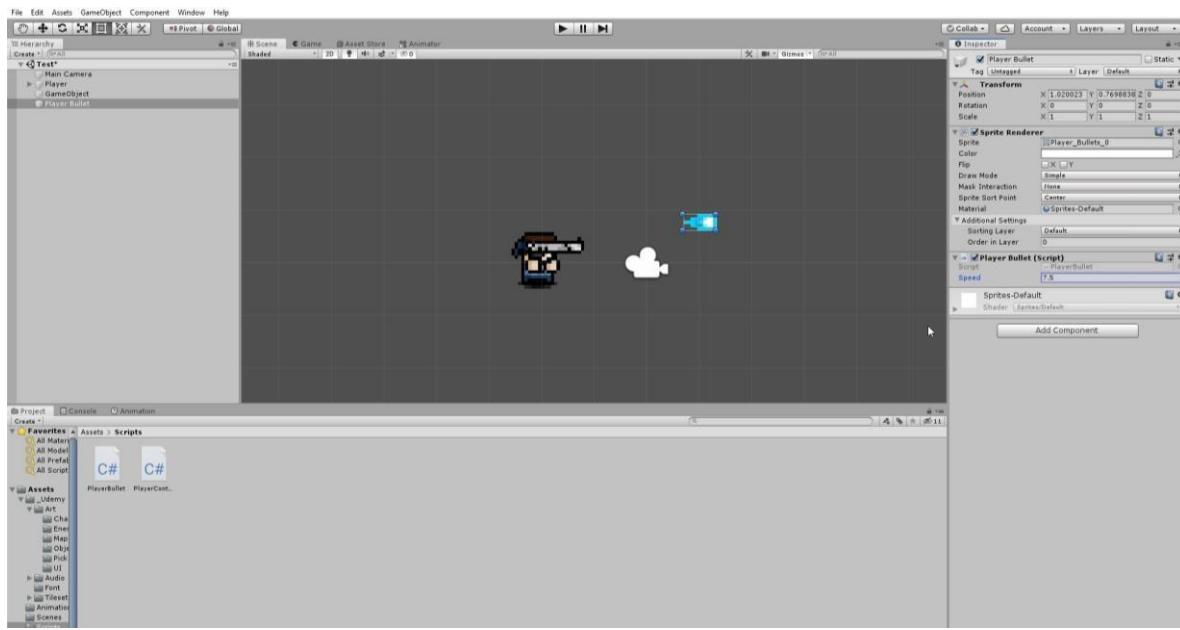
```

52
53     if(moveInput != Vector2.zero)
54     {
55         anim.SetBool("isMoving", true);
56     }
57     else
58     {
59         anim.SetBool("isMoving", false);
60     }
61 }
62

```

Sección 4: Disparo

Ahora creamos otro objeto para el disparo, con su respectivo script y lo instanciamos.



Creamos el script para la velocidad y dirección de la bala, y que cuando colisione con otro objeto desaparezca.

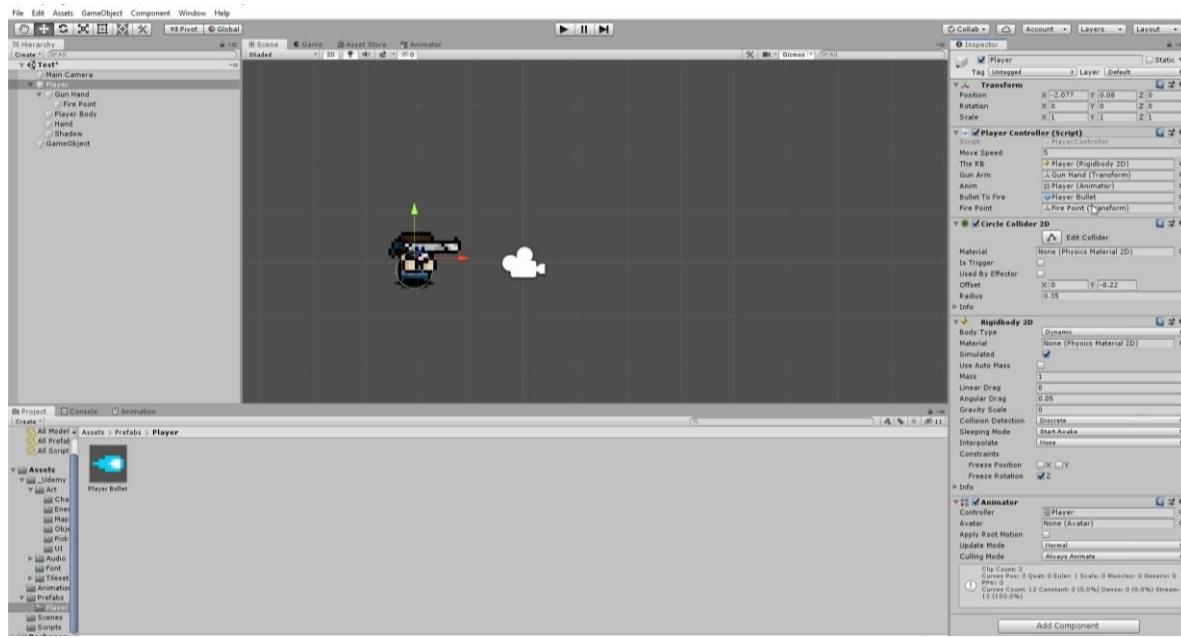
```
public class PlayerBullet : MonoBehaviour
{
    public float speed = 7.5f;
    public Rigidbody2D theRB;

    // Start is called before the first frame update
    void Start()
    {
    }

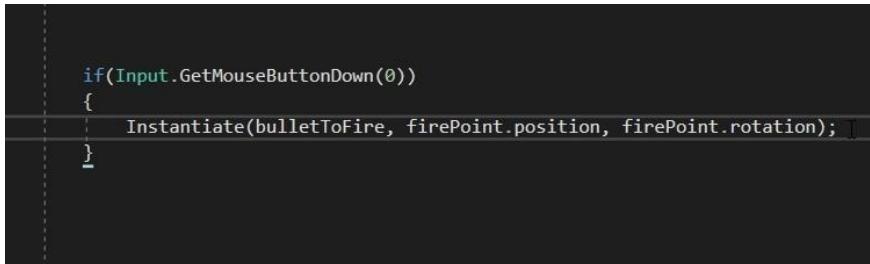
    // Update is called once per frame
    void Update()
    {
        theRB.velocity = transform.right * speed;
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        Destroy(gameObject);
    }
}
```

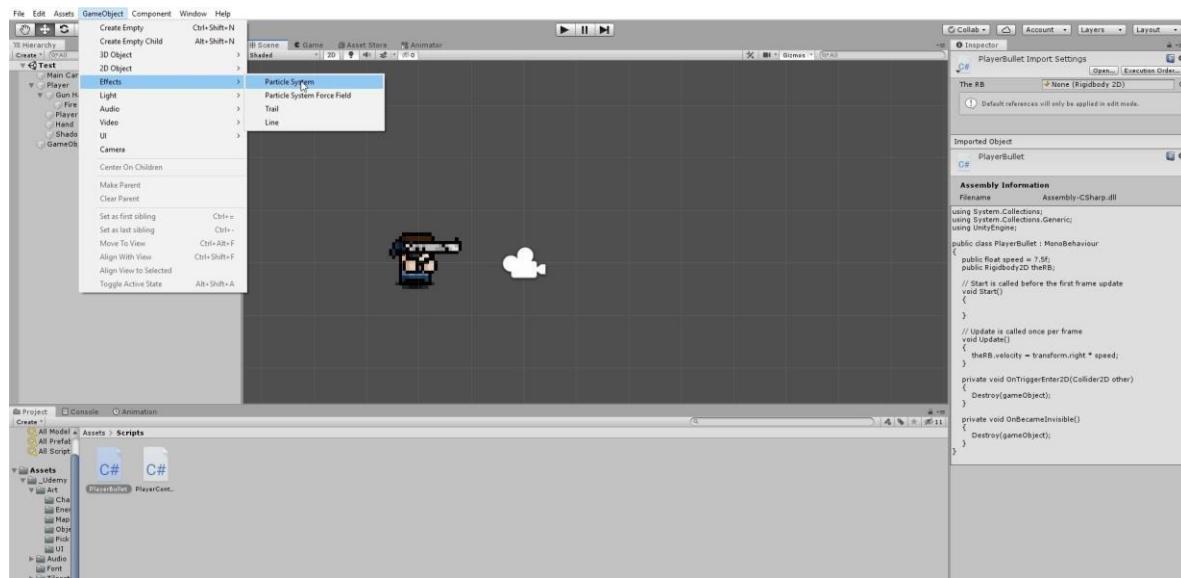
Instanciamos nuestro nuevos objetos creados en los campos del script.



Creamos la condicional para saber en que dirección esta apuntando el arma y salga la bala.



Creamos un Particle System para los efectos en partículas.



Las partículas botarán el efecto cuando colisione con otro objeto.

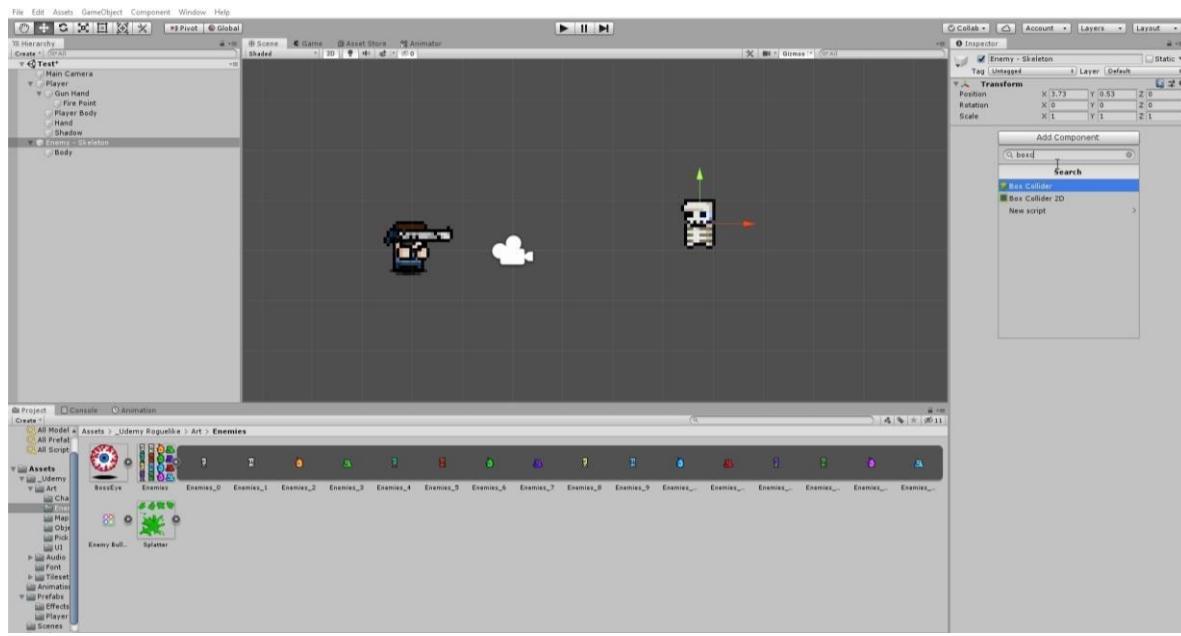
```
// Update is called once per frame
void Update()
{
    theRB.velocity = transform.right * speed;
}

private void OnTriggerEnter2D(Collider2D other)
{
    Instantiate(impactEffect, transform.position, transform.rotation);
    Destroy(gameObject);
}

private void OnBecameInvisible()
{
    Destroy(gameObject);
}
```

Sección 5: Creación enemigo

Creamos la estructura de nuestro enemy.



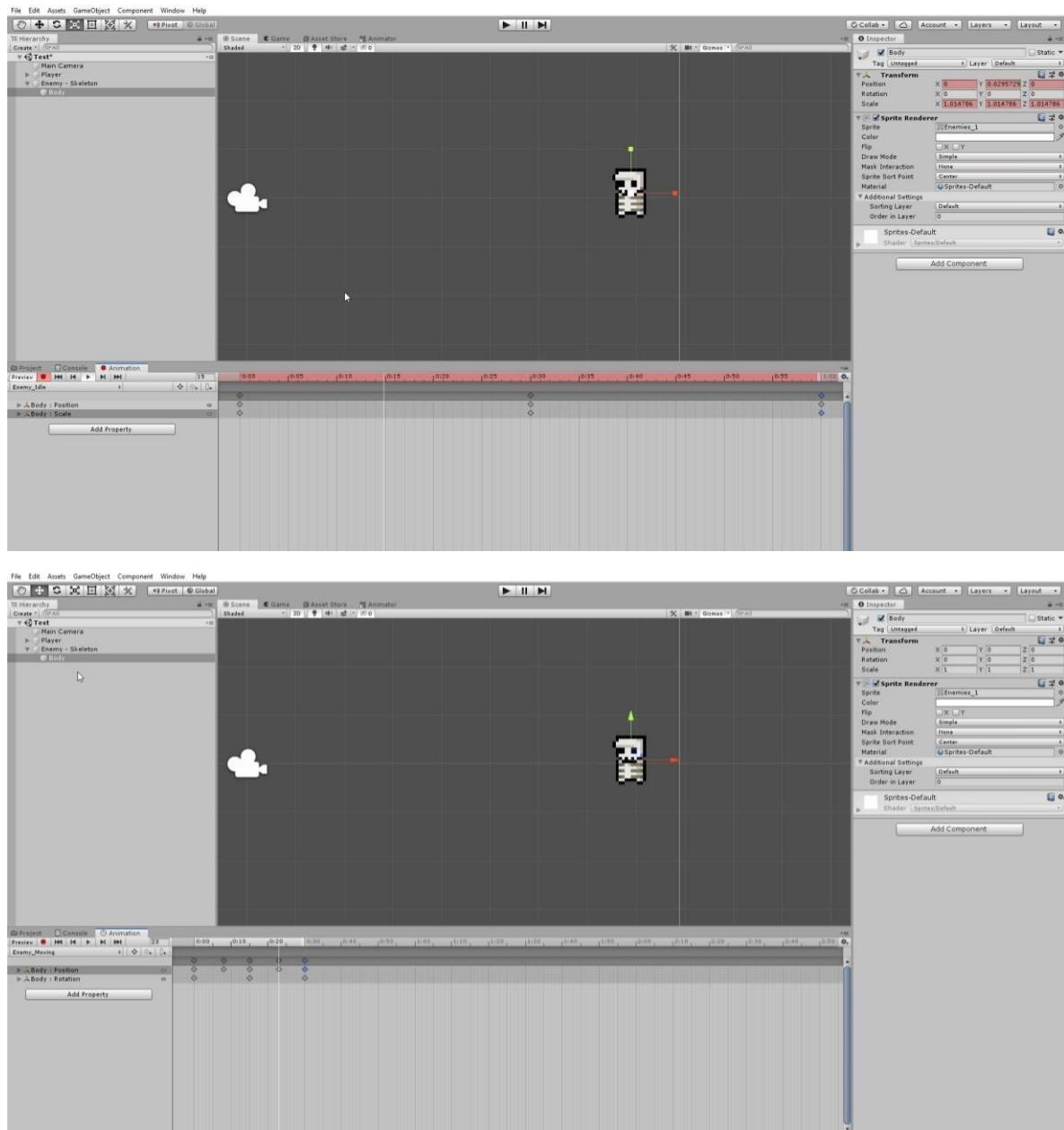
El script del enemigo para que se mueva hacia la posición del Player, solo si está dentro de un rango de distancia.

```
// Update is called once per frame
void Update()
{
    if(Vector3.Distance(transform.position, PlayerController.instance.transform.position) < rangeToChasePlayer)
    {
        moveDirection = PlayerController.instance.transform.position - transform.position;
    }
    else
    {
        moveDirection = Vector3.zero;
    }

    moveDirection.Normalize();

    theRB.velocity = moveDirection * moveSpeed;
}
```

Creamos la animación para el enemigo uno para cuando esté parado y otro en movimiento.



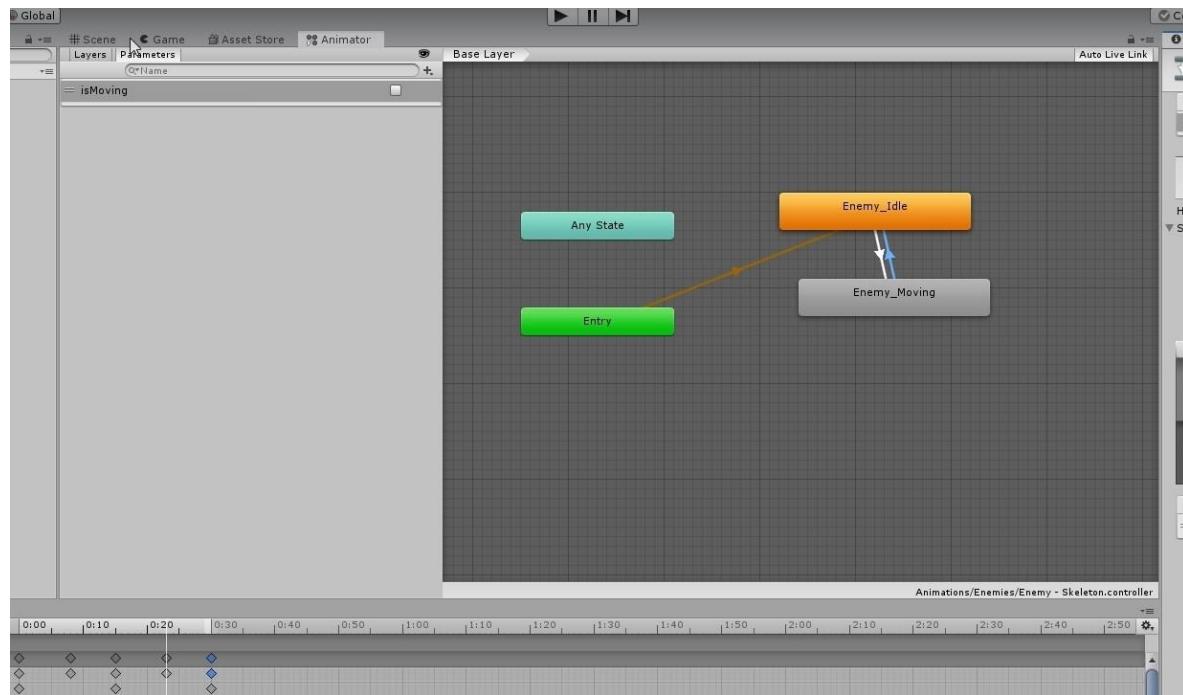
Agregamos en el script para que se active la animación si está en movimiento o no.

```

if (moveDirection != Vector3.zero)
{
    anim.SetBool("isMoving", true);
}
else
{
    anim.SetBool("isMoving", false);
}
}
}

```

Creamos jerarquías de animación.



Le asignamos un valor de vida a nuestro enemigo.

```
public float rangeToChasePlayer;
private Vector3 moveDirection;

public Animator anim;

public int health = 150;

// Start is called before the first frame update
void Start()
{
```

Y ponemos la condición de que si su vida es menor a 0 que se destruya el objeto.

```
public void DamageEnemy(int damage)
{
    health -= damage;

    if(health <= 0)
    {
        Destroy(gameObject);
    }
}
```

Aquí configuramos el daño que hará nuestra bala e instanciamos si la bala colisiona con un objeto que tenga la etiqueta enemy.

```
public class PlayerBullet : MonoBehaviour
{
    public float speed = 7.5f;
    public Rigidbody2D theRB;

    public GameObject impactEffect;

    public int damageToGive = 50;

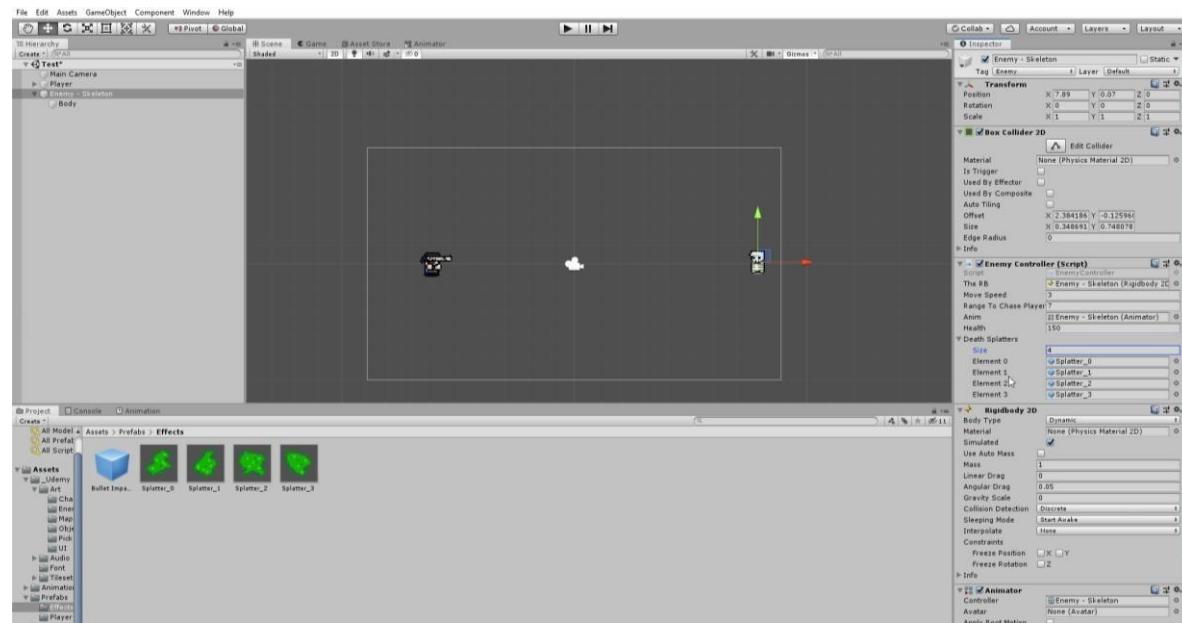
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        theRB.velocity = transform.right * speed;
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        Instantiate(impactEffect, transform.position, transform.rotation);
        Destroy(gameObject);

        if (other.tag == "Enemy")
        {
            other.GetComponent<EnemyController>().DamageEnemy(damageToGive);
        }
    }
}
```

Creamos charcos de sangre que emitirán los enemigos al ser asesinados.



Asignamos el script para que los charcos sean aleatorios.

```
//condicional para ver si la salud del enemigo
if(health <= 0)
{
    Destroy(gameObject);

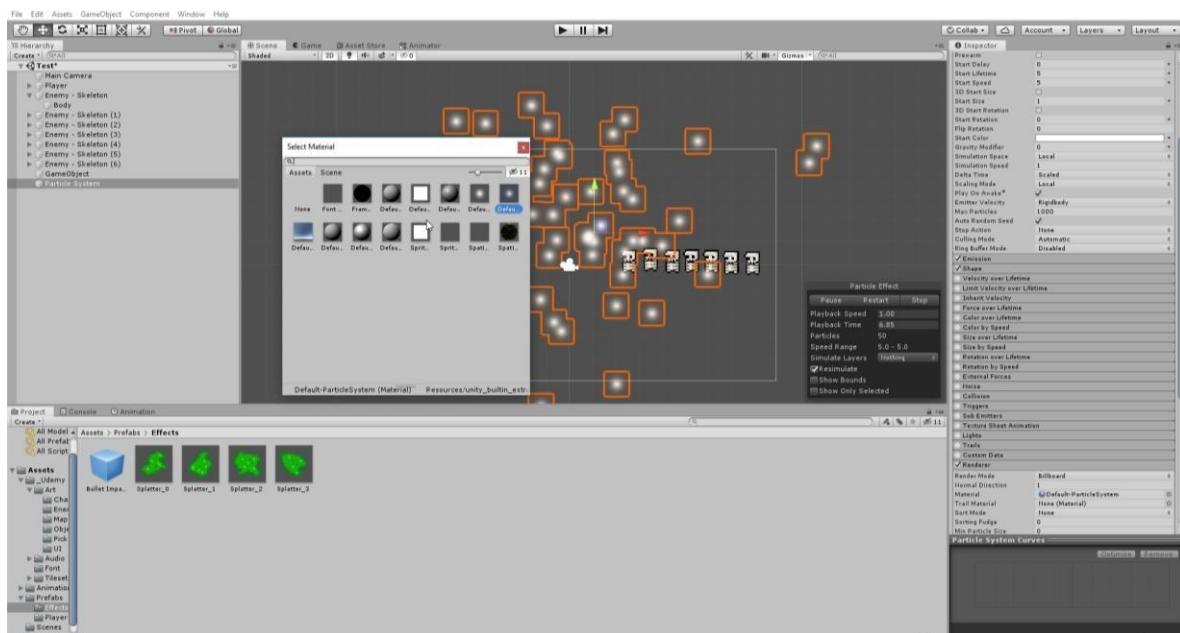
    AudioManager.instance.PlaySFX(1);

    //rango aleatorio para efecto de muerte
    int selectedSplatter = Random.Range(0, deathSplatters.Length);

    //rotacion aleatoria para efecto de muerte
    int rotation = Random.Range(0, 4);

    //instanciar que imagen de salpicadura muerte saldra una vez hallada el random y en que posicion
    Instantiate(deathSplatters[selectedSplatter], transform.position, Quaternion.Euler(0f, 0f, rotation * 90f));
}
```

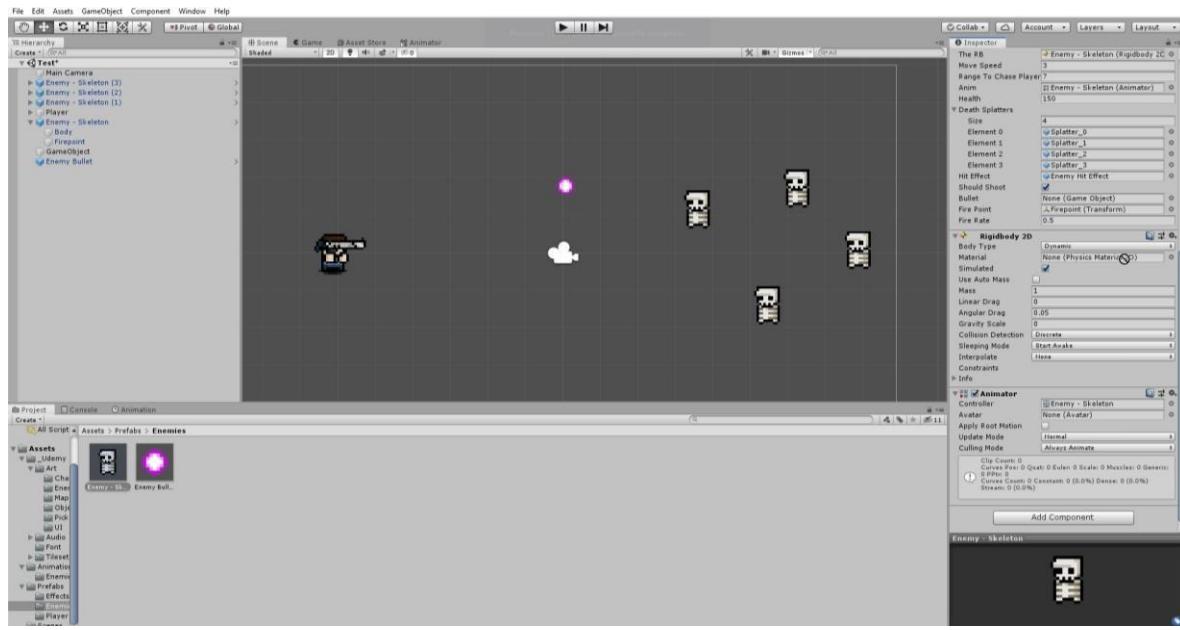
Ahora crearemos partículas de sangre que emitirán los enemigos en cada colisión con la bala.



