Universidad Nacional de San Agustin
Facultad de Ingenieria, Produccion y Servicios
Escuela Profesional de Ciencia de la Computacion

Sistemas de partículas

Integrantes: Kelvin Paul Pucho Zevallos

Profesor: Diego Alonso Iquira Becerra

Fecha de realización: 8 de junio de 2022
Fecha de entrega: 8 de junio de 2022

Arequipa - Perú

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Creación de Objetos

La escena esta constituida por los siguientes objetos:

- **Camera**: Objeto para representar la cámara principal
- **Floor**: Objeto para representar el suelo
- **Player**: Objeto de tipo capsula, este objeto es la base para complementar con el arma y el movimiento,
- **Muro** Estructura con la que colisionara las balas y se destruirá.
- **Canvas** Contiene un texto que representa la resistencia de la pared

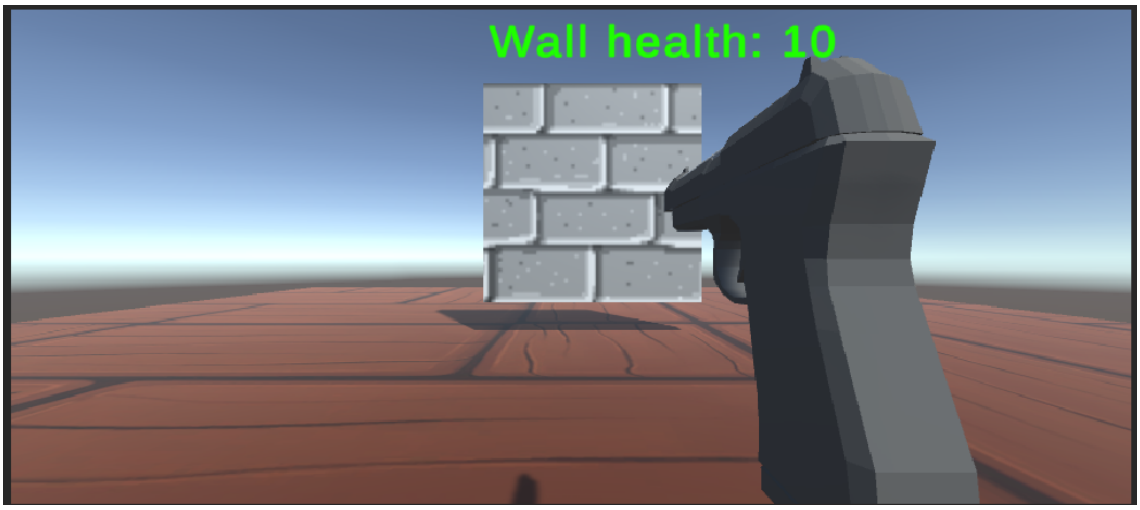

Figura 1: Escenario

Vista de Juego La cámara enfoca entorno al jugador.

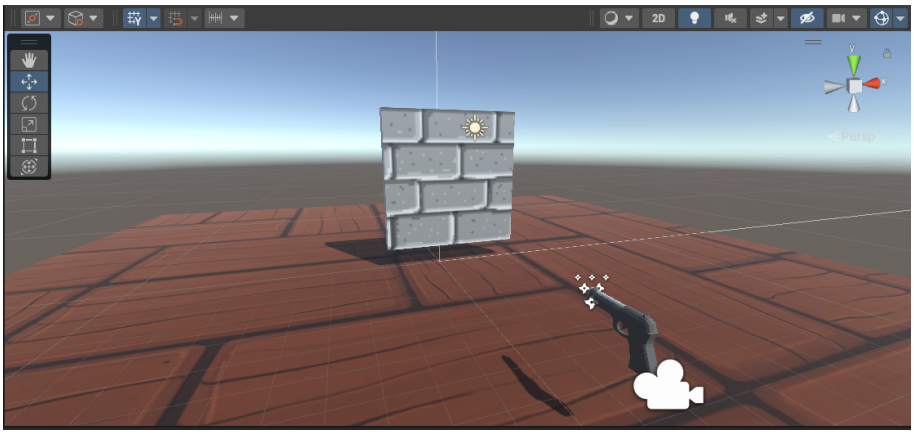

Figura 2: Vista del Juego

Hierarchy

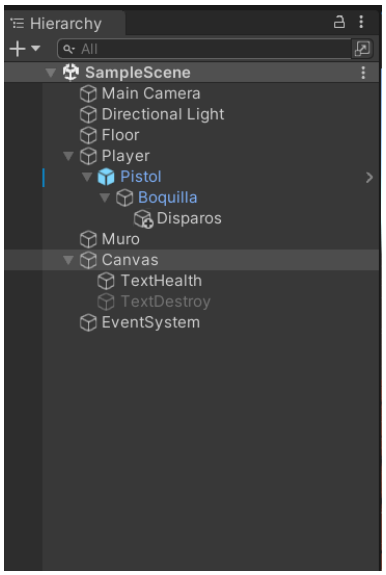

Figura 3: Hierarchy

2. Assets

Figura 4: Assets

- **Materiales**: Se creo materiales de tipo standard para el suelo y la pared. A diferencia del sprite para las partículas en este caso se uso el shader como UI para poder interactuar en las propiedades de las partículas.
- **Model y Prefabs**: El archivo model contiene el arma para convertirlo a prefrabs al igual que la explosión que es un objeto de sistema de partículas.
- **Scripts** Contiene los codigos para interaccion con el juego.