Script para el sangrado del enemigo por bala.

```
//salpicadura de sangre cuando enemigo sea colisionado con la bala  
Instantiate(hitEffect, transform.position, transform.rotation);
```

Ahora creamos un prefab para las balas del enemigo.



Script para controlar el disparo, la distancia y el intervalo.

```

[Header("Shooting")]
//para que el enemigo pueda disparar y controlar el intervalo y las balas que dispara
public bool shouldShoot;
public GameObject bullet;
public Transform firePoint;
public float fireRate;
private float fireCounter;

public float speed; //velocidad en la que se dirige la bala
private Vector3 direction; //direccion en la que se movera

// Controlar el disparo y sea dirigido hacia el player
@ Mensaje de Unity | 0 referencias
void Start()
{
    direction = PlayerController.instance.transform.position - transform.position;
    direction.Normalize();
}

// Update is called once per frame
@ Mensaje de Unity | 0 referencias
void Update()
{
    transform.position += direction * speed * Time.deltaTime;
}

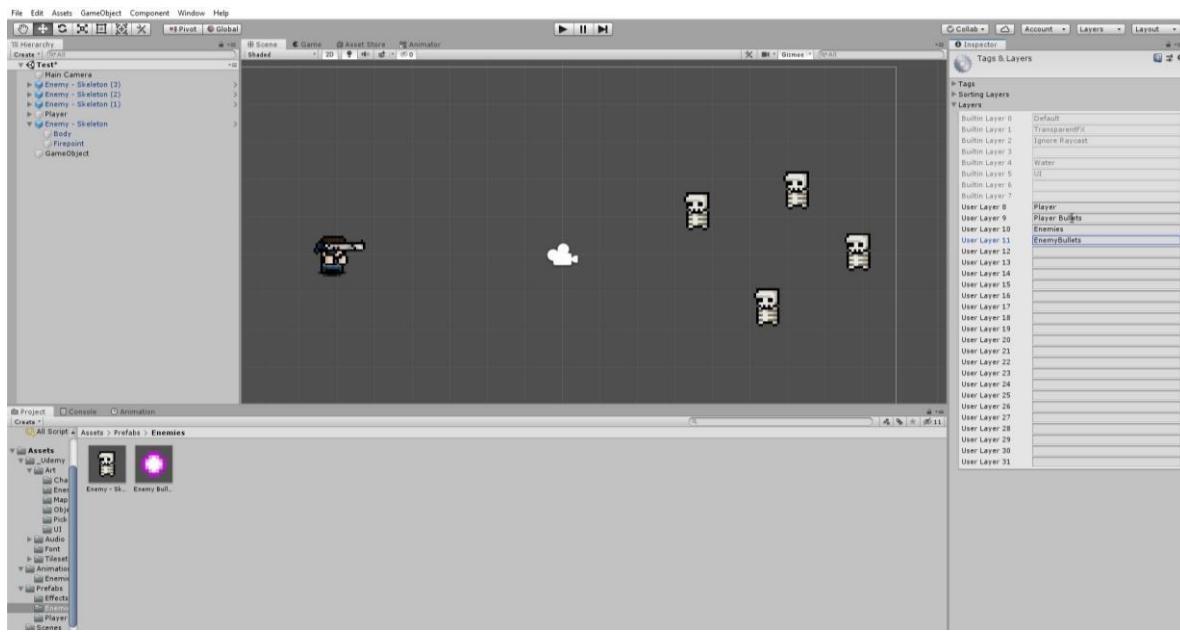
@ Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Player")
    {
        PlayerHealthController.instance.DamagePlayer();
    }

    Destroy(gameObject);
    AudioManager.instance.PlaySFX(4);
}

//cuando colisione o salga de la pantalla se destruya por completo
@ Mensaje de Unity | 0 referencias
private void OnBecameInvisible()
{
    Destroy(gameObject);
}

```

Ahora crearemos capas, para diferenciar de las balas de player con las de enemigo.



Controlar que el enemigo se encuentre visible en la pantalla del player para poder disparar.

```
//controlar si el enemigo se encuentra visible en la pantalla recien podra disparar
if(theBody.isVisible && PlayerController.instance.gameObject.activeInHierarchy)
{
    moveDirection = Vector3.zero;

    //condicional para que el enemigo se diriga hacia el player si este esta a un rango.
    if (Vector3.Distance(transform.position, PlayerController.instance.transform.position) < rangeToChasePlayer && shouldChasePlayer)
    {
        moveDirection = PlayerController.instance.transform.position - transform.position;
    } else
    {
        if(shouldWander)
        {
            if(wanderCounter > 0)
            {
                wanderCounter -= Time.deltaTime;

                //move the enemy
                moveDirection = wanderDirection;

                if(wanderCounter <= 0)
                {
                    pauseCounter = Random.Range(pauseLength * .75f, pauseLength * 1.25f);
                }
            }
        }
    }
}
```

Solo disparará si se encuentra dentro de una distancia del player.

```
//Solo dispara si se encuentra dentro de un rango de distancia con el player
if (shouldShoot && Vector3.Distance(transform.position, PlayerController.instance.transform.position) < shootRange)
{
    fireCounter -= Time.deltaTime;

    if (fireCounter <= 0)
    {
        fireCounter = fireRate;
        Instantiate(bullet, firePoint.position, firePoint.rotation);
        AudioManager.instance.PlaySFX(13);
    }
}
```

Sección 6: Salud y Daños

Controlar la salud del enemigo y cuando se impactado con una bala actualizar la salud.

```

PlayerController.instance.bodySR.color = new Color(PlayerController.instance.bodySR.color.r,
    PlayerController.instance.bodySR.color.g, PlayerController.instance.bodySR.color.b, .5f);

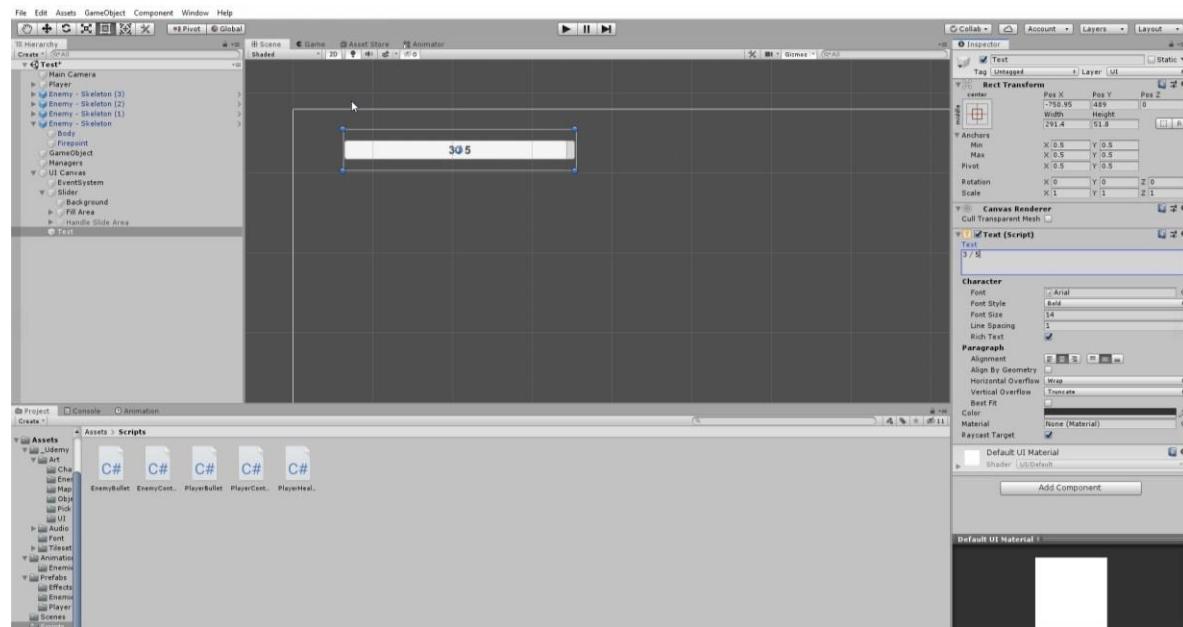
if (currentHealth <= 0)
{
    PlayerController.instance.gameObject.SetActive(false);
}

    UIController.instance.deathScreen.SetActive(true);

    AudioManager.instance.PlayGameOver();
    AudioManager.instance.PlaySFX(8);
}

```

Creamos un objeto para controlar la vida de nuestro jugador que va actualizando según los sucesos.



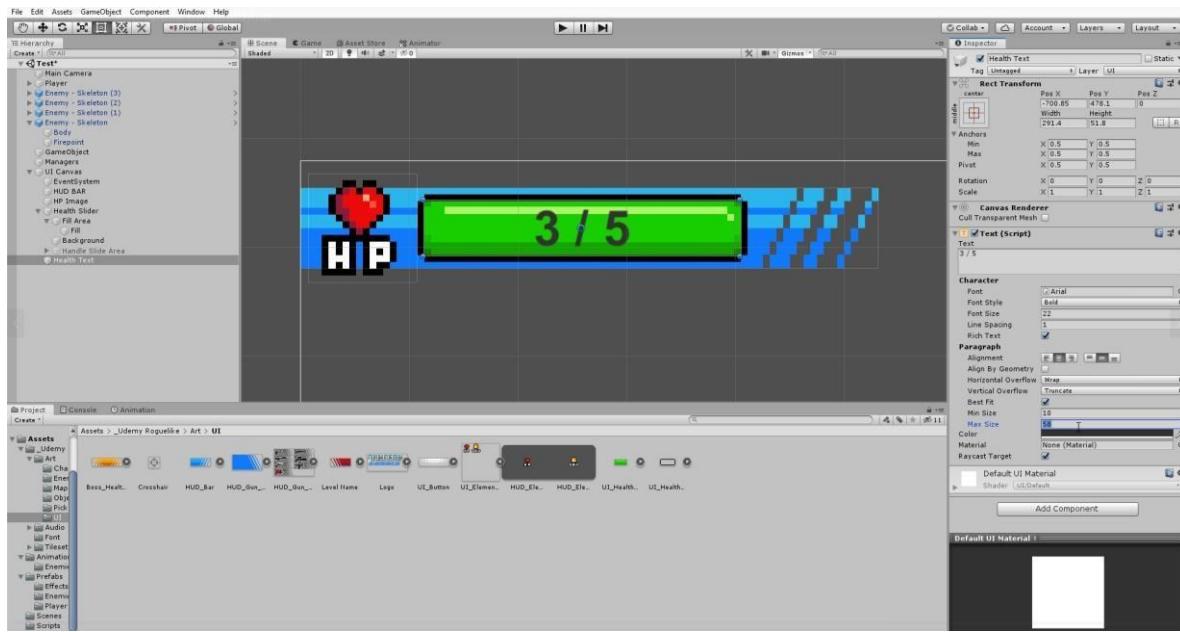
Script para que se actualice como texto.

```

//actualizar la barra de vida con una cadena de texto
UIController.instance.healthSlider.value = currentHealth;
UIController.instance.healthText.text = currentHealth.ToString() + " / " + maxHealth.ToString();

```

Diseñamos nuestra barra de vida con los Sprite prediseñados.



Creamos un script para cuando player muera nos regrese al menú principal

```
if (currentHealth <= 0)
{
    PlayerController.instance.gameObject.SetActive(false);

    //Cada vez que player muera nos manda al menu principal
    UIController.instance.deathScreen.SetActive(true);

    AudioManager.instance.PlayGameOver();
    AudioManager.instance.PlaySFX(8);
}
```



Agregamos un script para que cuando una bala afecte a player sea inmune por un segundo

```

void Update()
{
    if(invincCount > 0)
    {
        invincCount -= Time.deltaTime;

        //despues de 1 se vuelve vulnerable
        if(invincCount <= 0)
        {
            PlayerController.instance.bodySR.color = new Color(PlayerController.instance.bodySR.color.r, PlayerController.instance.bodySR.color.g, PlayerController.instance.bodySR.color.b, .5f);
        }
    }
}

6 referencias
public void DamagePlayer()
{
    //cuando reciba daño se vuelve invulnerable por 1 seg
    if (invincCount <= 0)
    {
        AudioManager.instance.PlaySFX(11);
        currentHealth--;

        invincCount = damageInvincLength;

        PlayerController.instance.bodySR.color = new Color(PlayerController.instance.bodySR.color.r, PlayerController.instance.bodySR.color.g, PlayerController.instance.bodySR.color.b, .5f);

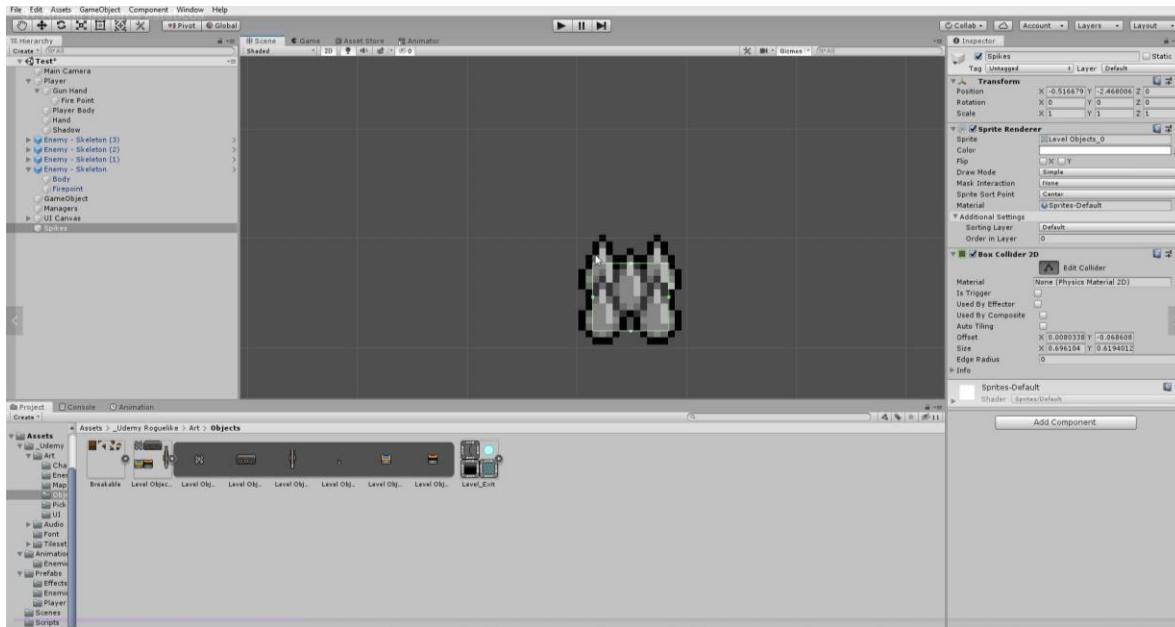
        if (currentHealth <= 0)
        {
            PlayerController.instance.gameObject.SetActive(false);

            //Cada vez que player muera nos manda al menu principal
            UIManager.instance.deathScreen.SetActive(true);

            AudioManager.instance.PlayGameOver();
            AudioManager.instance.PlaySFX(8);
        }
    }
}

```

Creamos otro objeto estático que hará daño



Agregamos el script para que dicho objeto haga daño cuando tenga contacto con player.

```

//cada vez que player tenga contacto con un objeto de peligro se hara daño
@ Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Player")
    {
        PlayerHealthController.instance.DamagePlayer();
    }
}

@ Mensaje de Unity | 0 referencias
private void OnTriggerStay2D(Collider2D other)
{
    if (other.tag == "Player")
    {
        PlayerHealthController.instance.DamagePlayer();
    }
}

//Si permanece dentro del objeto tambien seguira haciendose daño
@ Mensaje de Unity | 0 referencias
private void OnCollisionEnter2D(Collision2D other)
{
    if (other.gameObject.tag == "Player")
    {
        PlayerHealthController.instance.DamagePlayer();
    }
}

@ Mensaje de Unity | 0 referencias
private void OnCollisionStay2D(Collision2D other)
{
    if (other.gameObject.tag == "Player")
    {
        PlayerHealthController.instance.DamagePlayer();
    }
}

```

Sección 7: Correr y Aplastar

Creamos un nuevo poder que es dash esto funcionara con el espacio. Solo se podrá usar dash cuando estemos bajo el efecto de inmunidad.

```

//asignamos un nuevo poder de dash cuando presionemos espacio
if (Input.GetKeyDown(KeyCode.Space))
{
    if (dashCoolCounter <= 0 && dashCounter <= 0)
    {
        activeMoveSpeed = dashSpeed;
        dashCounter = dashLength;

        anim.SetTrigger("dash");

        PlayerHealthController.instance.MakeInvincible(dashInvincibility);

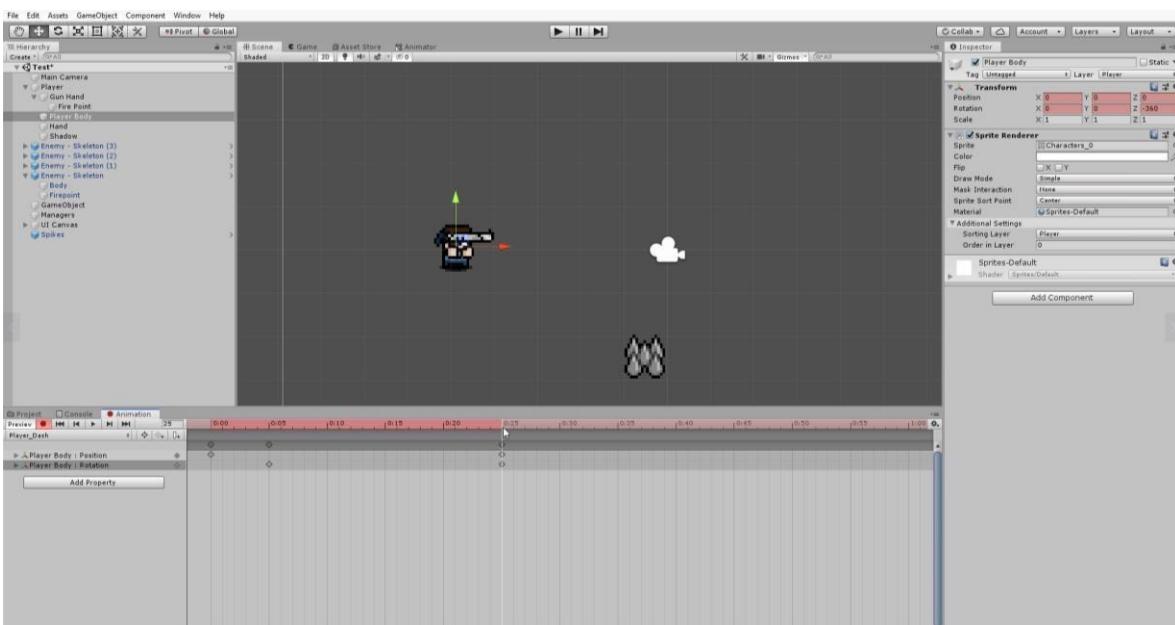
        AudioManager.instance.PlaySFX(8);
    }
}

//solo se activara el dash cuando estemos bajo el segundo de inmunidad
if (dashCounter > 0)
{
    dashCounter -= Time.deltaTime;
    if (dashCounter <= 0)
    {
        activeMoveSpeed = moveSpeed;
        dashCoolCounter = dashCooldown;
    }
}

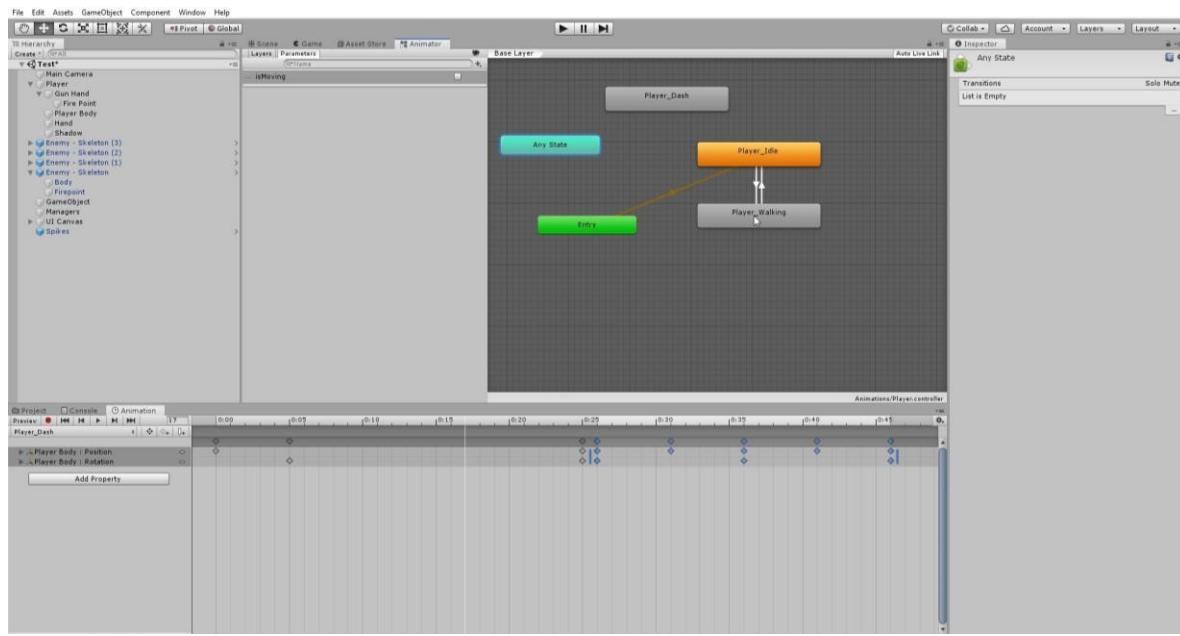
if (dashCoolCounter > 0)
{
    dashCoolCounter -= Time.deltaTime;
}

```

Creamos una animación para diferenciar cuando estemos usando dash.



Configuramos la jerarquía de animación.

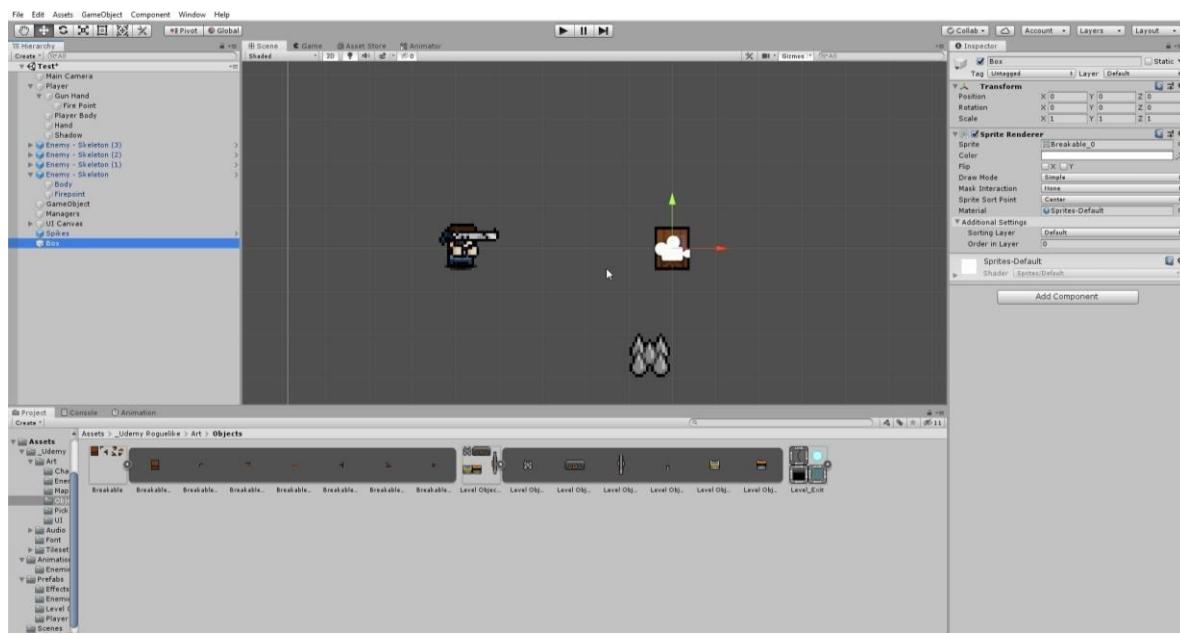


En el script asignamos que cuando se realice el dash se activara la animación.

```
activeMoveSpeed = dashSpeed;
dashCounter = dashLength;

//activara la animacion de dash
anim.SetTrigger("dash");
```

Crearemos nuevo objeto que será una caja sorpresa.



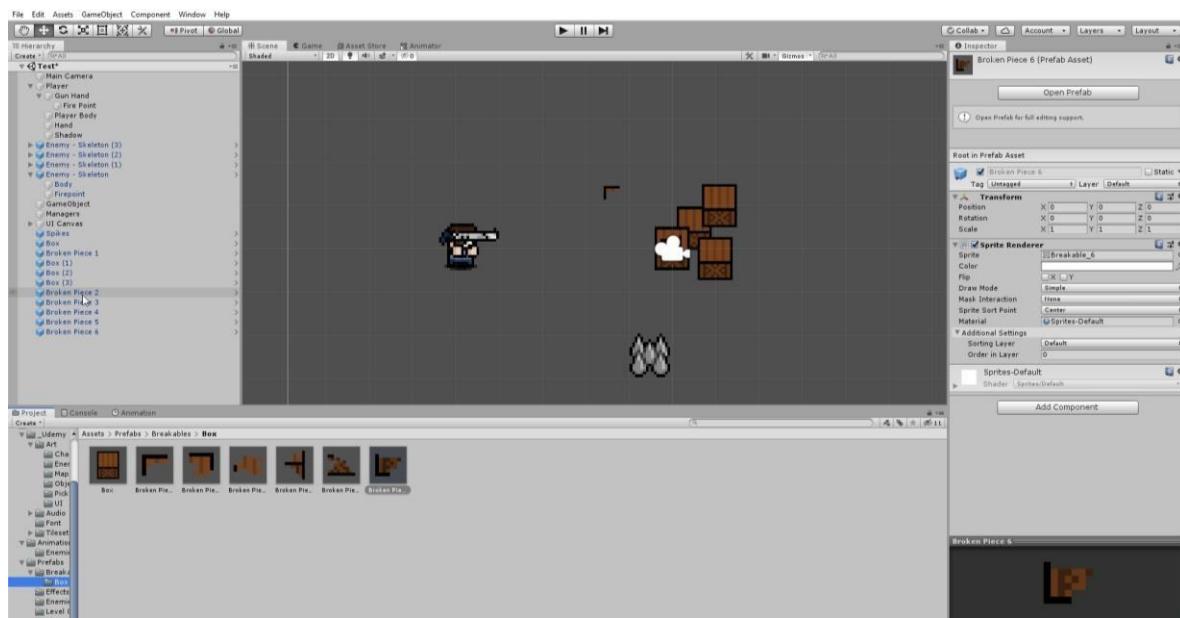
Asignamos que este objeto solo se destruirá si es afectado por la bala de player o el poder dash.

```

    Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D other)
{
    //cada vez que player use dash contra un caja se destruira
    if (other.tag == "Player")
    {
        if (PlayerController.instance.dashCounter > 0)
        {
            Smash();
        }
    }
    //cada vez que colisione con una bala tambien se destruira
    if(other.tag == "PlayerBullet")
    {
        Smash();
    }
}

```

Ahora crearemos una animación para cuando la caja sea destruida.



Cuando se destruya la caja las piezas volaran lentamente hasta quedar estáticas.

```

//funcion para destruir objeto
2 referencias
public void Smash()
{
    Destroy(gameObject);

    AudioManager.instance.PlaySFX(0);

    //mostrar piezas rotas
    int piecesToDrop = Random.Range(1, maxPieces);

    for (int i = 0; i < piecesToDrop; i++)
    {
        int randomPiece = Random.Range(0, brokenPieces.Length);

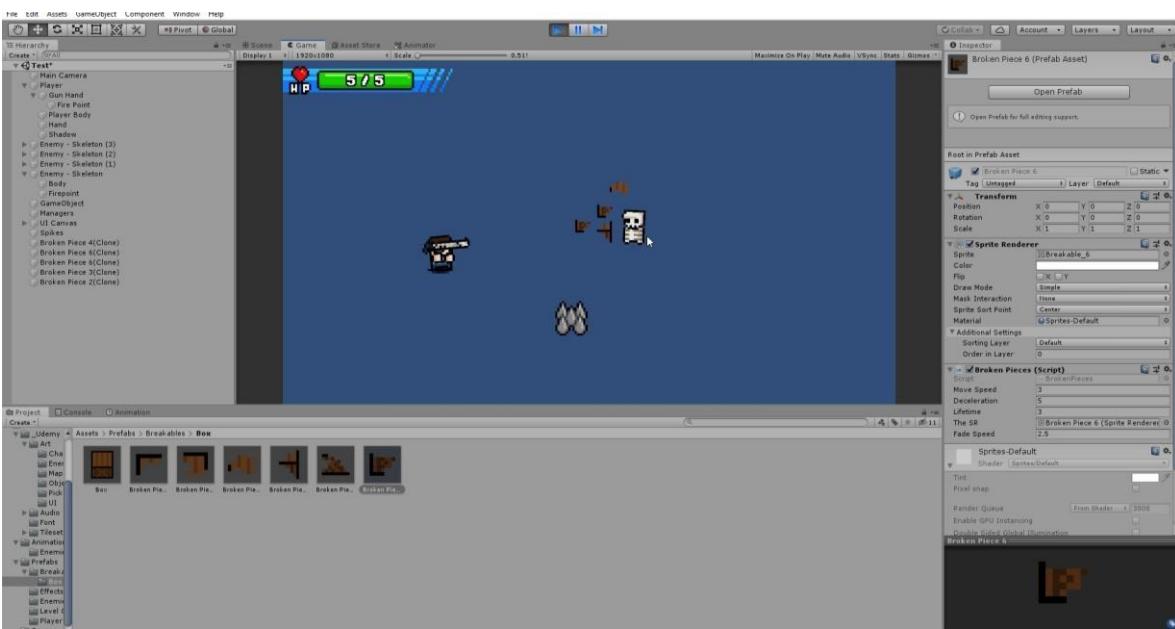
        Instantiate(brokenPieces[randomPiece], transform.position, transform.rotation);
    }

    //soltar piezas
    if (shouldDropItem)
    {
        float dropChance = Random.Range(0f, 100f);

        if (dropChance < itemDropPercent)
        {
            int randomItem = Random.Range(0, itemsToDrop.Length);

            Instantiate(itemsToDrop[randomItem], transform.position, transform.rotation);
        }
    }
}

```



Creamos la script para que cuando las piezas estén estáticas se destruyan.

```

//las piezas se desplazaran lentamente hasta quedarse quietas
transform.position += moveDirection * Time.deltaTime;

moveDirection = Vector3.Lerp(moveDirection, Vector3.zero, deceleration * Time.deltaTime);

lifetime -= Time.deltaTime;

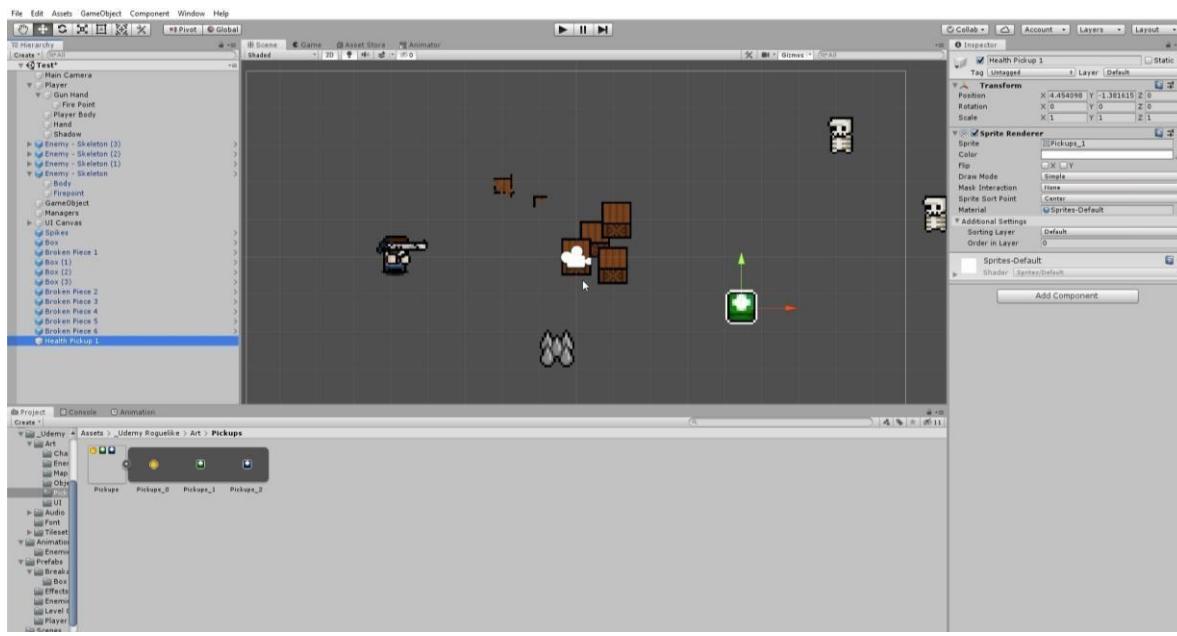
//si las piezas estan estaticas se destruiran
if(lifetime < 0)
{
    theSR.color = new Color(theSR.color.r, theSR.color.g, theSR.color.b, Mathf.MoveTowards(theSR.color.a, 0f, fadeSpeed * Time.deltaTime));

    if (theSR.color.a == 0f)
    {
        Destroy(gameObject);
    }
}

```

Sección 8: Pickups

Creamos nuestro objeto salud



Creamos el código para salud, y lo instanciamos con la salud del player.

```

public int healAmount = 1; //salud aumentada

public float waitToBeCollected = .5f;

// Start is called before the first frame update
@ Mensaje de Unity | 0 referencias
void Start()
{
}

// Update is called once per frame
@ Mensaje de Unity | 0 referencias
void Update()
{
    if(waitToBeCollected > 0)
    {
        waitToBeCollected -= Time.deltaTime;
    }
}

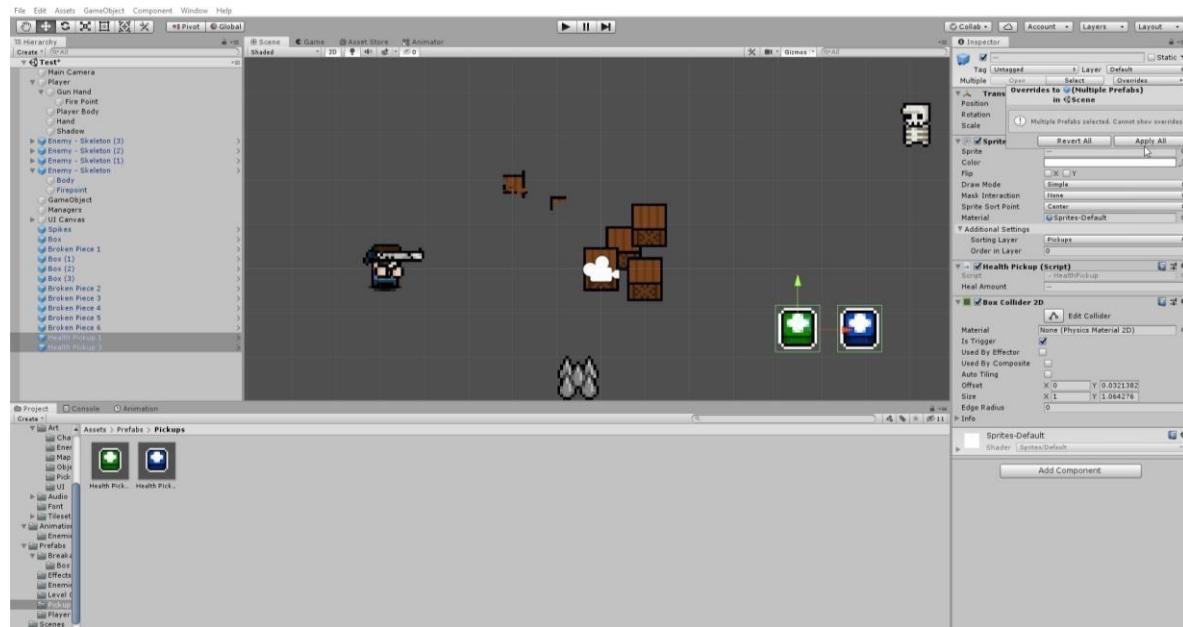
//cada vez que player tenga colision con este objeto aumentara su salud
@ Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Player" && waitToBeCollected <= 0)
    {
        PlayerHealthController.instance.HealPlayer(healAmount);

        Destroy(gameObject);

        AudioManager.instance.PlaySFX(7);
    }
}

```

Ahora creamos un prefab para que las cajas puedan dropear salud aleatoriamente.



Creamos el código para tenga 25% de probabilidad de dropear un objeto salud.

```

//soltar piezas aleatoriamente
if (shouldDropItem)
{
    float dropChance = Random.Range(0f, 100f);

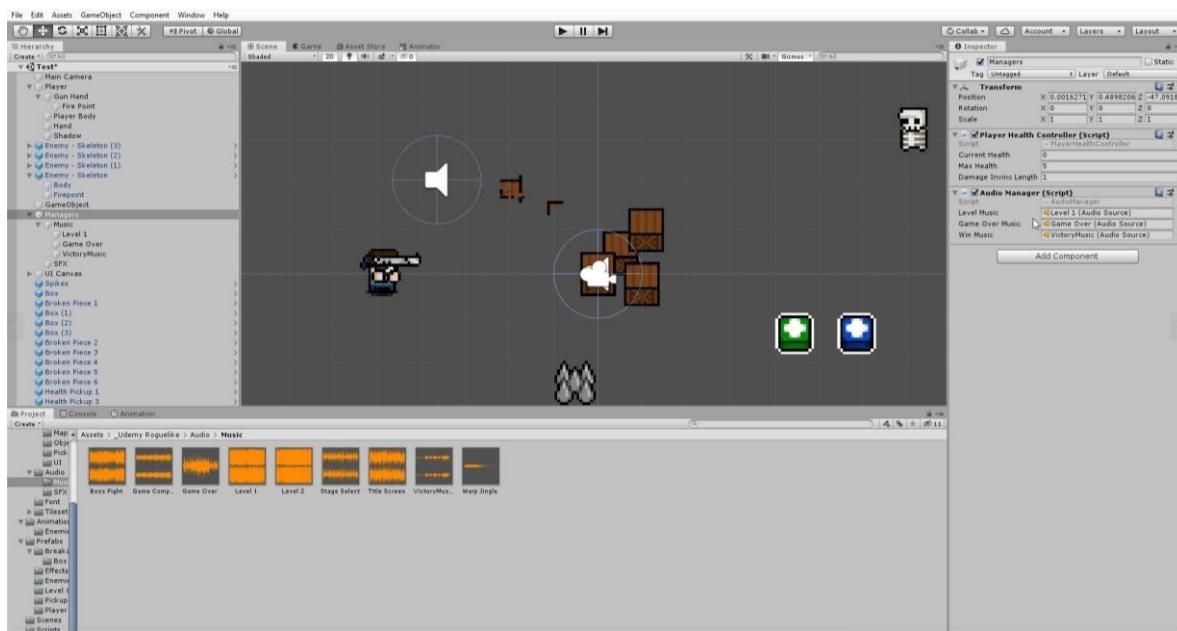
    if (dropChance < itemDropPercent)
    {
        int randomItem = Random.Range(0, itemsToDrop.Length);

        Instantiate(itemsToDrop[randomItem], transform.position, transform.rotation);
    }
}

```

Sección 9: Audio

Asignamos los audios para los diferentes escenarios.



En el script indicamos cuando termina e inicia el audio correspondiente

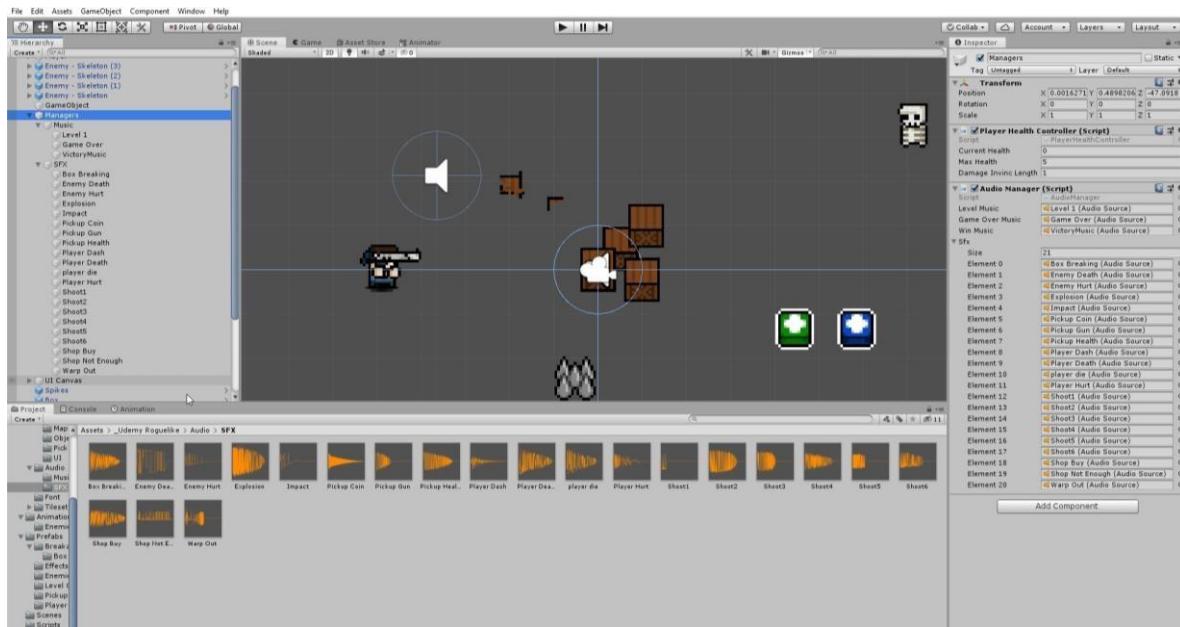
```

//controlar el audio segun los escenarios
1 referencia
public void PlayGameOver()
{
    levelMusic.Stop();
    gameOverMusic.Play();
}

1 referencia
public void PlayLevelWin()
{
    levelMusic.Stop();
    winMusic.Play();
}

```

Para los efectos de sonido de cada acción y objeto, creamos una carpeta de efectos y arrastramos todos nuestros audios.



En la script definimos como un array cuando se reproducirá cada efecto.

```
//instanciamos el audio efecto para cada acción
16 referencias
public void PlaySFX(int sfxToPlay)
{
    sfx[sfxToPlay].Stop();
    sfx[sfxToPlay].Play();
}
```

Instanciamos la posición de cada audio dentro de su sección propia, por ejemplo aquí instanciamos para cuando el enemigo ataque y cuando muera.

```
//intanciamos su audio para atacar
AudioManager.instance.PlaySFX(2);

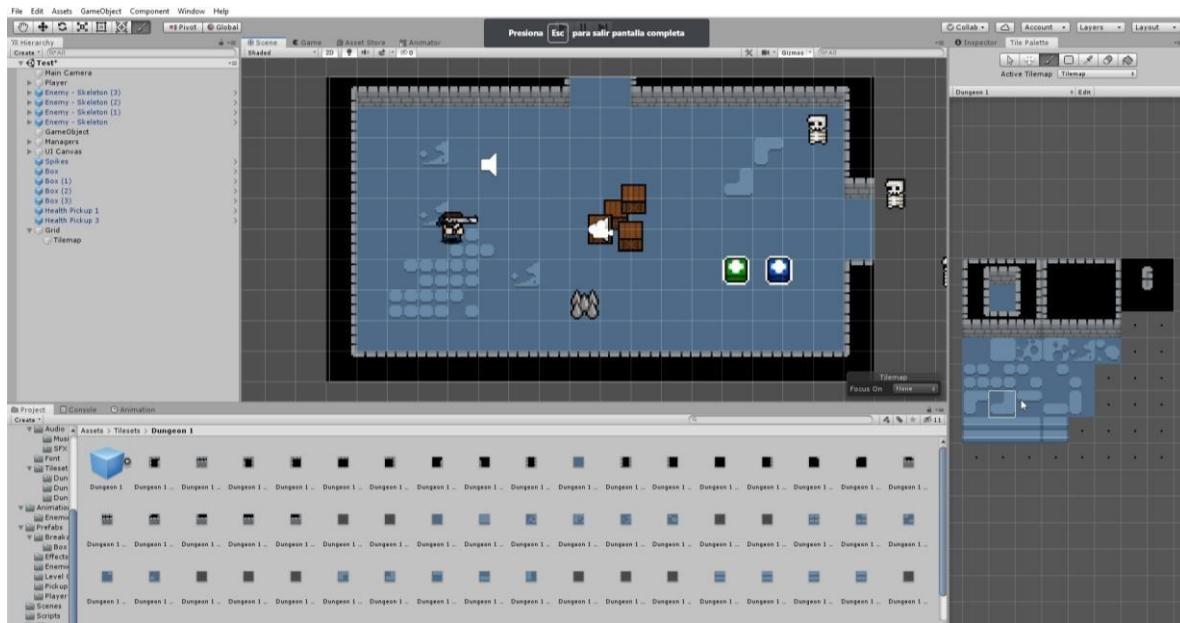
//salpicadura de sangre cuando enemigo sea colisionado con la bala
Instantiate(hitEffect, transform.position, transform.rotation);

//condicional para ver si la salud del enemigo
if(health <= 0)
{
    Destroy(gameObject);

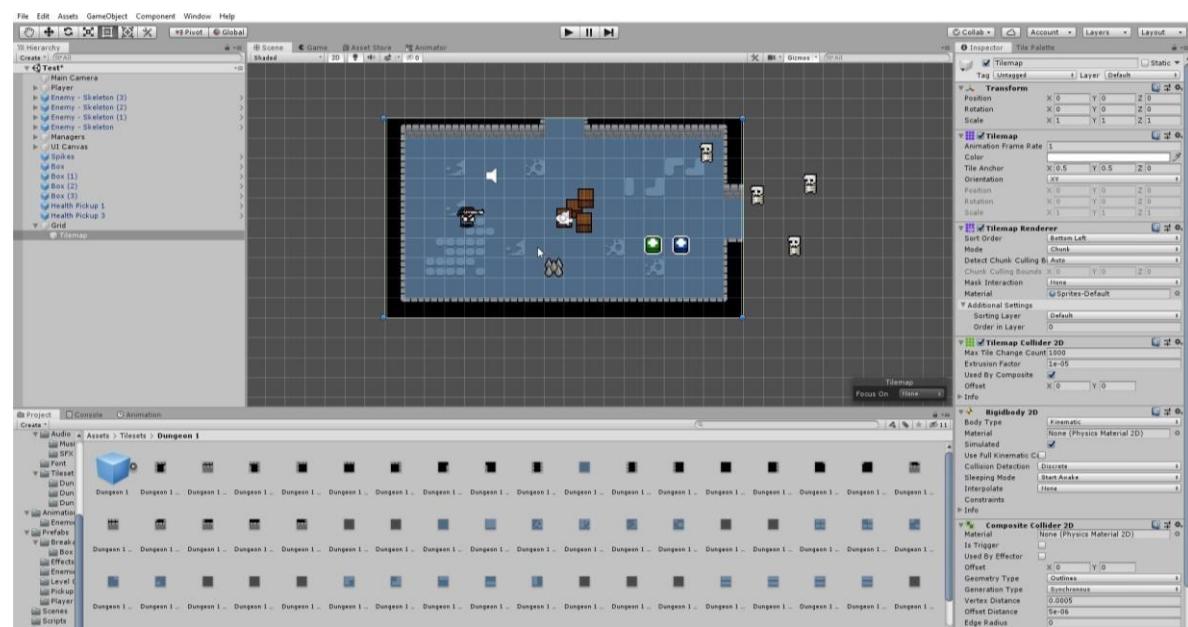
    //instanaciamos su propio audio de muerte
    AudioManager.instance.PlaySFX(1);
}
```

Sección 10: Crear niveles

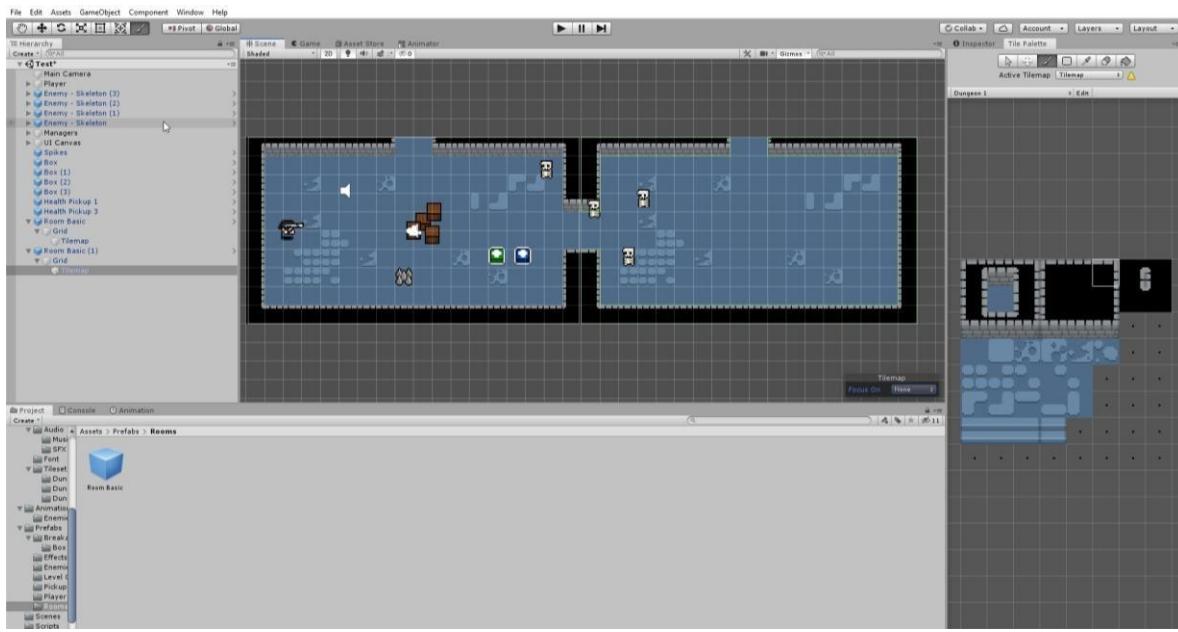
Con la herramienta Tile Palette diseñamos nuestro primer escenario.



A nuestras paredes le agregamos un tilemap collider y un composite collider para que nuestro player no pueda pasar sobre ellas.