3. Materials and Sprites

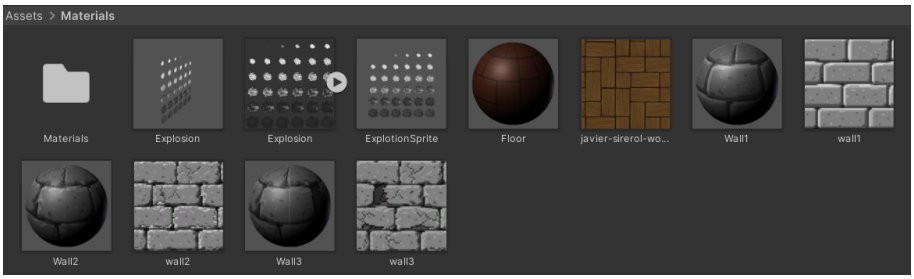

Figura 5: Materials and Sprite

4. Particles System (Explosion)

Se hizo una simulación de una explosión usando sistema de partículas y configurando las propiedades necesarias.

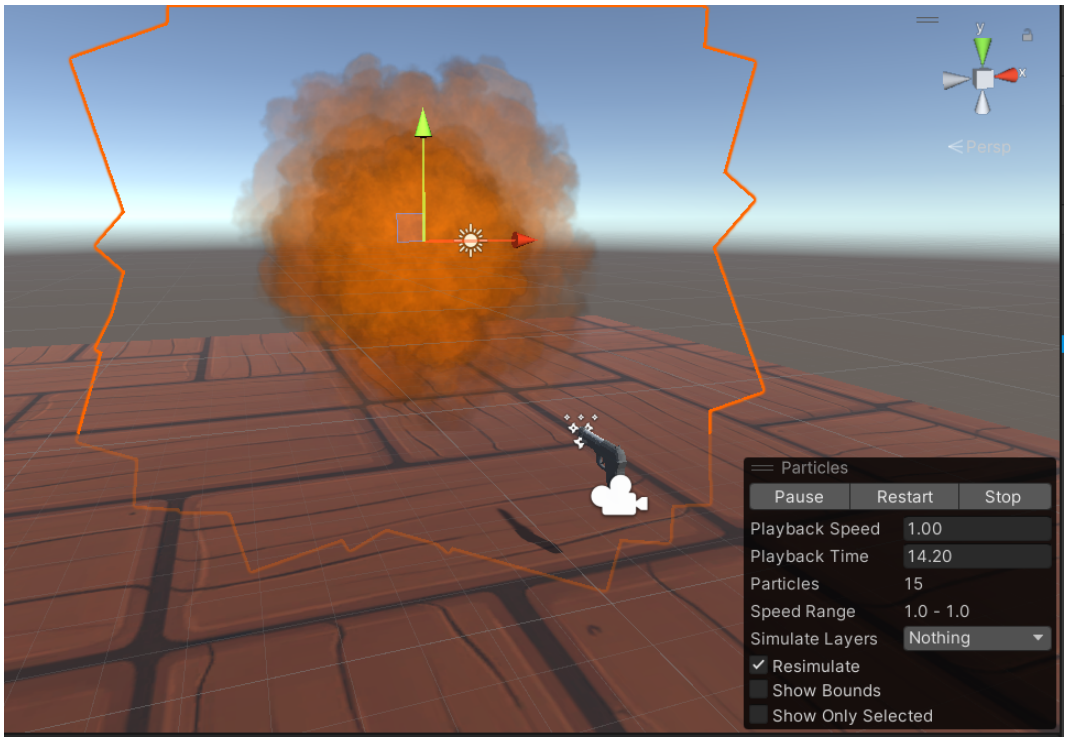

Figura 6: Explosion

4.1. Inspector del Prefabs Explosion

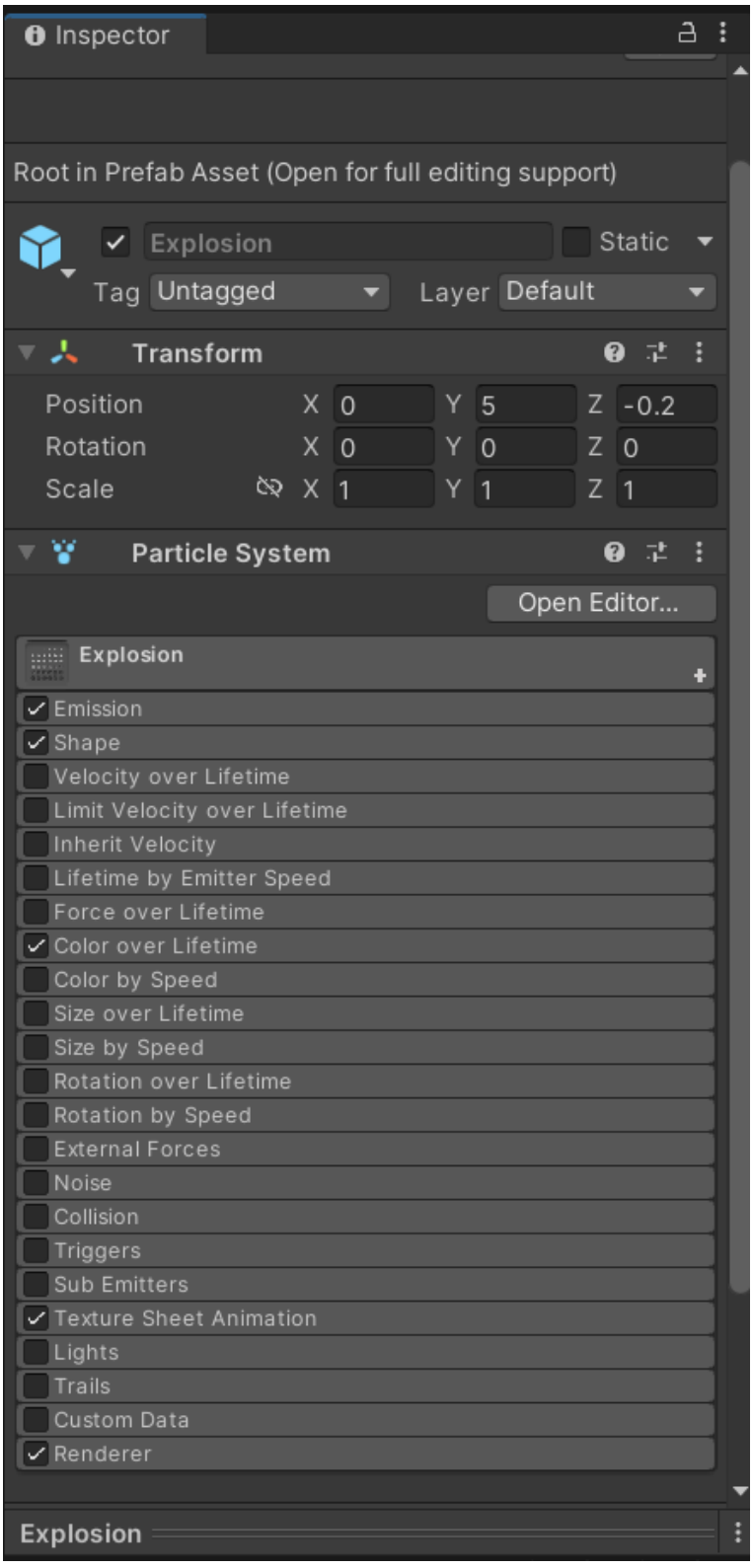