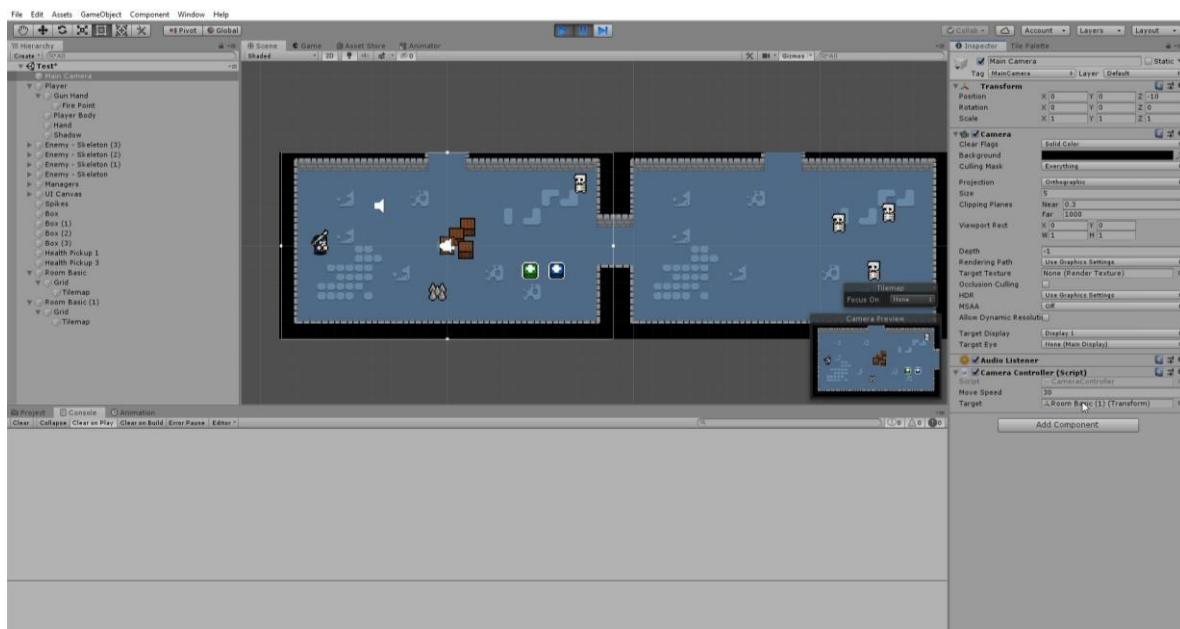
Ahora diseñamos un segundo escenario es secuencia.



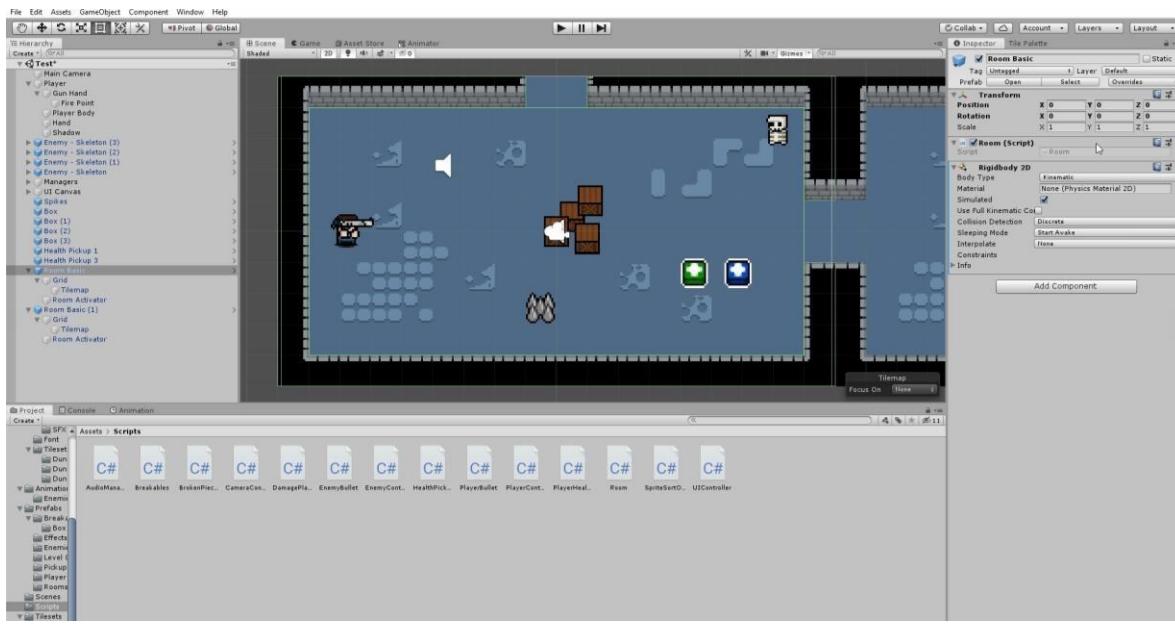
Agregamos un script para controlar la cámara y enfoque estáticamente todo el salón.

```
//controlar el enfoque de camara para que sea estatico por habitaciones
if (target != null)
{
    transform.position = Vector3.MoveTowards(transform.position, new Vector3(target.position.x, target.position.y,
}

if(Input.GetKeyDown(KeyCode.M) && !isBossRoom)
{
    if(!bigMapActive)
    {
        ActivateBigMap();
    } else
    {
        DeactivateBigMap();
    }
}
```



Agregamos un rigibody al centro de nuestra habitación para que este rote cada vez que cambiemos de habitación.



En el script indicamos que se actualice a la habitación en la que entramos.

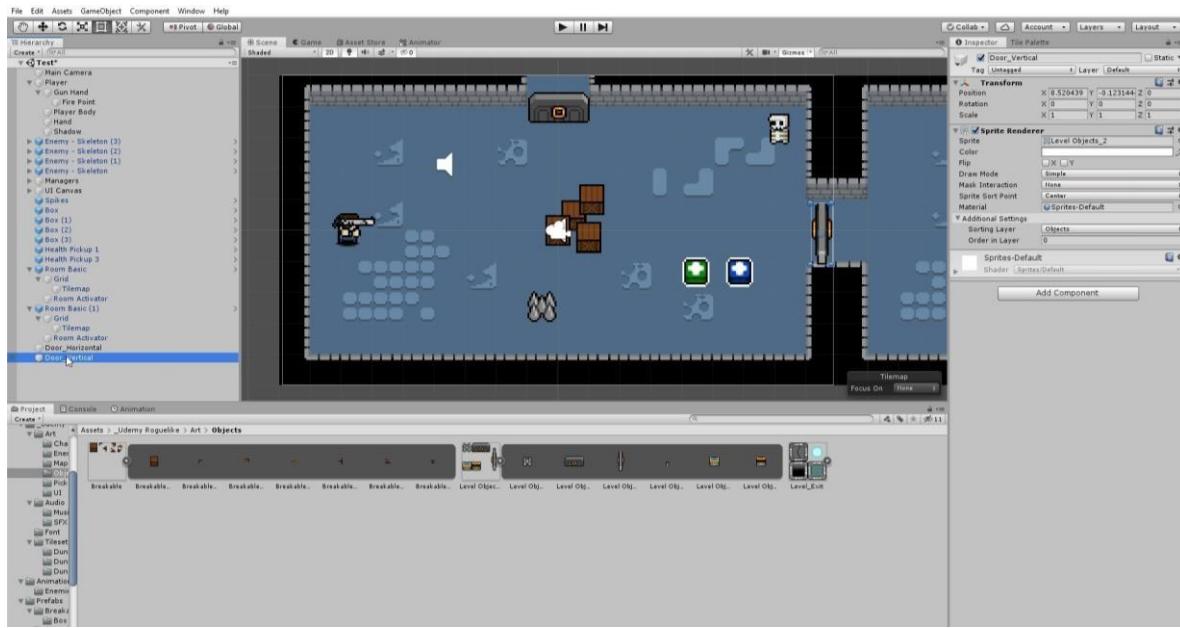
```
private void Awake()
{
    instance = this;
}

// Start is called before the first frame update
void Start()
{
}

// Update is called once per frame
void Update()
{
    if (target != null)
    {
        transform.position = Vector3.MoveTowards(transform.position, new Vector3(target
    }
}

public void ChangeTarget(Transform newTarget)
{
    target = newTarget;
}
```

Agregamos puertas para mayor dificultad.



Y agregamos código para que solo se abran cuando no queda ni un solo objeto enemigo.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Player")
    {
        //cambiar camara al entrar a otra habitacion
        CameraController.instance.ChangeTarget(transform);

        //cerrar puertas al entrar a nueva habitacion
        if(closeWhenEntered)
        {
            foreach(GameObject door in doors)
            {
                door.SetActive(true);
            }
        }

        roomActive = true;

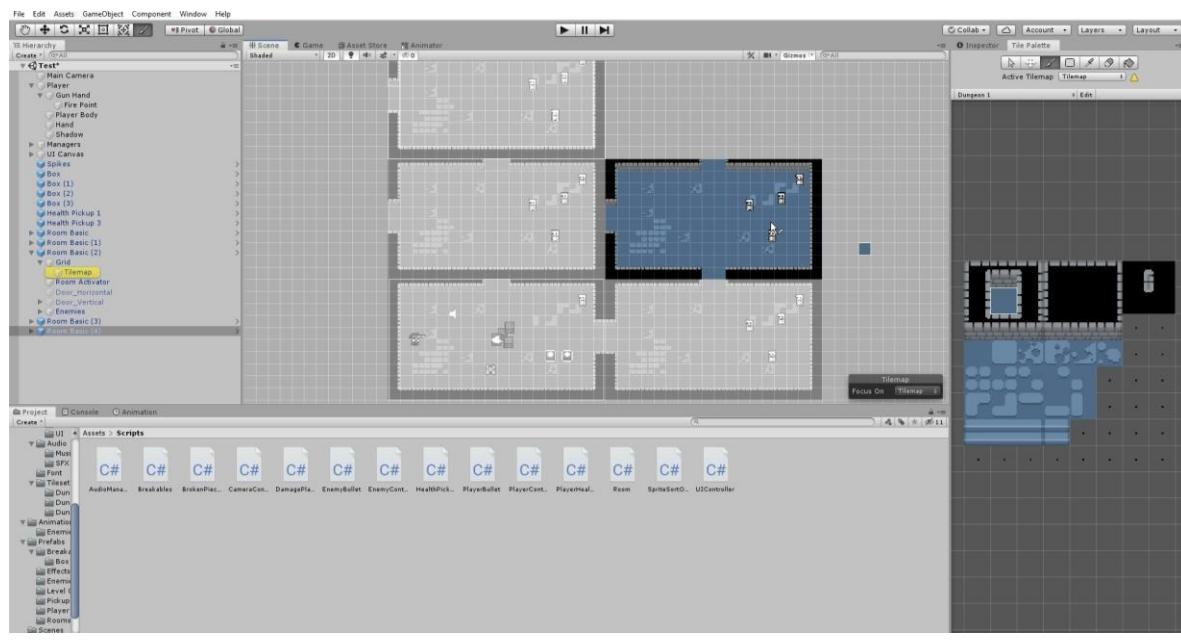
        mapHider.SetActive(false);
    }
}

if(enemies.Count > 0 && roomActive && openWhenEnemiesCleared)
{
    for(int i = 0; i < enemies.Count; i++)
    {
        if(enemies[i] == null)
        {
            enemies.RemoveAt(i);

            i--;
        }
    }

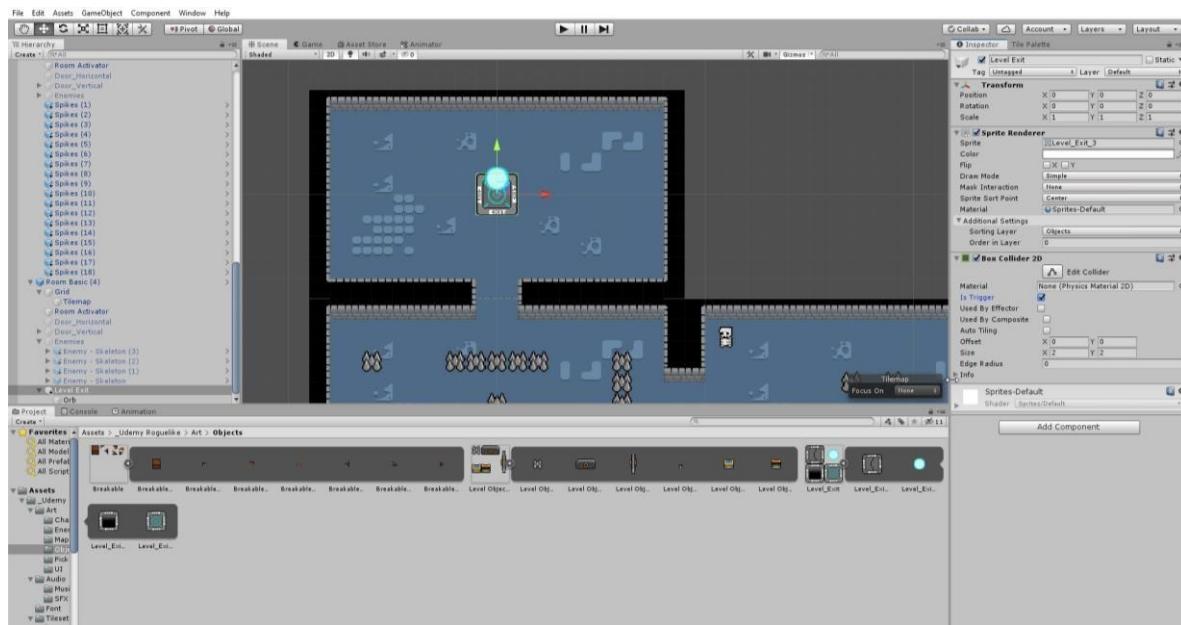
    if(enemies.Count == 0)
    {
        foreach (GameObject door in doors)
        {
            door.SetActive(false);
        }
    }
}
```

Terminamos de diseñar nuestro primer nivel.

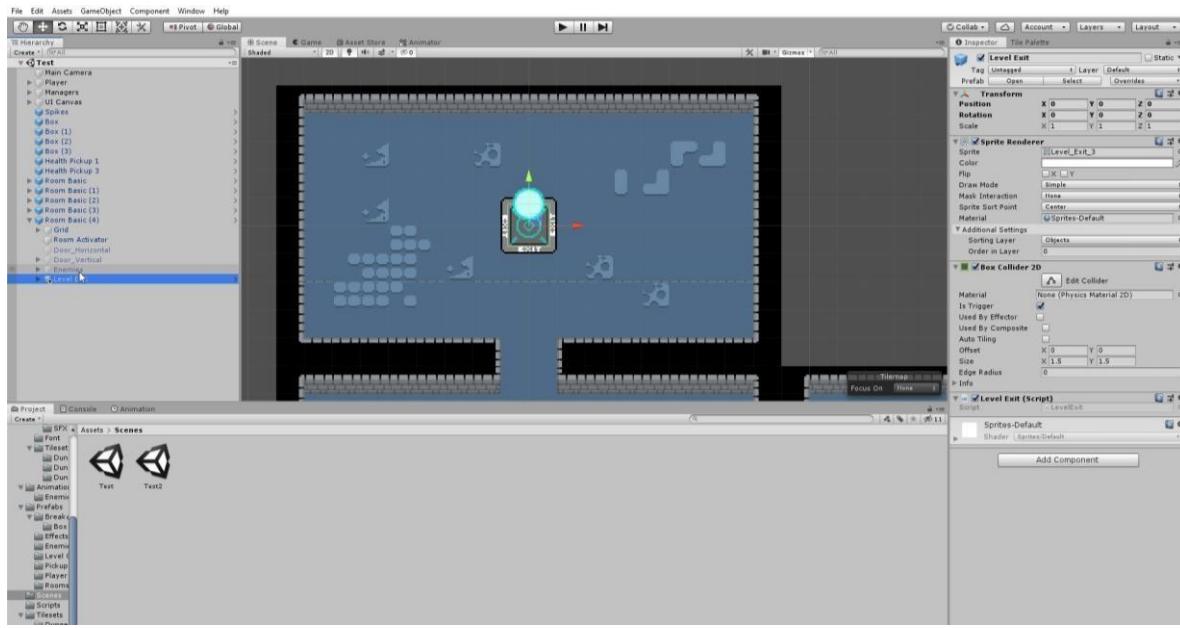


Sección 11: Cambio de nivel

Creamos un objeto de salida.



Y creamos una nueva escena para nuestro siguiente nivel.



Creamos un script para cuando player interactúe con el objeto exit cambie de escena.

```
① Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Player")
    {
        //Cambia de escena cuando player interctua con objeto exit
        SceneManager.LoadScene(levelToLoad);
    }
}

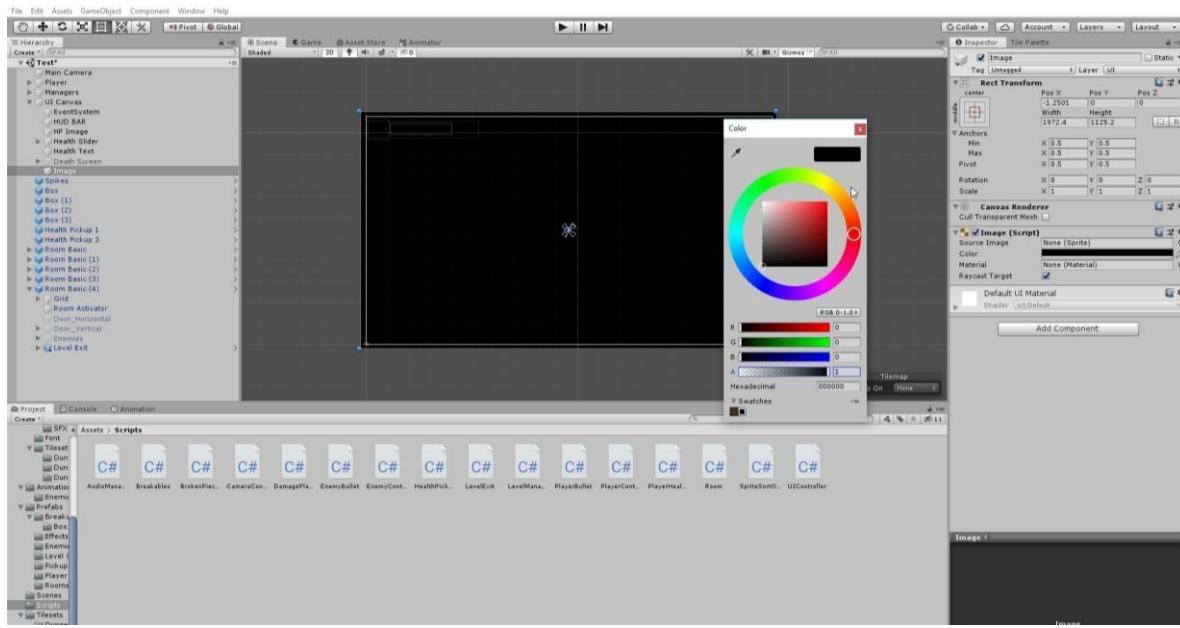
//intanciamos el audio para el cambio de escena
AudioManager.instance.PlayLevelWin();

//cuando es cambio escena player no puede moverse
PlayerController.instance.canMove = false;

UIController.instance.StartFadeToBlack();

//hace cambio de escena
yield return new WaitForSeconds(waitToLoad);
```

Creamos una imagen de color negro que será el intermediario entre escenas.



Creamos el script para que se active nuestra imagen, cuando estemos en medio de un cambio de escena.

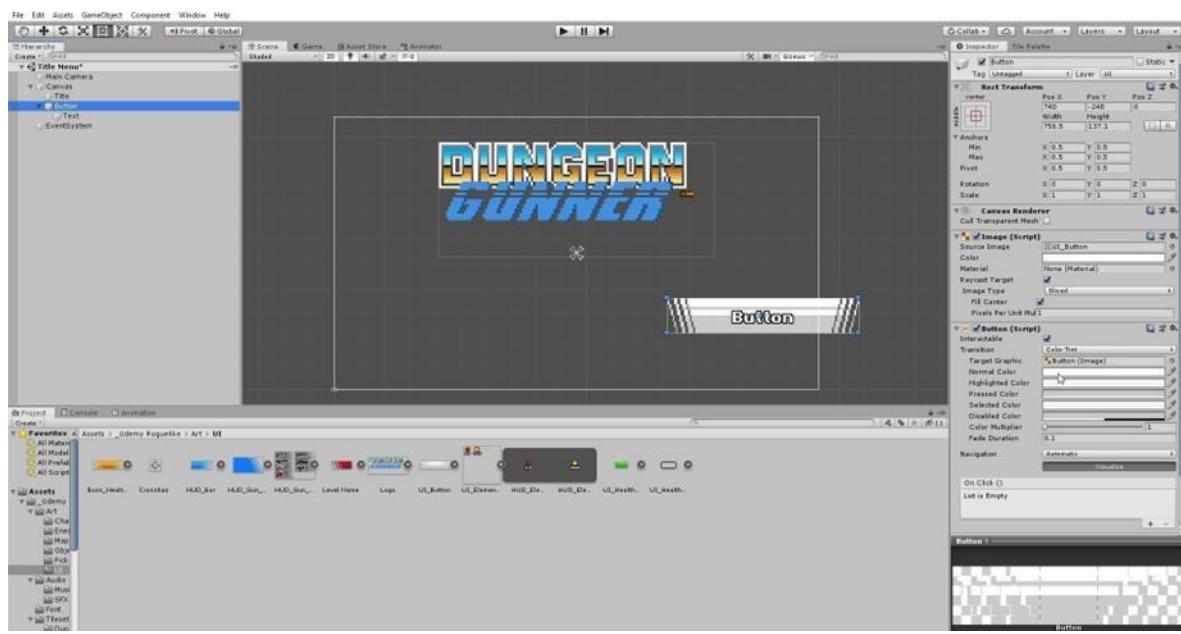
```
// Start is called before the first frame update
@Mensaje de Unity | 0 referencias
void Start()
{
    //cuando haya cambio de escena se activa fondo negro
    fadeOutBlack = true;
    fadeToBlack = false;

    currentGun.sprite = PlayerController.instance.availableGuns[PlayerController.instance.currentGun].gunUI;
    gunText.text = PlayerController.instance.availableGuns[PlayerController.instance.currentGun].weaponName;
}

// Update is called once per frame
@Mensaje de Unity | 0 referencias
void Update()
{
    //cuando haya pasado el tiempo regresamos a nuestra nueva escena
    if(fadeOutBlack)
    {
        fadeScreen.color = new Color(fadeScreen.color.r, fadeScreen.color.g, fadeScreen.color.b, Mathf.MoveTowards(fadeScreen.color.a, 0f));
        if(fadeScreen.color.a == 0f)
        {
            fadeOutBlack = false;
        }
    }
}
```

Sección 12: Menú

Diseñamos nuestros botones y nuestro menú.



En el script instanciamos las escenas que se ejecutan al presionar cada botón.

```
//button inicio
0 referencias
public void StartGame()
{
    //carga nueva escena
    SceneManager.LoadScene(levelToLoad);
}

//boton salir
0 referencias
public void ExitGame()
{
    Application.Quit();
}

//boton eliminar guardados
0 referencias
public void DeleteSave()
{
    deletePanel.SetActive(true);
}

//boton confirmar eliminar guardados
0 referencias
public void ConfirmDelete()
{
    deletePanel.SetActive(false);

    foreach(CharacterSelector theChar in charactersToDelete)
    {
        PlayerPrefs.SetInt(theChar.playerToSpawn.name, 0);
    }
}
```

Creamos el botón pause que funcionara con la tecla escape.

```
// Update is called once per frame
@ Mensaje de Unity | 0 referencias
void Update()
{
    //asignamos tecla para que se pause
    if(Input.GetKeyDown(KeyCode.Escape))
    {
        PauseUnpause();
    }
}
```

```

//boton pausa
2 referencias
public void PauseUnpause()
{
    //condicional para que este activo o no
    if(!isPaused)
    {
        UIController.instance.pauseMenu.SetActive(true);

        isPaused = true;

        Time.timeScale = 0f;
    } else
    {
        UIController.instance.pauseMenu.SetActive(false);

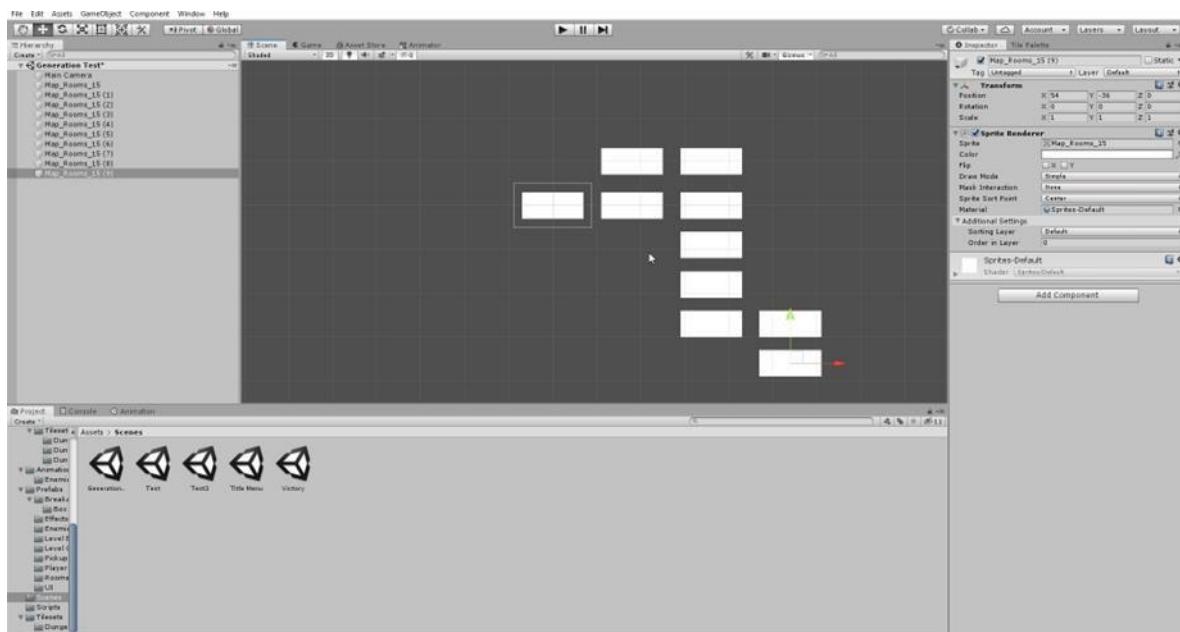
        isPaused = false;

        Time.timeScale = 1f;
    }
}

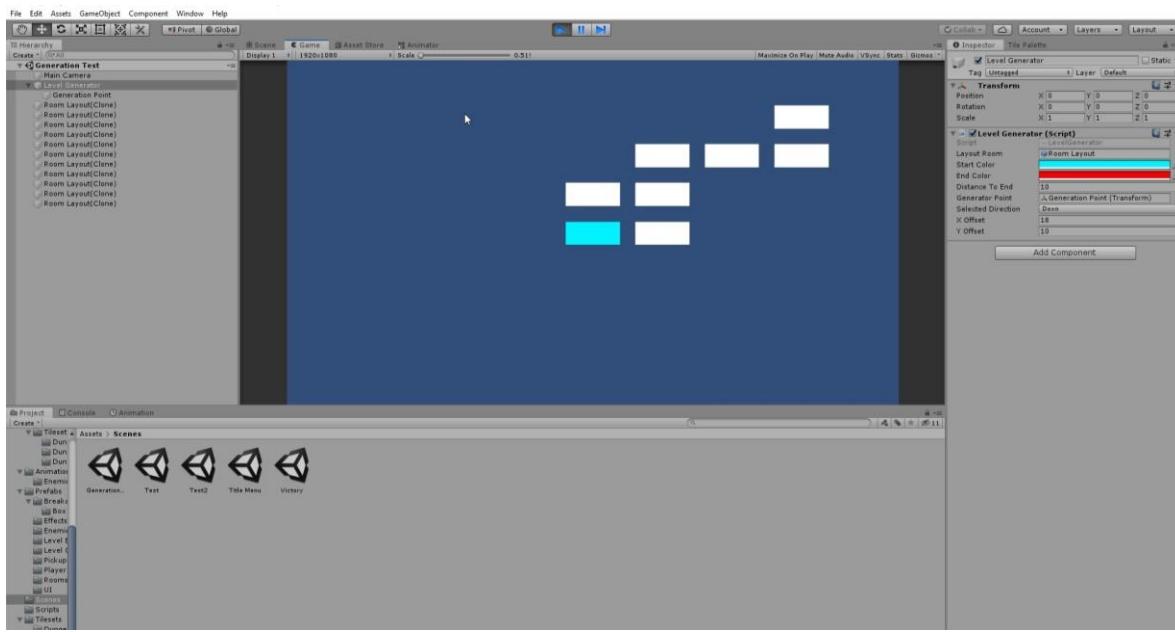
```

Sección 13: Generación de niveles procesales

Creamos nuestras habitaciones en cualquier orden ya que estas se ordenarán aleatoriamente.



Cuando player se encuentre en una habitación este cuadrante cambiara de color.



En el script indicamos cuando se sombreara el cuadrante.

```
// Update is called once per frame
void Update()
{
    //para que se active cada salon en el mapa mientras vamos explorando
    UNITY_EDITOR
    if(Input.GetKey(KeyCode.R))
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Y hará que los cuadrantes se ordenen aleatoriamente.

```
for(int i = 0; i < distanceToEnd; i++)
{
    //agregar habitaciones aleatoriamente
    GameObject newRoom = Instantiate(layoutRoom, generatorPoint.position, generatorPoint.rotation);

    layoutRoomObjects.Add(newRoom);

    if(i + 1 == distanceToEnd)
    {
        newRoom.GetComponent<SpriteRenderer>().color = endColor;
        layoutRoomObjects.RemoveAt(layoutRoomObjects.Count - 1);

        endRoom = newRoom;
    }

    selectedDirection = (Direction)Random.Range(0, 4);
    MoveGenerationPoint();

    //para que se visualice en el mapa mientras vamos explorando
    while (Physics2D.OverlapCircle(generatorPoint.position, .2f, whatIsRoom))
    {
        MoveGenerationPoint();
    }
}
```

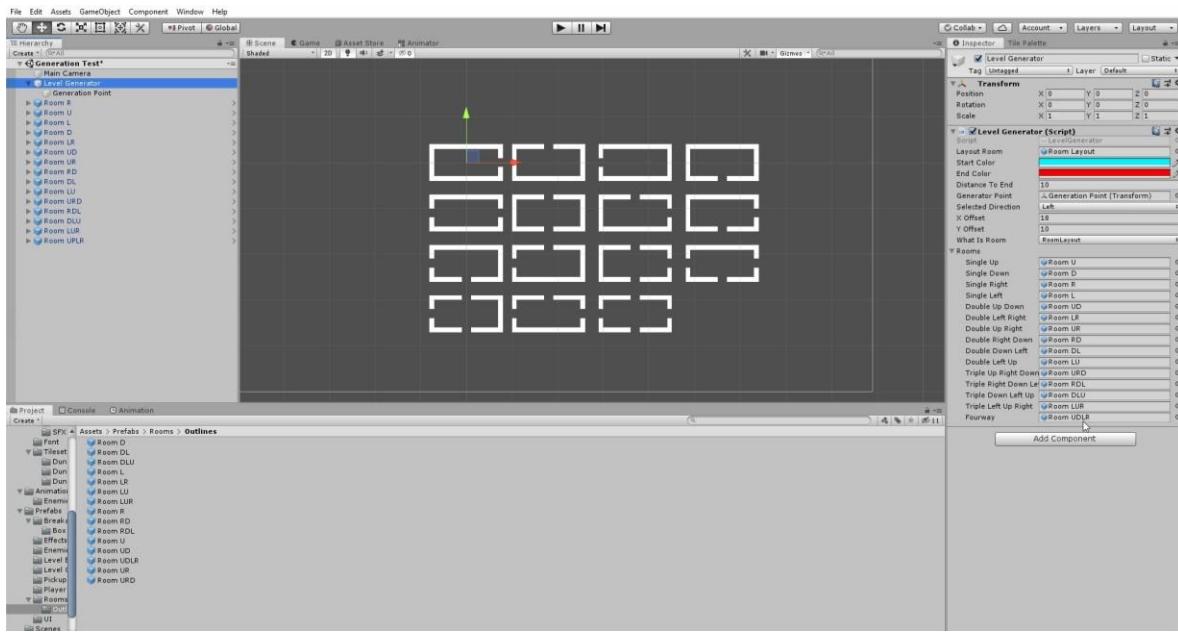
Ahora creamos un script para el contorno de nuestras habitaciones.

```

public class RoomPrefabs
{
    //declaramos todos los contornos de las habitaciones
    public GameObject singleUp, singleDown, singleRight, singleLeft,
    doubleUpDown, doubleLeftRight, doubleUpRight, doubleRightDown, doubleDownLeft, doubleLeftUp,
    tripleUpRightDown, tripleRightDownLeft, tripleDownLeftUp, tripleLeftUpRight,
    fourway;
}

```

Creamos un prefab con todos los diferentes contornos para nuestras habitaciones.



Al generar los cuadrantes aleatoriamente los contornos y sus salidas también deben ordenarse aleatoriamente para eso creamos el siguiente script.

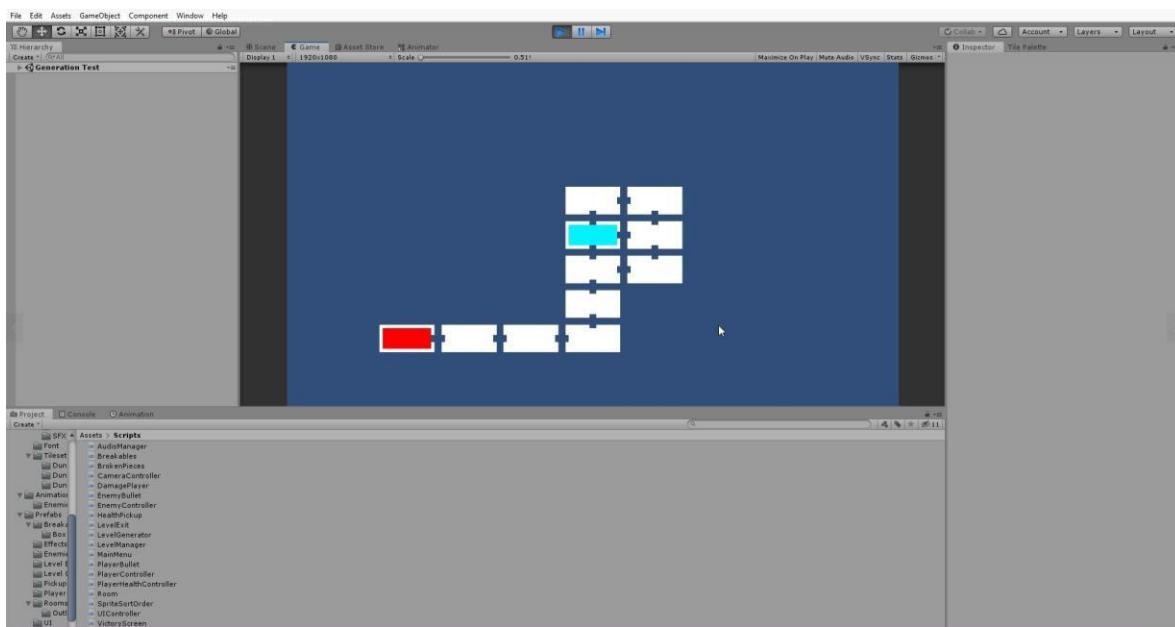
```

//detectar si hay una habitacion a los costados
bool roomAbove = Physics2D.OverlapCircle(roomPosition + new Vector3(0f, yOffset, 0f), .2f, whatIsRoom);
bool roomBelow = Physics2D.OverlapCircle(roomPosition + new Vector3(0f, -yOffset, 0f), .2f, whatIsRoom);
bool roomLeft = Physics2D.OverlapCircle(roomPosition + new Vector3(-xOffset, 0f, 0f), .2f, whatIsRoom);
bool roomRight = Physics2D.OverlapCircle(roomPosition + new Vector3(xOffset, 0f, 0f), .2f, whatIsRoom);

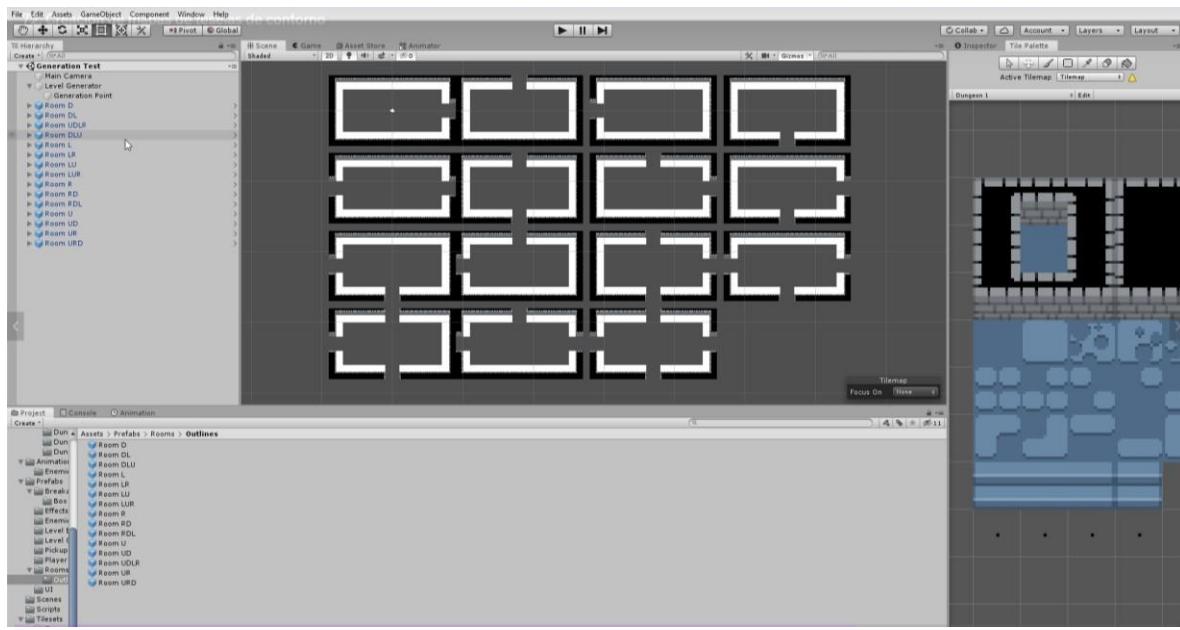
//comprobar si es que hay habitacion
int directionCount = 0;
if(roomAbove)
{
    directionCount++;
}
if (roomBelow)
{
    directionCount++;
}
if (roomLeft)
{
    directionCount++;
}
if (roomRight)
{
    directionCount++;
}

switch(directionCount)
{

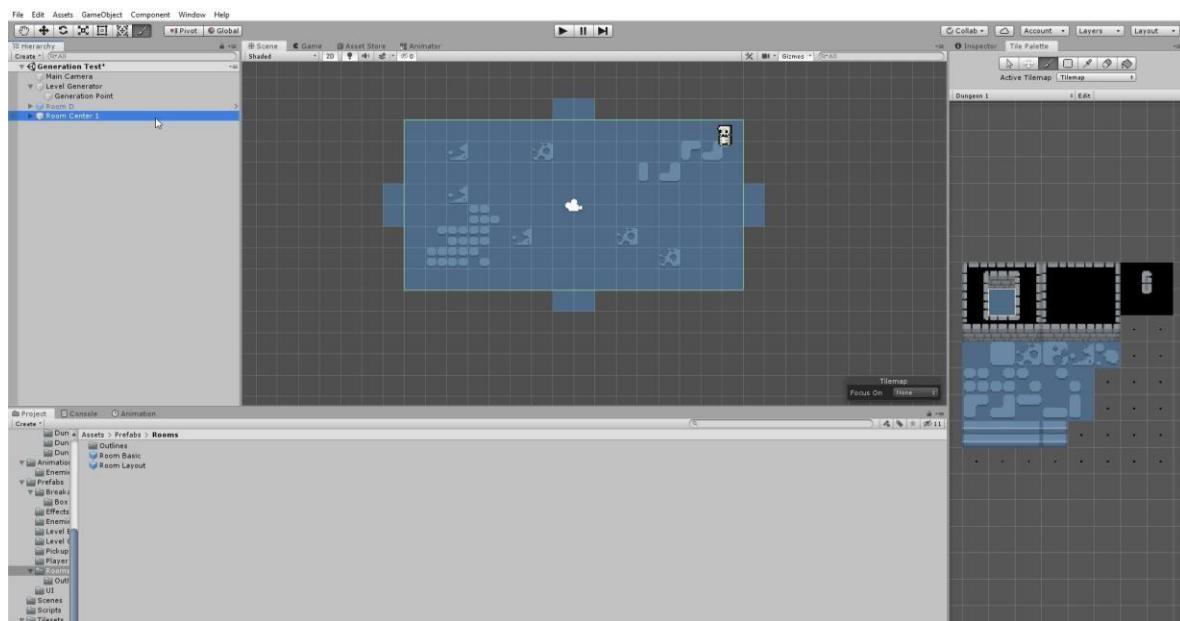
```



Ahora como hicimos anteriormente, regresamos a nuestro tile palette, y diseñamos las paredes para todos nuestros contornos.



Y rediseñamos nuestra habitación ya antes prefabricada.



Agregamos las puertas a los 4 extremos y copiamos el código ya antes creado, para que se abran si no hay enemigos.

```

// Start is called before the first frame update
@ Mensaje de Unity | 0 referencias
void Start()
{
    //abrir puerta si no hay enemigos
    if(openWhenEnemiesCleared)
    {
        theRoom.closeWhenEntered = true;
    }
}

// Update is called once per frame
@ Mensaje de Unity | 0 referencias
void Update()
{
    //controlar de que no haya ningun objeto enemigo en la habitacion para abrir puerta
    if (enemies.Count > 0 && theRoom.roomActive && openWhenEnemiesCleared)
    {
        for (int i = 0; i < enemies.Count; i++)
        {
            if (enemies[i] == null)
            {
                enemies.RemoveAt(i);

                i--;
            }
        }

        //si no hay enemigos se abre
        if (enemies.Count == 0)
        {
            theRoom.OpenDoors();
        }
    }
}

```

Creamos un script para aparecer en una habitación aleatoriamente, pero que no sea una habitación donde se encuentre la salida a otro nivel.

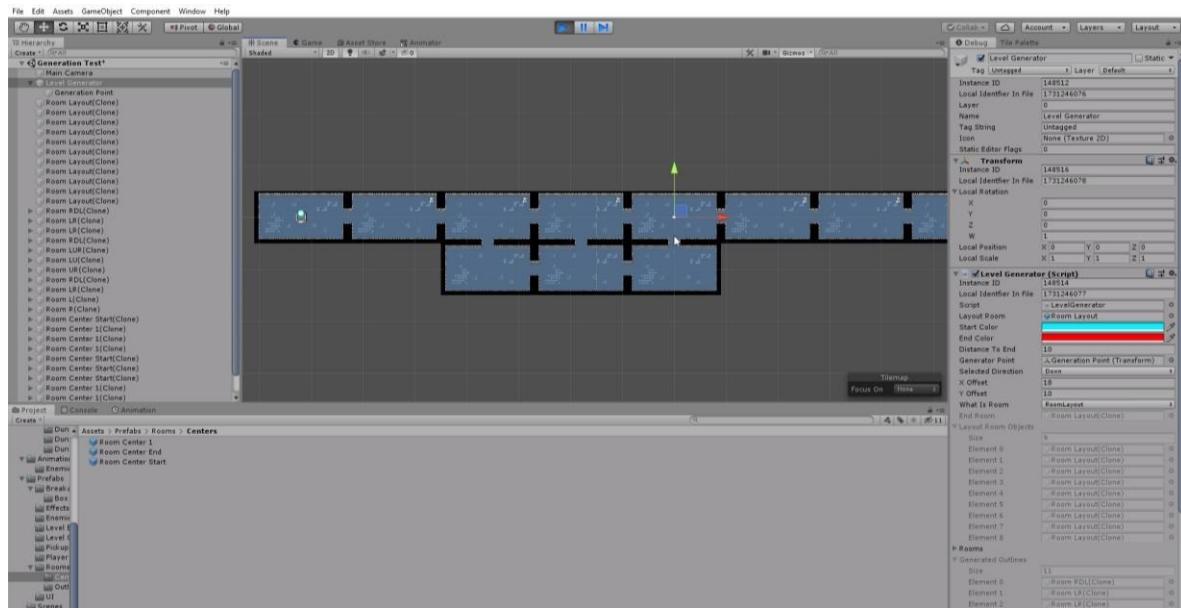
```

//validaciones para que nunca podamos aparecer la habitacion de salida
if(outline.transform.position == Vector3.zero)
{
    Instantiate(centerStart, outline.transform.position, transform.rotation).theRoom = outline.GetComponent<Room>();
    generateCenter = false;
}

if(outline.transform.position == endRoom.transform.position)
{
    Instantiate(centerEnd, outline.transform.position, transform.rotation).theRoom = outline.GetComponent<Room>();
    generateCenter = false;
}

if(includeShop)
{
    if (outline.transform.position == shopRoom.transform.position)
    {
        Instantiate(centerShop, outline.transform.position, transform.rotation).theRoom = outline.GetComponent<Room>();
        generateCenter = false;
    }
}

```



Sección 14: Crear más enemigos

Agregaremos diferentes tipos de enemigos



Colocamos el siguiente código en el script EnemyController para que el enemigo nos persiga desde cierta distancia

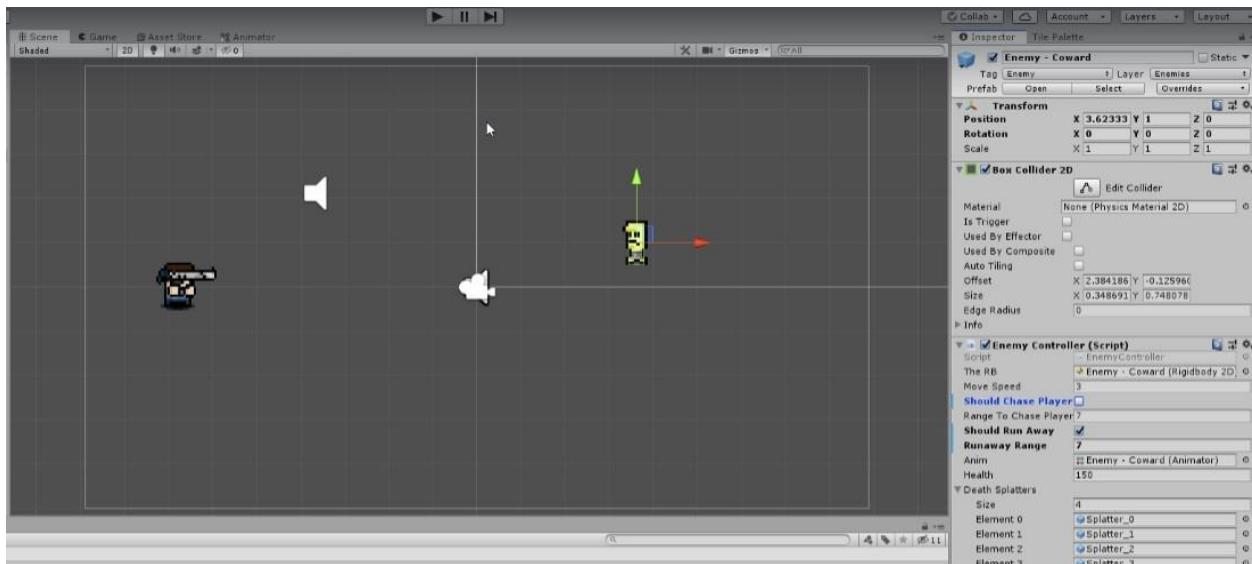
```
[Header("Chase Player")]
public bool shouldChasePlayer;
public float rangeToChasePlayer; //rango para que empiece a perseguirnos
private Vector3 moveDirection; //para que se mueva el enemigo

[Header("Run Away")]

public bool shouldRunAway;
public float runawayRange;
```

```
//Condicional para que el enemigo huya
if(shouldRunAway && Vector3.Distance(transform.position, PlayerController.instance.transform.position) < runawayRange)
{
    moveDirection = transform.position - PlayerController.instance.transform.position;
}
```

Seleccionamos a nuestro enemigo y marcaremos la casilla “Should chase player” y le asignamos un rango



Introducimos el siguiente código en el script de EnemyController, para que el enemigo deambule

```

public bool shouldWander;
public float wanderLength, pauseLength;
private float wanderCounter, pauseCounter;
private Vector3 wanderDirection;

// update is called once per frame
@Mensaje de Unity | 0 referencias
void Update()
{
    //controlar si el enemigo se encuentra visible en la pantalla recien podra disparar
    if(theBody.isVisible &amp; PlayerController.instance.gameObject.activeInHierarchy)
    {
        moveDirection = Vector3.zero;

        //condicional para que el enemigo se diriga hacia el player si este esta a un rango.
        if (Vector3.Distance(transform.position, PlayerController.instance.transform.position) < rangeToChasePlayer && shouldChasePlayer)
        {
            moveDirection = PlayerController.instance.transform.position - transform.position;
        } else
        {
            if(shouldWander)
            {
                if(wanderCounter > 0)
                {
                    wanderCounter -= Time.deltaTime;

                    //mueve al enemigo
                    moveDirection = wanderDirection;

                    if(wanderCounter <= 0)
                    {
                        pauseCounter = Random.Range(pauseLength * .75f, pauseLength * 1.25f);
                    }
                }

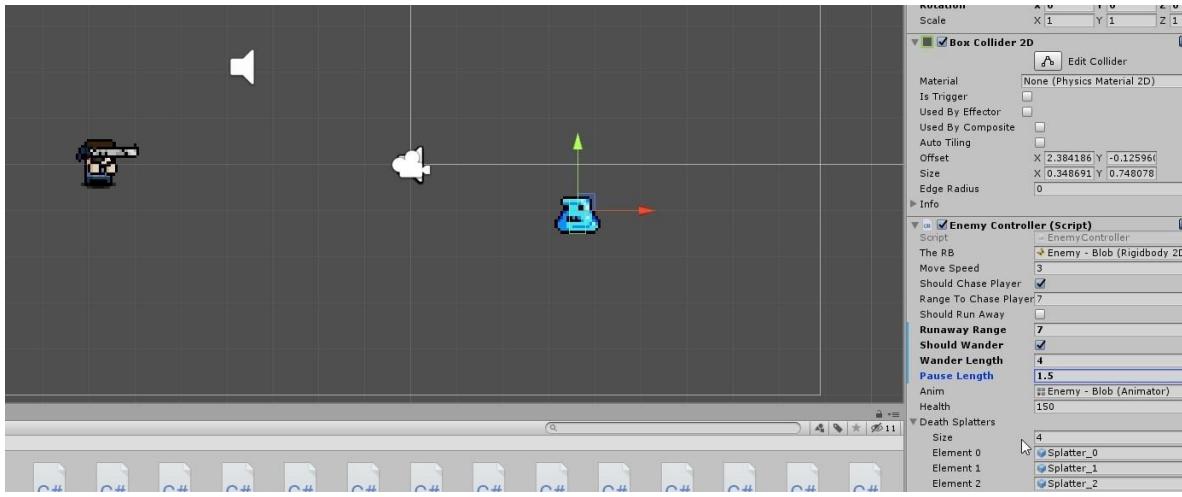
                if(pauseCounter > 0)
                {
                    pauseCounter -= Time.deltaTime;

                    if(pauseCounter <= 0)
                    {
                        wanderCounter = Random.Range(wanderLength * .75f, wanderLength * 1.25f);

                        wanderDirection = new Vector3(Random.Range(-1f, 1f), Random.Range(-1f, 1f), 0f);
                    }
                }
            }
        }
    }
}

```

Elegiremos nuestro siguiente enemigo y marcaremos la casilla “Should wonder” y le pondremos una pausa para que se detenga cada cierto tiempo



En el script EnemyController colocamos el siguiente código que hará que servirá para que nuestro enemigo patrulle

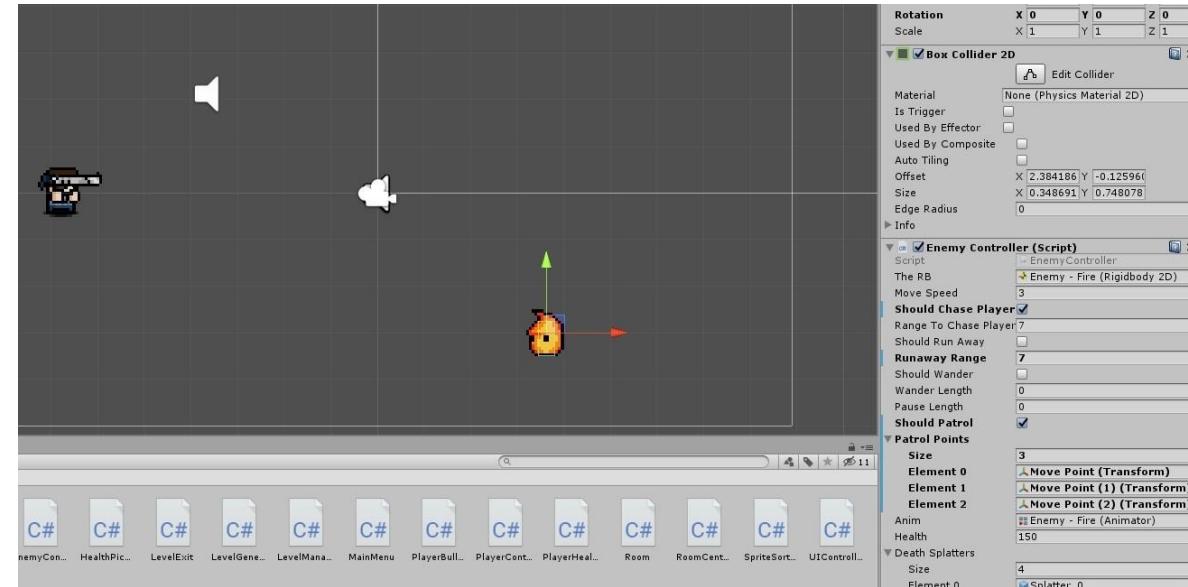
```
// variables de patrullaje

public bool shouldPatrol;
public Transform[] patrolPoints;
private int currentPatrolPoint;

//Patrullaje del enemigo
if(shouldPatrol)
{
    moveDirection = patrolPoints[currentPatrolPoint].position - transform.position;

    if(Vector3.Distance(transform.position, patrolPoints[currentPatrolPoint].position) < .2f)
    {
        currentPatrolPoint++;
        if(currentPatrolPoint >= patrolPoints.Length)
        {
            currentPatrolPoint = 0;
        }
    }
}
```

Entonces seleccionamos un nuevo enemigo y le marcamos la opción de “should patrol” y le agregamos los puntos que patrullará



Con esto ya podemos agregar a nuestros enemigos en las habitaciones.

Sección 15: Crear monedas

Entonces ahora procederemos a agregar monedas a nuestro juego, en el script LevelManager agregamos el siguiente código

```

① Script de Unity | 10 referencias
public class LevelManager : MonoBehaviour
{
    public static LevelManager instance;

    public float waitToLoad = 4f;

    public string nextLevel;

    public bool isPaused;

    public int currentCoins;

    public Transform startPoint;
}

1 referencia
public void GetCoins(int amount)
{
    currentCoins += amount;
}

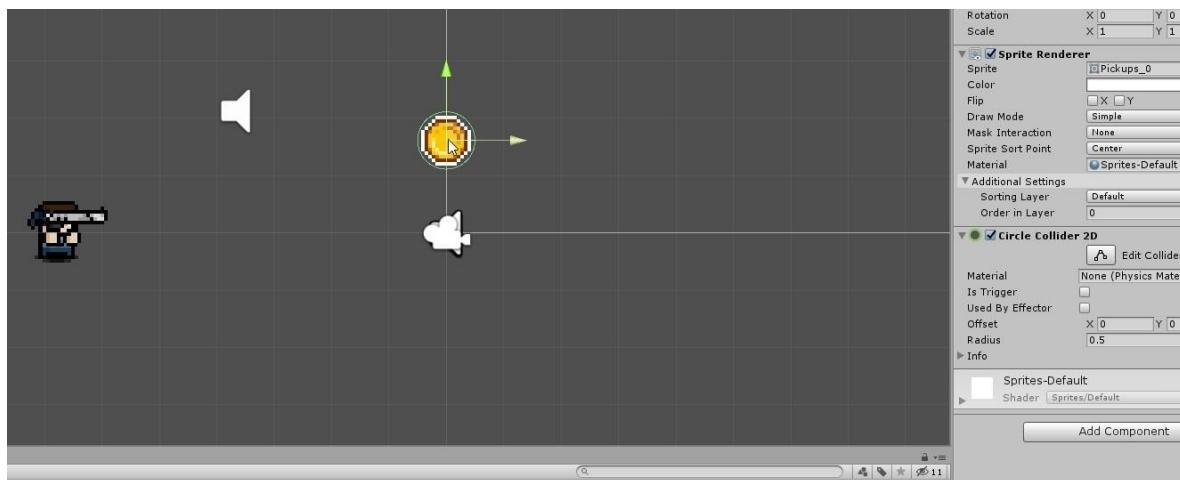
1 referencia
public void SpendCoins(int amount)
{
    currentCoins -= amount;

    if(currentCoins < 0)
    {
        currentCoins = 0;
    }
}

```

Esto ayudará a controlar las monedas que ganemos

Ahora colocaremos la moneda y le agregaremos el componente de “Circle Collider”



Crearemos un nuevo script llamado Coinpickup e ingresamos el siguiente código, donde las monedas se sumarán cada vez que choquemos con alguna y también podemos elegir el sonido que le pondremos

```
(@ Script de Unity | 0 referencias
public class CoinPickup : MonoBehaviour
{
    public int coinValue = 1;

    public float waitToBeCollected;

    // Start is called before the first frame update
    @ Mensaje de Unity | 0 referencias
    void Start()
    {

    }

    // Update is called once per frame
    @ Mensaje de Unity | 0 referencias
    void Update()
    {
        if (waitToBeCollected > 0)
        {
            waitToBeCollected -= Time.deltaTime;
        }
    }

    // metodo para recolectar moendas al chocar con ellas
    @ Mensaje de Unity | 0 referencias
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player" && waitToBeCollected <= 0)
        {
            LevelManager.instance.GetCoins(coinValue);

            Destroy(gameObject);

            AudioManager.instance.PlaySFX(5);
        }
    }
}
```

Ahora agregamos la interfaz de las monedas para mostrar su cantidad



En el script UIController declaramos la variable del texto de las monedas

```

public class UIController : MonoBehaviour
{
    public static UIController instance;

    public Slider healthSlider;
    public Text healthText, coinText;
}

```

En el script Level Manager agregamos el siguiente código para que muestre la cantidad de monedas que tenemos

```

void Start()
{
    Time.timeScale = 1f;

    UIController.instance.coinText.text = currentCoins.ToString();
}

referencia
public void GetCoins(int amount)
{
    currentCoins += amount;

    UIController.instance.coinText.text = currentCoins.ToString();
}

referencia
public void SpendCoins(int amount)
{
    currentCoins -= amount;

    if(currentCoins < 0)
    {
        currentCoins = 0;
    }

    UIController.instance.coinText.text = currentCoins.ToString();
}
}

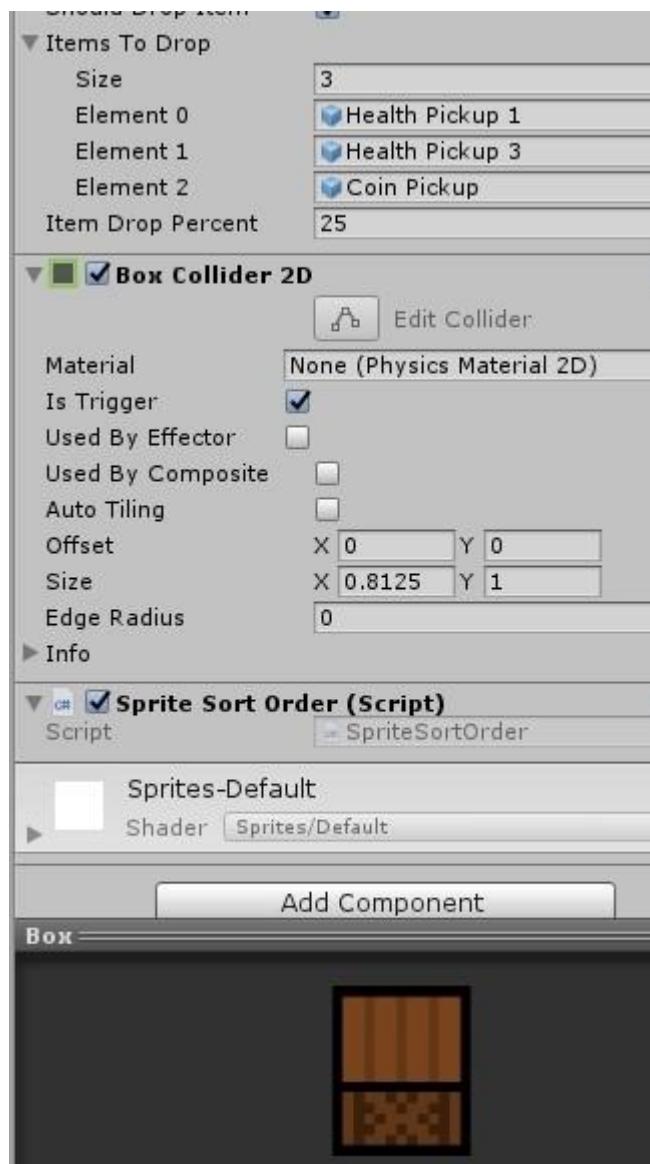
```

Ahora haremos que algunas cajas y enemigos tiren monedas

Agregamos la moneda a los pickups



Arrastramos la moneda a los items que dropea la caja



Ahora para que los enemigos tiren monedas al morir iremos al Script EnemyController y colocamos el siguiente código

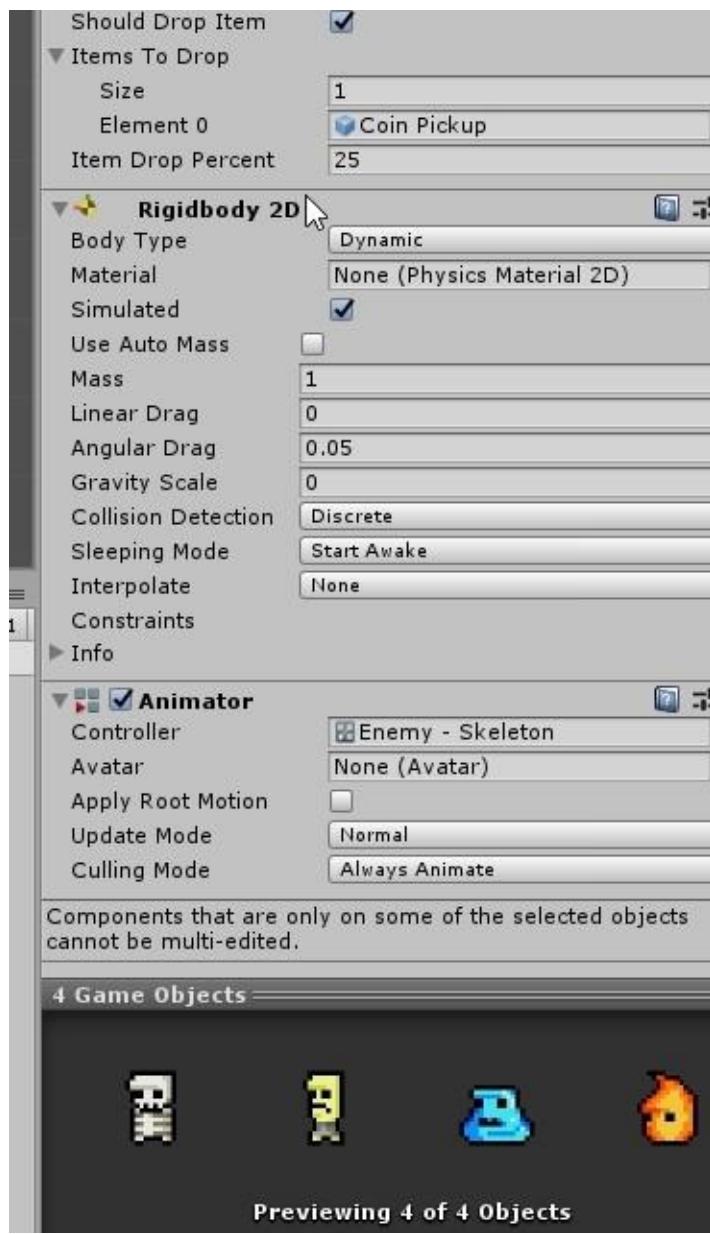
```
//variables para que los enemigos tiren items
public bool shouldDropItem;
public GameObject[] itemsToDrop;
public float itemDropPercent;
```

```
//Para que tiren monedas
if (shouldDropItem)
{
    float dropChance = Random.Range(0f, 100f);

    if (dropChance < itemDropPercent)
    {
        int randomItem = Random.Range(0, itemsToDrop.Length);

        Instantiate(itemsToDrop[randomItem], transform.position, transform.rotation);
    }
}
```

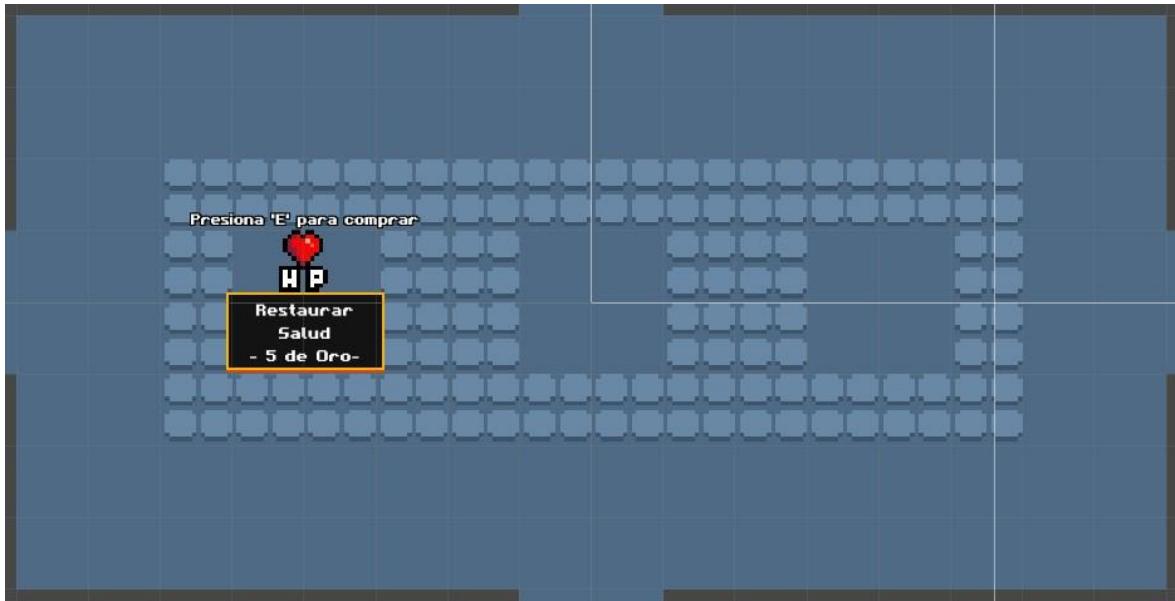
Ahora vamos al prefabs de los enemigos y arrastramos el pickup de la moneda, le establecemos un porcentaje de probabilidad de soltar el ítem.



Sección 16: Crear tienda

Ahora lo que haremos será crear una pequeña tienda, diseñamos primero la habitación.

Ahora diseñamos nuestro primer ítem de la tienda



Ahora para que sea posible su compra, crearemos el script ShopItem y arrastramos el script al al ítem.

Y en el script colocamos el siguiente código para que nos muestre el mensaje de compra

```
① Script de Unity | 0 referencias
② public class ShopItem : MonoBehaviour
{
    public GameObject buyMessage;

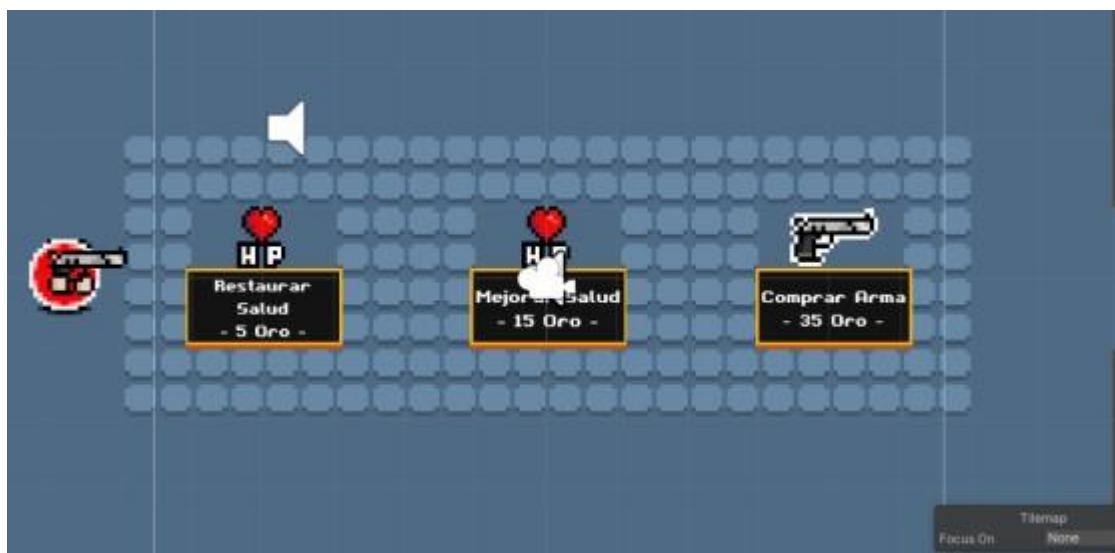
    //Aparezca el cuadro para comprar si esta cerca del item
    ③ Mensaje de Unity | 0 referencias
    private void OnTriggerEnter2D(Collider2D other)
    {
        if(other.tag == "Player")
        {
            buyMessage.SetActive(true);

            inBuyZone = true;
        }
    }

    //Desaparezca el cuadro de compra al estar fuera del área
    ④ Mensaje de Unity | 0 referencias
    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            buyMessage.SetActive(false);

            inBuyZone = false;
        }
    }
}
```

Ahora terminamos de diseñar nuestros demás items de la tienda



En el script ShopItem colocaremos el siguiente código, que ayudará a comprar el item y obtener su beneficio

```

using System.Collections;
using UnityEngine;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

public class ShopItem : MonoBehaviour
{
    public GameObject buyMessage;
    private bool inBuyZone; //Poder comprar si estamos en la zona
    public bool isHealthRestore, isHealthUpgrade, isWeapon; //Items de la tienda
    public int itemCost;//Costo del item
    public int healthUpgradeAmount;//Actualizar vida

    // Update is called once per frame
    void Update()
    {
        if(inBuyZone)//Condicional para saber si estamos en la zona de compra
        {
            if(Input.GetKeyDown(KeyCode.E))//Comprar con la tecla E
            {
                if(LevelManager.instance.currentCoins >= itemCost)
                {
                    LevelManager.instance.SpendCoins(itemCost);
                    //condicional si se compra restauración de vida
                    if(isHealthRestore)
                    {
                        PlayerHealthController.instance.HealPlayer(PlayerHealthController.instance.maxHealth);
                    }
                    //condicional si se compra actualización de vida
                    if(isHealthUpgrade)
                    {
                        PlayerHealthController.instance.IncreaseMaxHealth(healthUpgradeAmount);
                    }
                    //El item desaparecerá cuando se compre
                    gameObject.SetActive(false);
                    inBuyZone = false;

                    AudioManager.instance.PlaySFX(18);
                } else
                {
                    AudioManager.instance.PlaySFX(19);
                }
            }
        }
    }
}


```

Para aumentar la salud cuando se compre el ítem vamos al script PlayedHealthController y agregamos el siguiente código

```

//Aumentará la salud cuando se compre el ítem
public void IncreaseMaxHealth(int amount)
{
    maxHealth += amount;
    currentHealth = maxHealth;

    UIController.instance.healthSlider.MaxValue = maxHealth;
    UIController.instance.healthSlider.value = currentHealth;
    UIController.instance.healthText.text = currentHealth.ToString() + " / " + maxHealth.ToString();
}


```

Volvemos al script ShopItem y agregaremos el siguiente código para que cuando se compre un ítem este desaparezca

```

//El ítem desaparecerá cuando se compre
gameObject.SetActive(false);
inBuyZone = false;

```

Ahora introduciremos la tienda en los niveles

Vamos al script LevelGenerator y agregaremos lo siguiente para que se generé la tienda aleatoriamente.

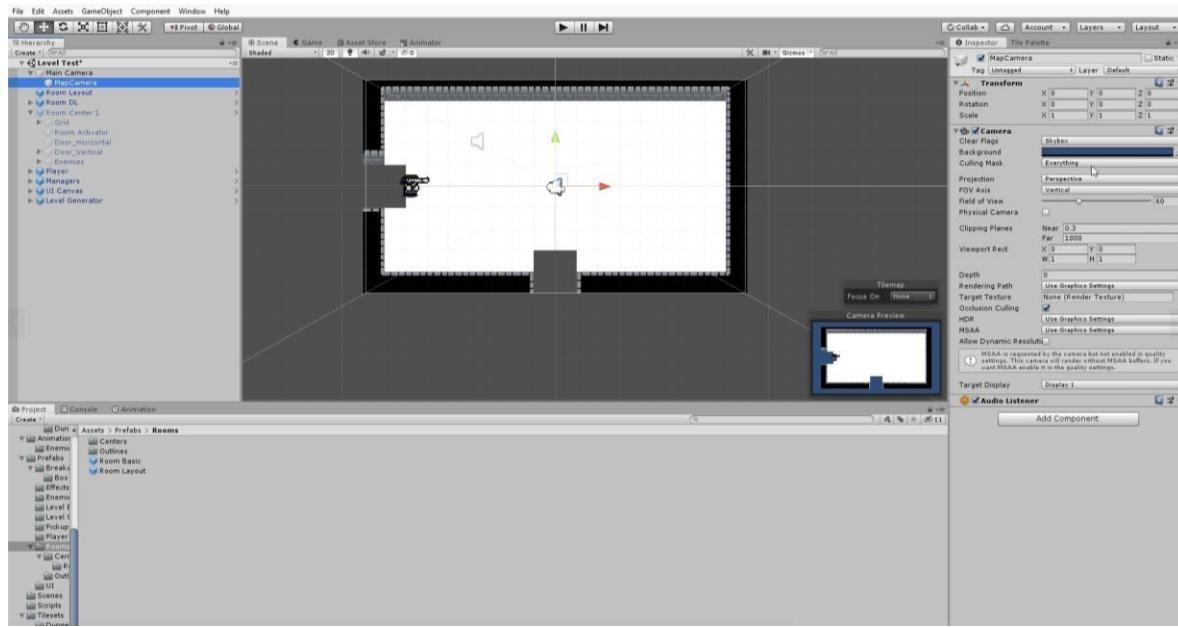
```
public bool includeShop;//incluir la tienda
public int minDistanceToShop, maxDistanceToShop;//distancia para comprar el arma

//condicional para incluir la tienda
if(includeShop)
{
    int shopSelector = Random.Range(minDistanceToShop, maxDistanceToShop + 1); //distancia de generación
    shopRoom = layoutRoomObjects[shopSelector];
    layoutRoomObjects.RemoveAt(shopSelector);
    shopRoom.GetComponent<SpriteRenderer>().color = shopColor;
}

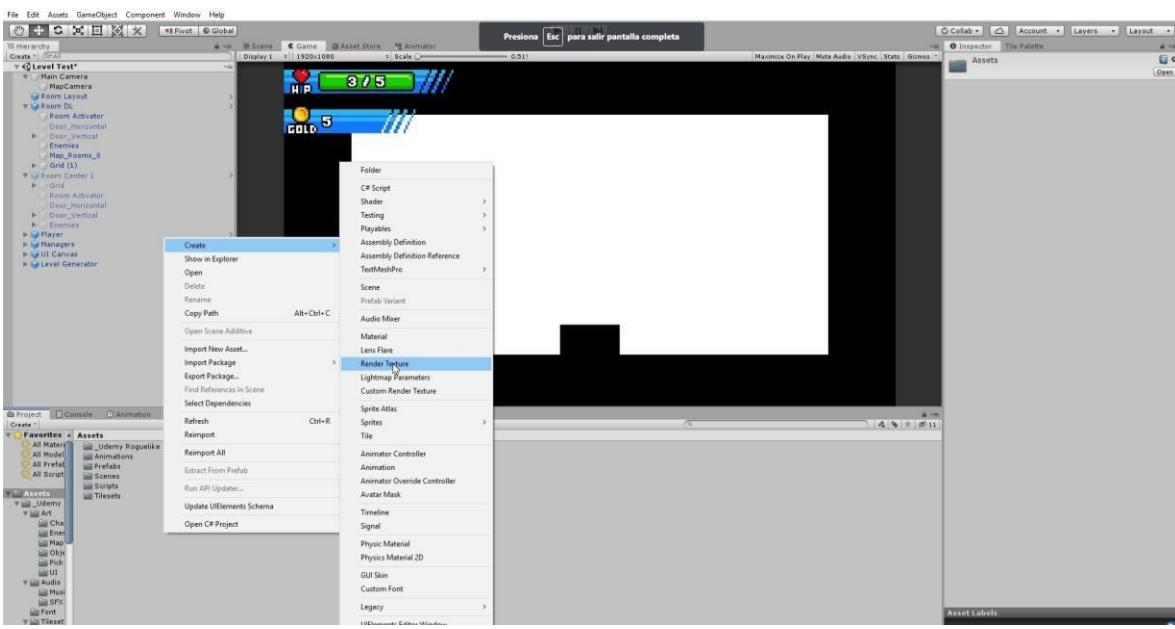
//crear entorno de la habitación
CreateRoomOutline(Vector3.zero);
foreach(GameObject room in layoutRoomObjects)
{
    CreateRoomOutline(room.transform.position);
}
CreateRoomOutline(endRoom.transform.position);
if(includeShop)
{
    CreateRoomOutline(shopRoom.transform.position);
}
```

Sección 17: Crear minimapa

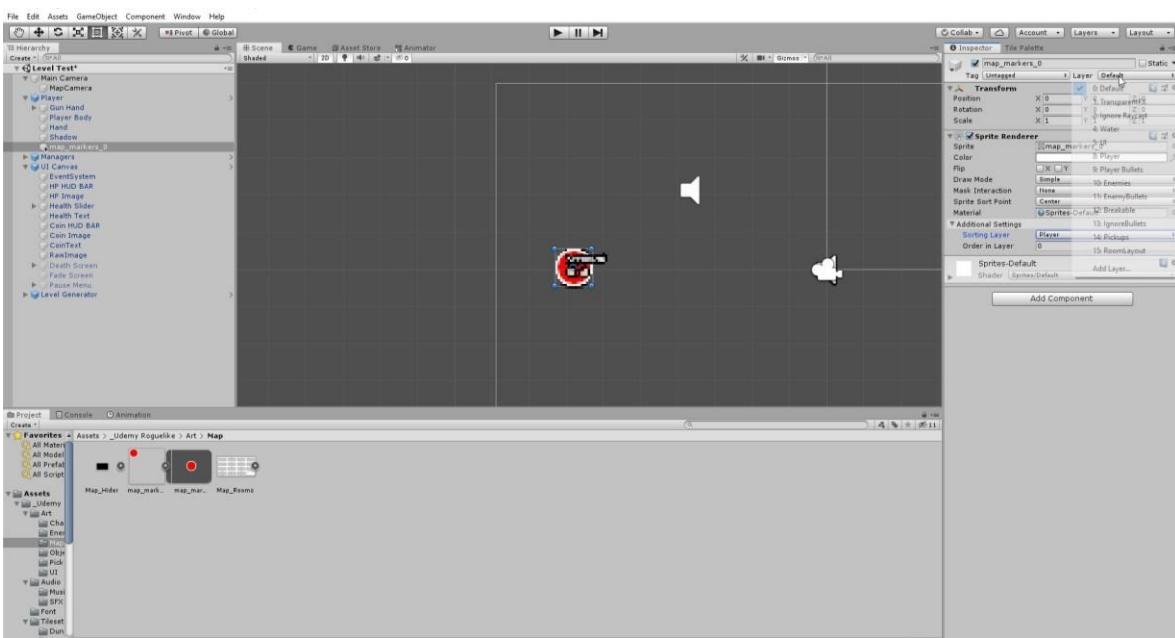
Creamos una nueva cámara dentro de nuestra cámara principal el cual servirá para nuestro minimapa.



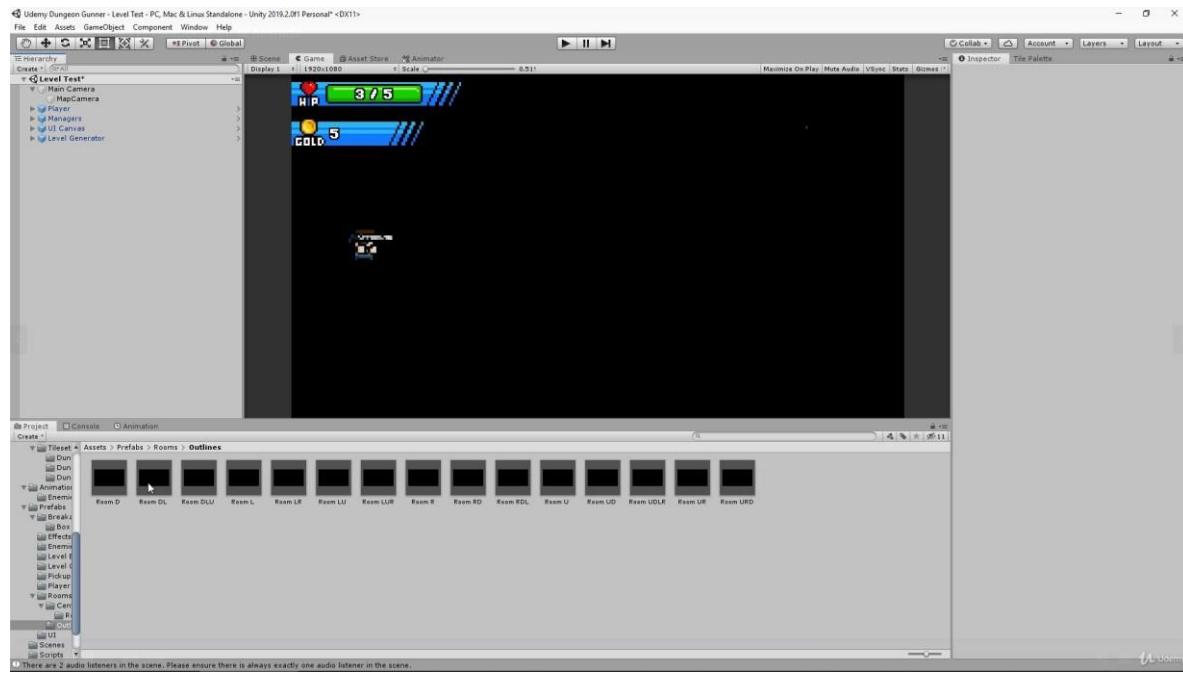
Creamos un objeto render texture y lo instanciamos con nuestra nueva cámara.



Agregamos un sprite nuevo que será representante de nuestro player pero como un punto dentro del minimapa.



Y para que no se muestre igual que nuestro mapa normal, ocultamos todas las habitaciones.

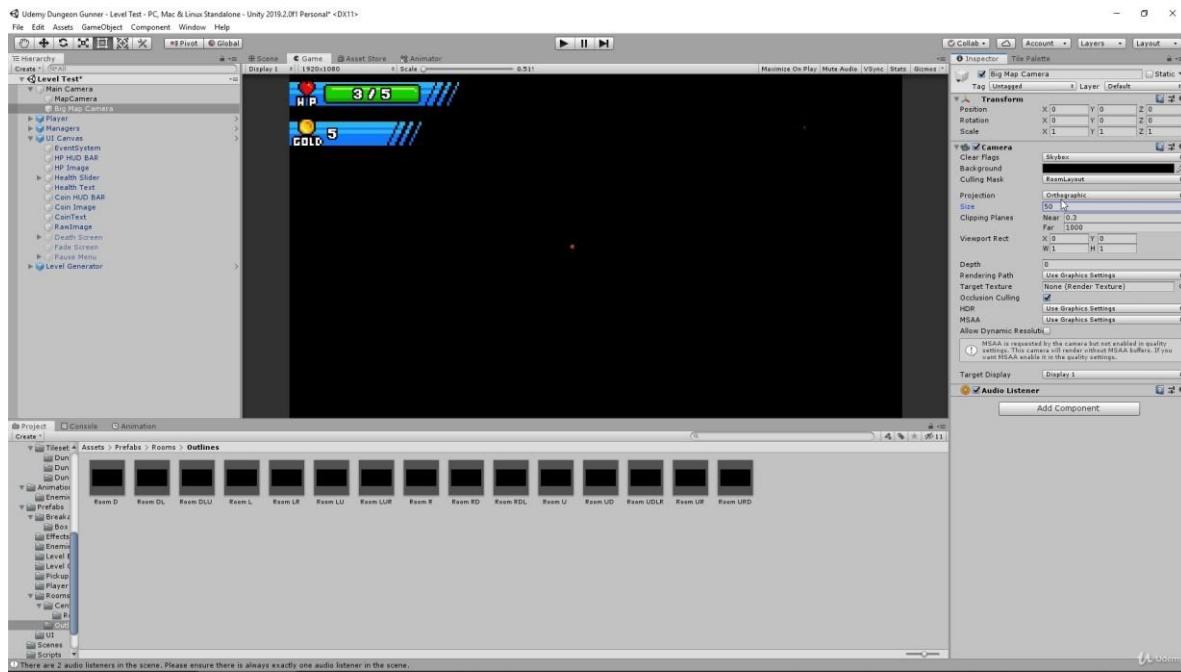


Asignamos la tecla con la cual visualizamos el mapa completo.

```
// Update is called once per frame
@ Mensaje de Unity | 0 referencias
void Update()
{
    //controlar el enfoque de camara para que sea estatico por habitaciones
    if (target != null)
    {
        transform.position = Vector3.MoveTowards(transform.position, new Vector3(target
    }

    if(Input.GetKeyDown(KeyCode.M) && !isBossRoom)
    {
        if(!bigMapActive)
        {
            ActivateBigMap();
        } else
        {
            DeactivateBigMap();
        }
    }
}
```

Agregamos otra cámara que será el mapa pero en pantalla completa.



En el script indicamos que cuando presionemos la tecla indicada mostremos el mapa completo y de la misma forma ocultamos el mapa completo.

```
//activamos mapa grande
1 referencia
public void ActivateBigMap()
{
    if (!LevelManager.instance.isPaused)
    {

        bigMapActive = true;

        bigMapCamera.enabled = true;
        mainCamera.enabled = false;

        PlayerController.instance.canMove = false;

        Time.timeScale = 0f;

        UIController.instance.mapDisplay.SetActive(false);
        UIController.instance.bigMapText.SetActive(true);
    }
}

//desactivamos mapa grande
1 referencia
public void DeactivateBigMap()
{
    if (!LevelManager.instance.isPaused)
    {
        bigMapActive = false;

        bigMapCamera.enabled = false;
        mainCamera.enabled = true;

        PlayerController.instance.canMove = true;

        Time.timeScale = 1f;

        UIController.instance.mapDisplay.SetActive(true);
        UIController.instance.bigMapText.SetActive(false);
    }
}
```

Sección 18: Crear tipo de armas

Creamos un nuevo Script llamado Gun y agregamos lo siguiente

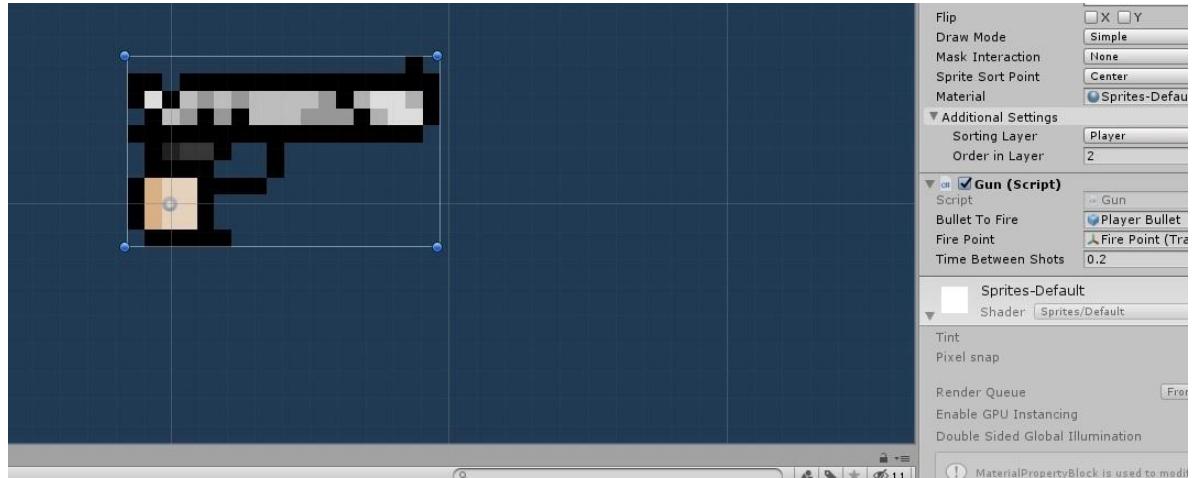
```
Script de Unity | 9 referencias
public class Gun : MonoBehaviour
{
    public GameObject bulletToFire;
    public Transform firePoint; //distancia

    public float timeBetweenShots; //tiempo entre tiros
    private float shotCounter;

    void Update()
    {
        if (PlayerController.instance.canMove && !LevelManager.instance.isPaused)
            //Condicional para que el jugador pueda disparar mientras puede moverse y el juego no este en pausa
        {
            if (shotCounter > 0)
            {
                shotCounter -= Time.deltaTime;
            }
            else
            {

                if (Input.GetMouseButtonDown(0) || Input.GetMouseButton(0)) //Disparar con click o si se mantiene presionado
                {
                    Instantiate(bulletToFire, firePoint.position, firePoint.rotation);
                    shotCounter = timeBetweenShots;
                    AudioManager.instance.PlaySFX(12);
                }
            }
        }
    }
}
```

Entonces seleccionamos el arma y arrastramos el script y le asignamos los datos, como la velocidad entre disparos.



Ahora agregaremos diferentes armas, cambiando los sprites y cambiamos su velocidad entre disparos



Para poder cambiar entre armas, iremos al script PlayerController y agregamos una lista de armas

```
public List<Gun> availableGuns = new List<Gun>(); //Lista de armas
public int currentGun;
```

Arrastramos un arma al personaje



En el mismo script seguimos añadiendo más código, el siguiente para poder cambiar entre armas

```

if(Input.GetKeyDown(KeyCode.Tab))//condicional para cambiar el arma
{
    if(availableGuns.Count > 0)//si tenemos mas de 1 arma
    {
        currentGun++;
        if(currentGun >= availableGuns.Count)
        {
            currentGun = 0;
        }

        SwitchGun();

    } else
    {
        Debug.LogError("El jugado no tiene armas!");
    }
}

referencias
public void SwitchGun()
{
    foreach(Gun theGun in availableGuns)
    {
        theGun.gameObject.SetActive(false);
    }

    availableGuns[currentGun].gameObject.SetActive(true);

    UIController.instance.currentGun.sprite = availableGuns[currentGun].gunUI;
    UIController.instance.gunText.text = availableGuns[currentGun].weaponName;

    AudioManager.instance.PlaySFX(6);
}

```

Ahora añadiremos a la interfaz el modelo del arma y nombre En

el script GUN añadimos

```

public string weaponName;
public Sprite gunUI;

```

A cada arma le ponemos su nombre



En el script de UIController agregamos

```
public Image currentGun;
public Text gunText;
```

En el script PlayerController agregamos

```
void Start()
{
    //theCam = Camera.main;

    activeMoveSpeed = moveSpeed;
    //Cambiara los modelos de las armas en la interfaz
    UIController.instance.currentGun.sprite = availableGuns[currentGun].gunUI;
    UIController.instance.gunText.text = availableGuns[currentGun].weaponName;
}
```

Para recoger las armas crearemos el script GunPikcup y agregamos el siguiente código

```

① Script de Unity | 1 referencia
② Public class GunPickup : MonoBehaviour
{
    public Gun theGun;

    public float waitToBeCollected = .5f;

    // Start is called before the first frame update
    ③ Mensaje de Unity | 0 referencias
    void Start()
    {
    }

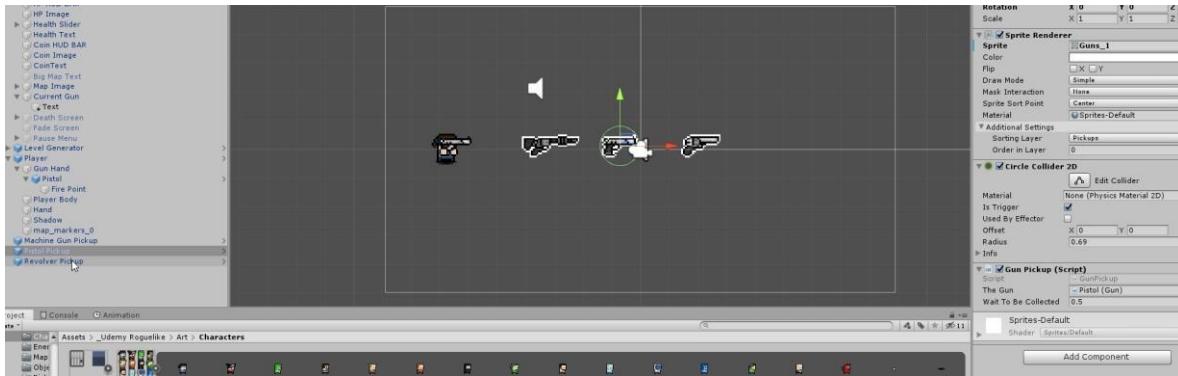
    // Update is called once per frame
    ④ Mensaje de Unity | 0 referencias
    void Update()
    {
        if (waitToBeCollected > 0)
        {
            waitToBeCollected -= Time.deltaTime;
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player" && waitToBeCollected <= 0)
        {
            bool hasGun = false; //comprobar que no recogeremos la misma arma
            foreach(Gun gunToCheck in PlayerController.instance.availableGuns)
            { //No agregará el arma si ya la tenemos
                if(theGun.weaponName == gunToCheck.weaponName)
                {
                    hasGun = true;
                }
            }
            //Si no tenemos el arma la cogeremos
            if(!hasGun)
            {
                //Clonar el arma y sus posiciones
                Gun gunClone = Instantiate(theGun);
                gunClone.transform.parent = PlayerController.instance.gunArm;
                gunClone.transform.position = PlayerController.instance.gunArm.position;
                gunClone.transform.localRotation = Quaternion.Euler(Vector3.zero);
                gunClone.transform.localScale = Vector3.one;

                PlayerController.instance.availableGuns.Add(gunClone);
                PlayerController.instance.currentGun = PlayerController.instance.availableGuns.Count - 1;
                PlayerController.instance.SwitchGun();
            }
        }
    }
}

```

Ahora creamos los prefabs para cada arma



Creamos un área especial para la generación del cofre



En el scrip LevelGenerator agregamos el siguiente código

```
public bool includeGunRoom;
public int minDistanceToGunRoom, maxDistanceToGunRoom; //Distancia en la que aparecerá el cuarto del cofre

//condicional para incluir el cuarto del cofre
if (includeGunRoom)
{
    int grSelector = Random.Range(minDistanceToGunRoom, maxDistanceToGunRoom + 1);
    gunRoom = layoutRoomObjects[grSelector];
    layoutRoomObjects.RemoveAt(grSelector);
    gunRoom.GetComponent<SpriteRenderer>().color = gunRoomColor;
}

}
}

if(includeGunRoom)
{
    if (outline.transform.position == gunRoom.transform.position)
    {
        Instantiate(centerGunRoom, outline.transform.position, transform.rotation).theRoom = outline.GetComponent<Room>();
        generateCenter = false;
    }
}
```

Crearemos ahora un cofre donde podamos recolectar armas.

Creamos un script llamado GunChest con el siguiente código

```
④ Script de Unity | 0 referencias
public class GunChest : MonoBehaviour
{
    public GunPickup[] potentialGuns;

    public SpriteRenderer theSR;
    public Sprite chestOpen;

    public GameObject notification;

    private bool canOpen, isOpen;

    public Transform spawnPoint;

    public float scaleSpeed = 2f;
```

```

// Mensaje de Unity | 0 referencias
void Update()
{
    if(canOpen && !isOpen)
    {//Abrir cofre
        if(Input.GetKeyDown(KeyCode.E))
        {
            int gunSelect = Random.Range(0, potentialGuns.Length);

            Instantiate(potentialGuns[gunSelect], spawnPoint.position, spawnPoint.rotation);

            theSR.sprite = chestOpen;

            isOpen = true;

            transform.localScale = new Vector3(1.2f, 1.2f, 1.2f);
        }
    }
    //pequeña animación al abrir el cofre
    if(isOpen)
    {
        transform.localScale = Vector3.MoveTowards(transform.localScale, Vector3.one, Time.deltaTime * scaleSpeed);
    }
}

```

//Metodos para activa el cofre si esta en el area

④ Mensaje de Unity | 0 referencias

private void OnTriggerEnter2D(Collider2D other)

```

{
    if(other.tag == "Player")
    {
        notification.SetActive(true);

        canOpen = true;
    }
}

```

④ Mensaje de Unity | 0 referencias

private void OnTriggerExit2D(Collider2D other)

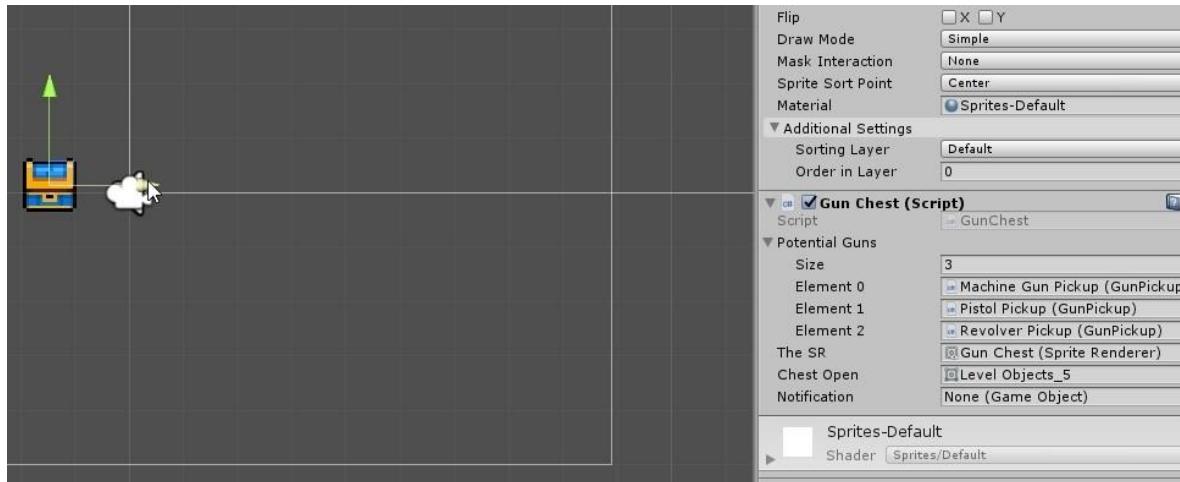
```

{
    if (other.tag == "Player")
    {
        notification.SetActive(false);

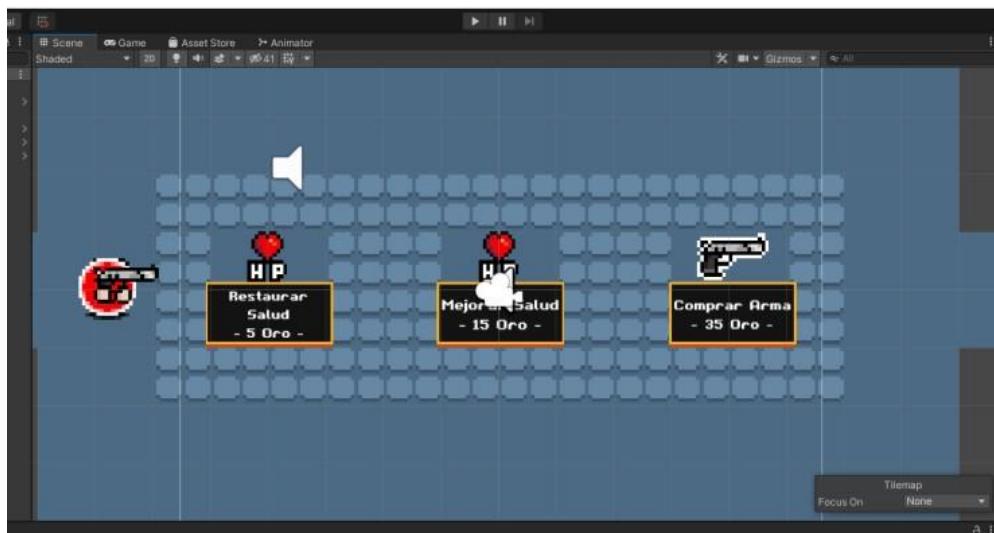
        canOpen = false;
    }
}

```

Ahora agregamos a nuestro cofre las armas que puede tener



Con las armas listas, podemos ahora agregarla a la tienda



En el script ShopItem agregamos

```

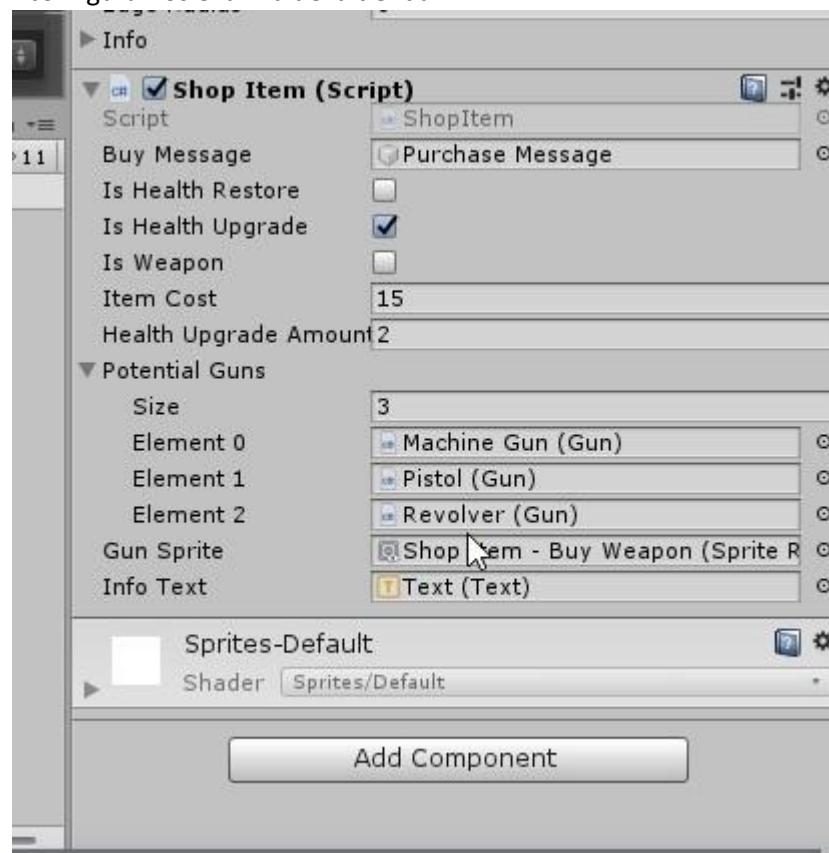
public Gun[] potentialGuns; // matriz de armas
private Gun theGun;
public SpriteRenderer gunSprite;
public Text infoText;

void Start()
{//Actualizará la tienda dependiendo el arma que toque
if(isWeapon)
{
    int selectedGun = Random.Range(0, potentialGuns.Length);
    theGun = potentialGuns[selectedGun];

    gunSprite.sprite = theGun.gunShopSprite;
    infoText.text = theGun.weaponName + "\n - " + theGun.itemCost + " Oro - ";
    itemCost = theGun.itemCost;
}
}

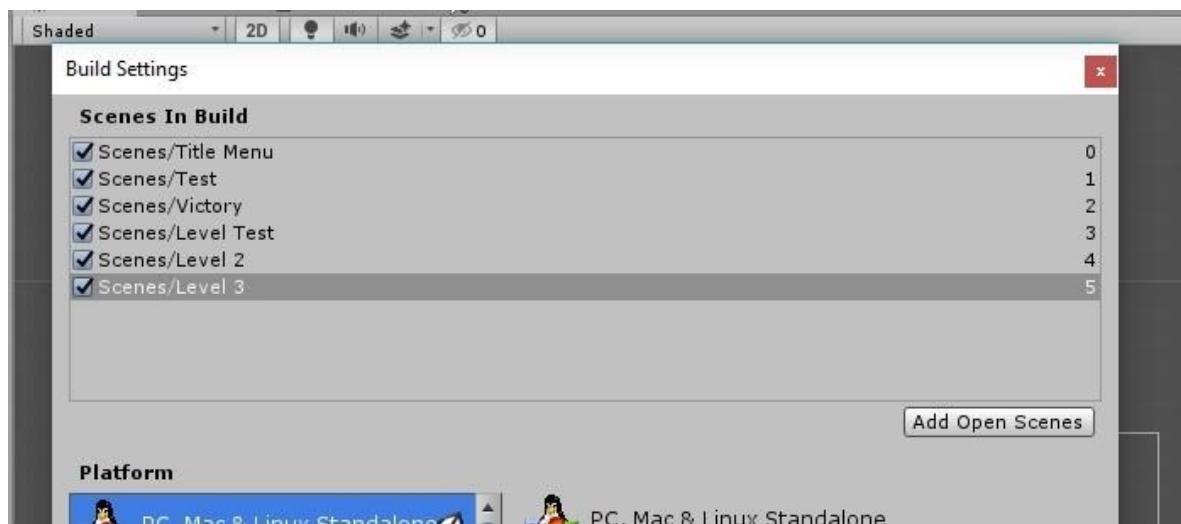
```

Y configuramos el arma de la tienda



Sección 19: Seguimiento de nivel

Una vez listas nuestras escenas, pondremos en orden los niveles, donde tendremos 3 niveles entonces haremos un build setting a nuestros niveles



Ahora haremos un seguimiento de las estadísticas entre los niveles para que no se pierdan la salud, las monedas, ni las armas

Creamos el script CharacterTracker y agregamos las siguientes variables

```
public class CharacterTracker : MonoBehaviour
{
    public static CharacterTracker instance;
    //variables que se rastrearan
    public int currentHealth, maxHealth, currentCoins;

    @ Mensaje de Unity | 0 referencias
    private void Awake()
    {
        instance = this;
    }
}
```

Agregaremos los siguientes códigos para los siguientes Script para almacenar las estadísticas cuando terminamos el nivel

PlayerHealteController

```
@ Mensaje de Unity | 0 referencias
void Start()
{
    maxHealth = CharacterTracker.instance.maxHealth;
    currentHealth = CharacterTracker.instance.currentHealth;

    //currentHealth = maxHealth;

    UIController.instance.healthSlider.MaxValue = maxHealth;
    UIController.instance.healthSlider.value = currentHealth;
    UIController.instance.healthText.text = currentHealth.ToString() + " / " + maxHealth.ToString();
}

// Update is called once per frame.
```

LevelManager

```
void Start()
{
    PlayerController.instance.transform.position = startPoint.position
    PlayerController.instance.canMove = true;

    currentCoins = CharacterTracker.instance.currentCoins;

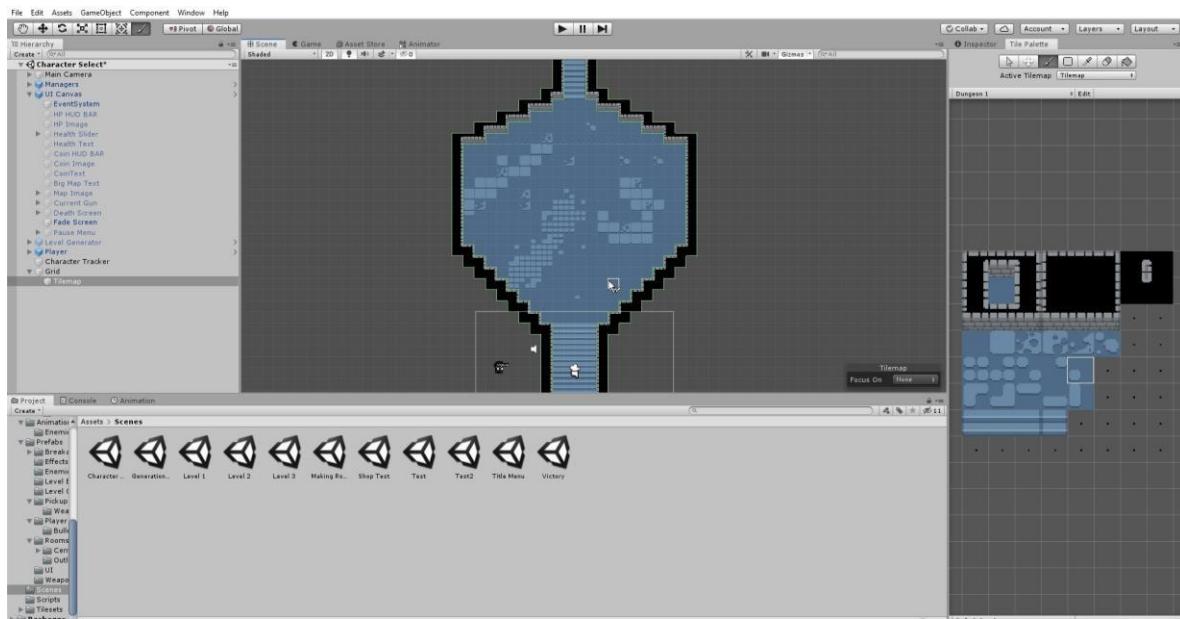
    Time.timeScale = 1f;

    UIController.instance.coinText.text = currentCoins.ToString();
}

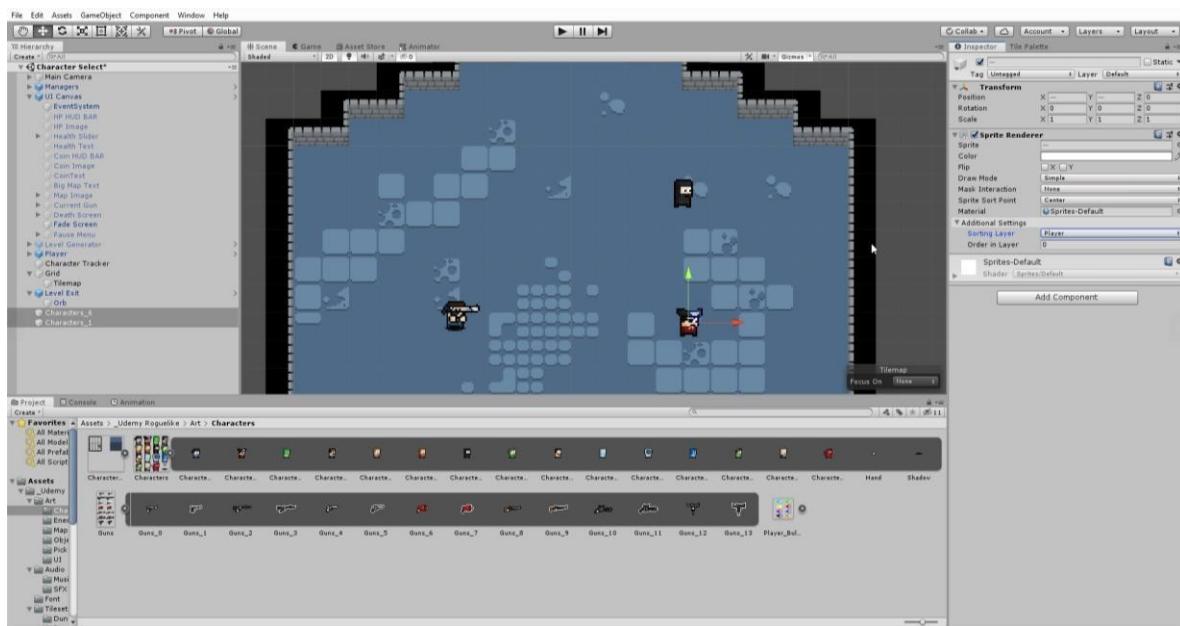
referencia
public IEnumerator LevelEnd()
{
    //Se guardan las variables al finalizar la escena
    CharacterTracker.instance.currentCoins = currentCoins;
    CharacterTracker.instance.currentHealth = PlayerHealthController.instance.currentHealth;
    CharacterTracker.instance.maxHealth = PlayerHealthController.instance.maxHealth;
```

Sección 20: Múltiples personajes

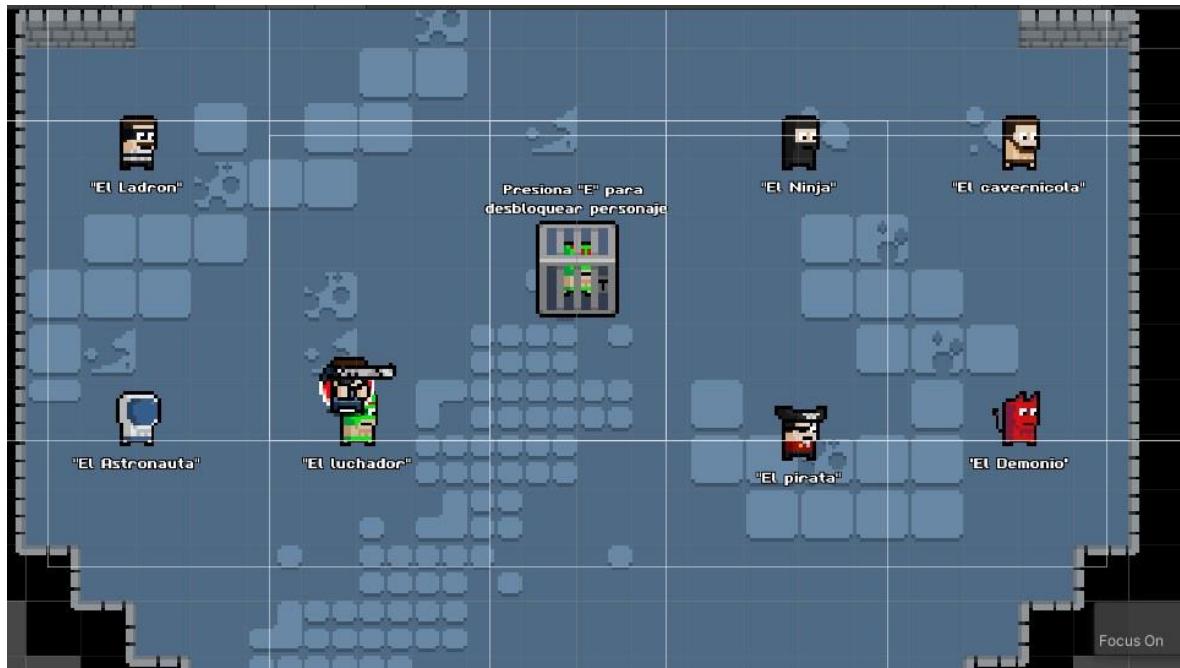
Con nuestra herramienta de Tile Palette diseñamos una nueva sala, en el cual se podrán seleccionar distintos personajes.



Creamos copias de nuestro player principal y le cambio el sprite y configuramos su arma para cada uno.



Agregamos varios personajes.



En el script asignamos para que el nuevo personaje seleccionado obtenga la posición, enfoque y control de nuestro personaje principal, y a la vez este se destruya. Pero creando una copia en otra posición, a la misma vez cada vez que un personaje se seleccione el actual ocupará su posición inicial.

```

if(canSelect)
{
    //Asignamos la tecla e para cambio de personaje
    if(Input.GetKeyDown(KeyCode.E))
    {
        //obtiene la posicion donde se encuentra el personaje actual
        Vector3 playerPos = PlayerController.instance.transform.position;

        //personaje actual se destruye
        Destroy(PlayerController.instance.gameObject);

        //el control del personaje apunta al nuevo personaje
        PlayerController newPlayer = Instantiate(playerToSpawn, playerPos, playerToSpawn.transform.rotation);
        PlayerController.instance = newPlayer;

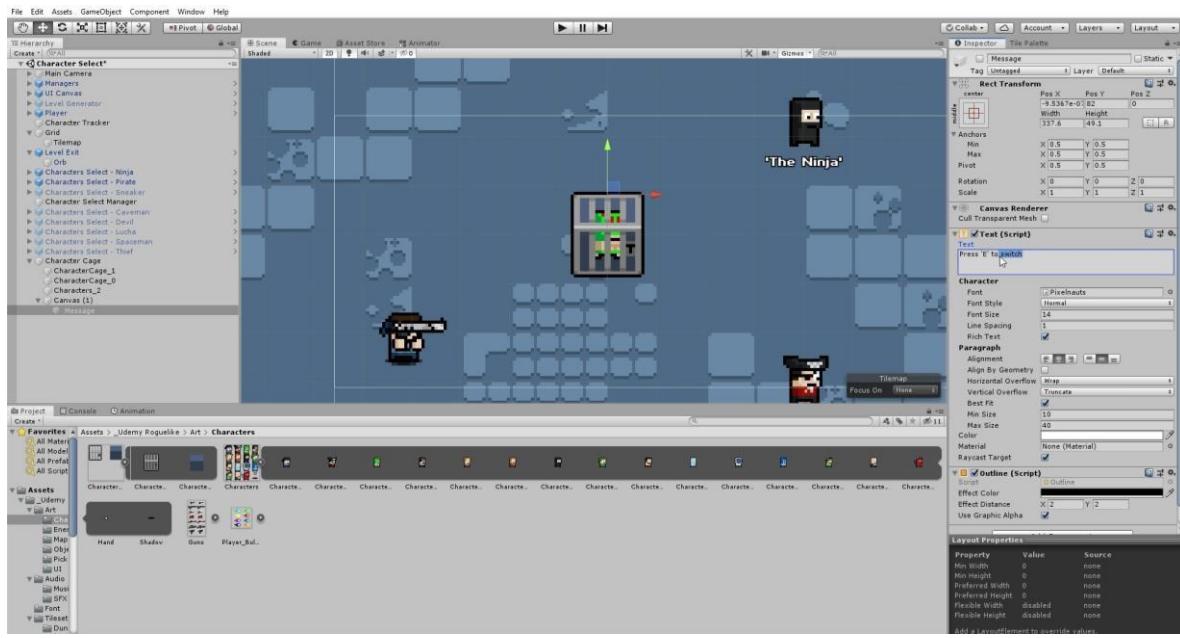
        gameObject.SetActive(false);

        //nuevo personaje se transforma en el player y la camara a ese nuevo objeto
        CameraController.instance.target = newPlayer.transform;

        //nuevo personaje principal obtiene nueva posicion
        CharacterSelectManager.instance.activePlayer = newPlayer;
        //cada vez que cambiamos otro obtendra su posicion inicial
        CharacterSelectManager.instance.activeCharSelect.gameObject.SetActive(true);
        CharacterSelectManager.instance.activeCharSelect = this;
    }
}

```

Ahora agregamos otro prefab para que se pueda desbloquear en el transcurso de cada nivel.



En el script definimos que este se active sólo si interactúa con el player y muestra el mensaje.

```
④ Mensaje de Unity | 0 referencias
private void OnTriggerEnter2D(Collider2D other)
{
    //condicional para saber si es player
    if(other.tag == "Player")
    {
        //se desbloquea
        canUnlock = true;
        //muestra mensaje
        message.SetActive(true);
    }
}

④ Mensaje de Unity | 0 referencias
private void OnTriggerExit2D(Collider2D other)
{
    //condicional para saber si es player
    if (other.tag == "Player")
    {
        canUnlock = false;
        //muestra mensaje
        message.SetActive(false);
    }
}
```

Para que el personaje a desbloquear se genere aleatoriamente creamos el siguiente script y le asignamos la tecla E para que se desbloquee.

```

//array para asignar personaje aleatorio
public CharacterSelector[] charSelects;
private CharacterSelector playerToUnlock;

public SpriteRenderer cagedSR;

// Start is called before the first frame update
@ Mensaje de Unity | 0 referencias
void Start()
{
    //agregar un personaje aleatorio a la jaula
    playerToUnlock = charSelects[Random.Range(0, charSelects.Length)];
    //establecemos su sprite
    cagedSR.sprite = playerToUnlock.playerToSpawn.bodySR.sprite;
}

// Update is called once per frame
@ Mensaje de Unity | 0 referencias
void Update()
{
    if(canUnlock)
    {
        //asignamos tecla para desbloquear
        if (Input.GetKeyDown(KeyCode.E))
        {
            //instanciamos posicion de personaje actual
            PlayerPrefs.SetInt(playerToUnlock.playerToSpawn.name, 1);

            Instantiate(playerToUnlock, transform.position, transform.rotation);

            gameObject.SetActive(false);
        }
    }
}

```

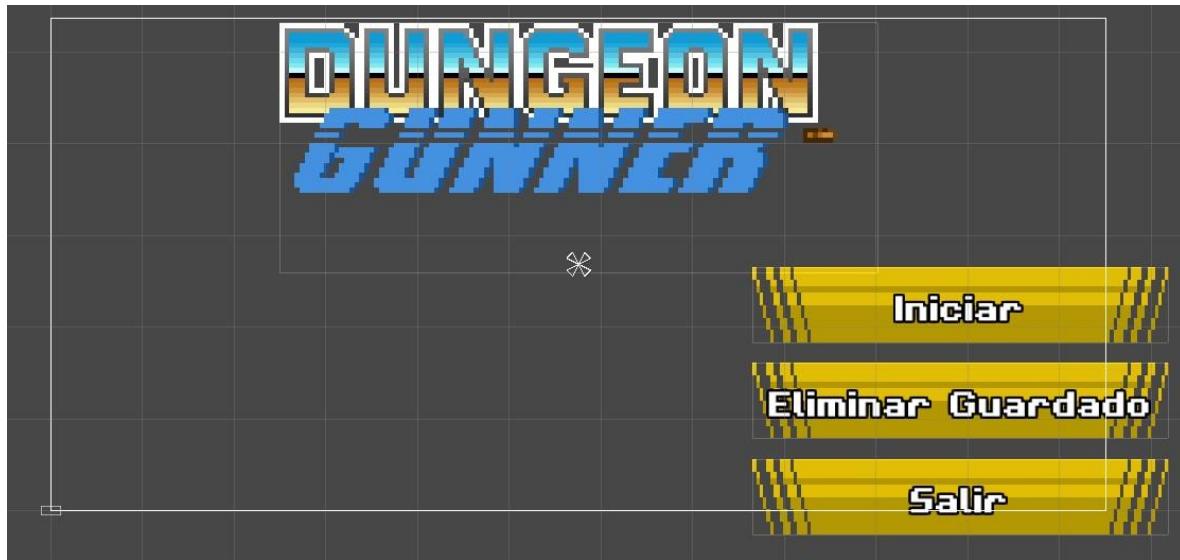
Creamos un script para guardar todos nuestros personajes desbloqueados dentro de un array.

```

void Start()
{
    //para guardar personajes y solo mostrar si se encuentran desbloqueados
    if (shouldUnlock)
    {
        if (PlayerPrefs.HasKey(playerToSpawn.name))
        {
            if (PlayerPrefs.GetInt(playerToSpawn.name) == 1)
            {
                gameObject.SetActive(true);
            }
            else
            {
                gameObject.SetActive(false);
            }
        }
        else
        {
            gameObject.SetActive(false);
        }
    }
}

```

Y ahora creamos un botón para que podamos eliminar los personajes desbloqueados.



En el script definimos que recorra nuestro array de personajes desbloqueados y los elimine uno a uno.

```
//boton eliminar guardados
0 referencias
public void DeleteSave()
{
    deletePanel.SetActive(true);
}

//boton confirmar eliminar guardados
0 referencias
public void ConfirmDelete()
{
    deletePanel.SetActive(false);
    //elimina cada uno de estos personajes
    foreach(CharacterSelector theChar in charactersToDelete)
    {
        PlayerPrefs.SetInt(theChar.playerToSpawn.name, 0);
    }
}
//cancelar eliminacion
0 referencias
public void CancelDelete()
{
    deletePanel.SetActive(false);
}
```

Validamos para que cada vez que iniciemos nueva partida no se generen más players.



En el script definimos que al presionar uno de esos botones el player actual se destruya.

```
//Crear nuevo juego
0 referencias
public void NewGame()
{
    Time.timeScale = 1f;

    SceneManager.LoadScene(newGameScene);
    //destruir personaje actual
    Destroy(PlayerController.instance.gameObject);
}

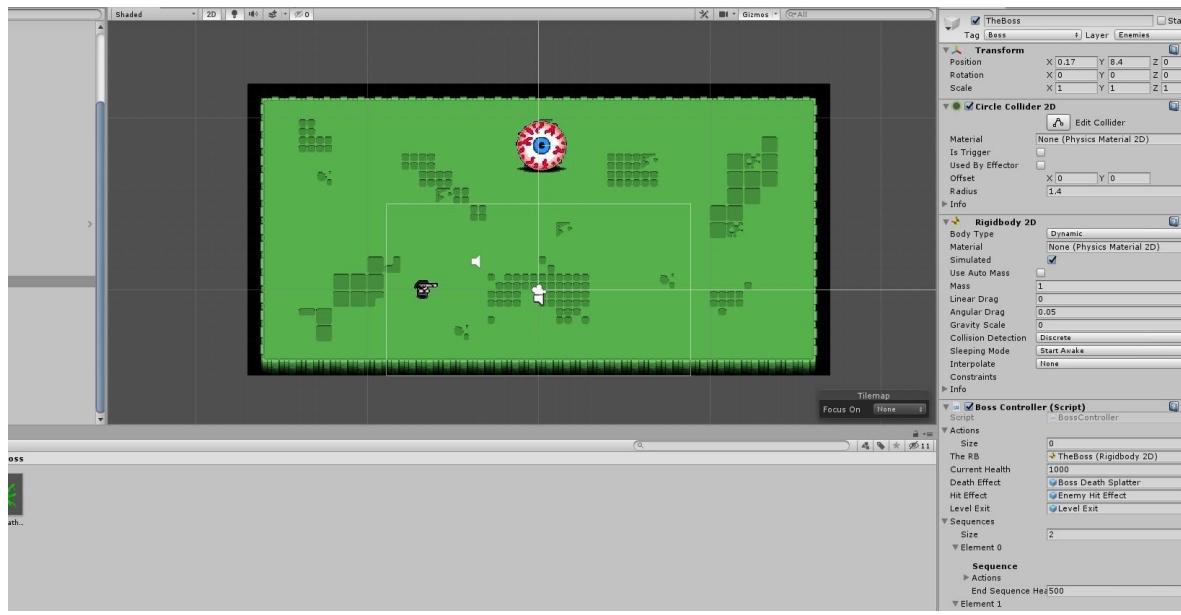
//Regresar al menu
0 referencias
public void ReturnToMainMenu()
{
    Time.timeScale = 1f;

    SceneManager.LoadScene(mainMenuScene);
    //destruir personaje actual
    Destroy(PlayerController.instance.gameObject);
}

//continuar con la partida
0 referencias
public void Resume()
{
    LevelManager.instance.PauseUnpause();
}
```

Sección 21: Jefe Final

Para el jefe final diseñaremos primero su habitación.



Luego creamos el script BossController y ponemos el siguiente código para controlar su movimiento, disparo y salud.

```
① Script de Unity | 4 referencias
public class BossController : MonoBehaviour
{
    public static BossController instance;

    public BossAction[] actions;
    private int currentAction;
    private float actionCounter;

    private float shotCounter;
    private Vector2 moveDirection;
    public Rigidbody2D theRB;

    public int currentHealth;

    public GameObject deathEffect, hitEffect;
    public GameObject levelExit;

    public BossSequence[] sequences;
    public int currentSequence;

    ② Mensaje de Unity | 0 referencias
    private void Awake()
    {
        instance = this;
    }

    // Start is called before the first frame update
    ③ Mensaje de Unity | 0 referencias
    void Start()
    {
        // Colocar la barra de vida del jefe
        actions = sequences[currentSequence].actions;

        actionCounter = actions[currentAction].actionLength;

        UIController.instance.bossHealthBar.MaxValue = currentHealth;
        UIController.instance.bossHealthBar.value = currentHealth;
    }
}
```

```
// Update is called once per frame
// Mensaje de Unity | 0 referencias
void Update()
{
    if(actionCounter > 0)
    {
        actionCounter -= Time.deltaTime;

        //movimiento
        moveDirection = Vector2.zero;

        if(actions[currentAction].shouldMove)
        {
            if(actions[currentAction].shouldChasePlayer)
            {
                moveDirection = PlayerController.instance.transform.position - transform.position;
                moveDirection.Normalize();
            }

            if(actions[currentAction].moveToPoint && Vector3.Distance(transform.position, actions[currentAction].pointToMoveTo.position) > .5f)
            {
                moveDirection = actions[currentAction].pointToMoveTo.position - transform.position;
                moveDirection.Normalize();
            }
        }

        theRB.velocity = moveDirection * actions[currentAction].moveSpeed;
    }

    //disparo
    if(actions[currentAction].shouldShoot)
    {
        shotCounter -= Time.deltaTime;
        if(shotCounter <= 0)
        {
            shotCounter = actions[currentAction].timeBetweenShots;

            foreach(Transform t in actions[currentAction].shotPoints)
            {
                Instantiate(actions[currentAction].itemToShoot, t.position, t.rotation);
            }
        }
    }

    } else
    {
        currentAction++;
        if(currentAction >= actions.Length)
        {
            currentAction = 0;
        }

        actionCounter = actions[currentAction].actionLength;
    }
}
```

```
}

//salud del jefe y animación de daño
referencia
public void TakeDamage(int damageAmount)
{
    currentHealth -= damageAmount;

    if (currentHealth <= 0)
    {
        gameObject.SetActive(false);

        Instantiate(deathEffect, transform.position, transform.rotation);

        if (Vector3.Distance(PlayerController.instance.transform.position, levelExit.transform.position) < 2f)
        {
            levelExit.transform.position += new Vector3(4f, 0f, 0f);
        }

        levelExit.SetActive(true);

        UIController.instance.bossHealthBar.gameObject.SetActive(false);
    }
    else
    {
        if(currentHealth <= sequences[currentSequence].endSequenceHealth && currentSequence < sequences.Length - 1)
        {
            currentSequence++;
            actions = sequences[currentSequence].actions;
            currentAction = 0;
            actionCounter = actions[currentAction].actionLength;
        }
    }
}

UIController.instance.bossHealthBar.value = currentHealth;
}
```

IV. RECOMENDACIONES

- Puede recurrir a assets ya creados por la comunidad ya que es el camino más rápido, pero si se desea crear algo totalmente personalizado puede hacerlo con pixilArt (básico) o Blender (avanzado).
- Si tiene algunas dudas sobre el uso de las herramientas de Unity puede acceder a la documentación <https://docs.unity3d.com/Manual/index.html>

V. CONCLUSIONES

- Se concluyó con éxito la creación del videojuego “Dungeon Gunner”
- Se aplicó los conocimientos adquiridos en el curso