Figura 7: Inspector

Se configuro las propiedades siguientes:

- Emission
- Shape
- Color over lifetime
- Texture Sheet Animation
- Renderer

5. Particles System (Shoot)

Se hizo una simulación de disparos usando sistema de partículas y configurando las propiedades necesarias.

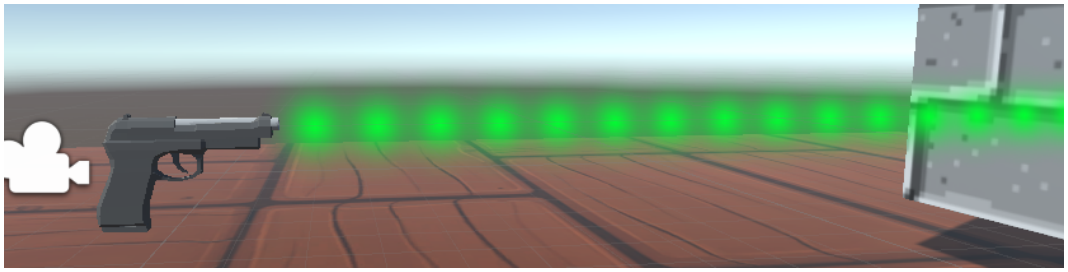

Figura 8: Shoot

5.1. Inspector del Prefabs Shoot

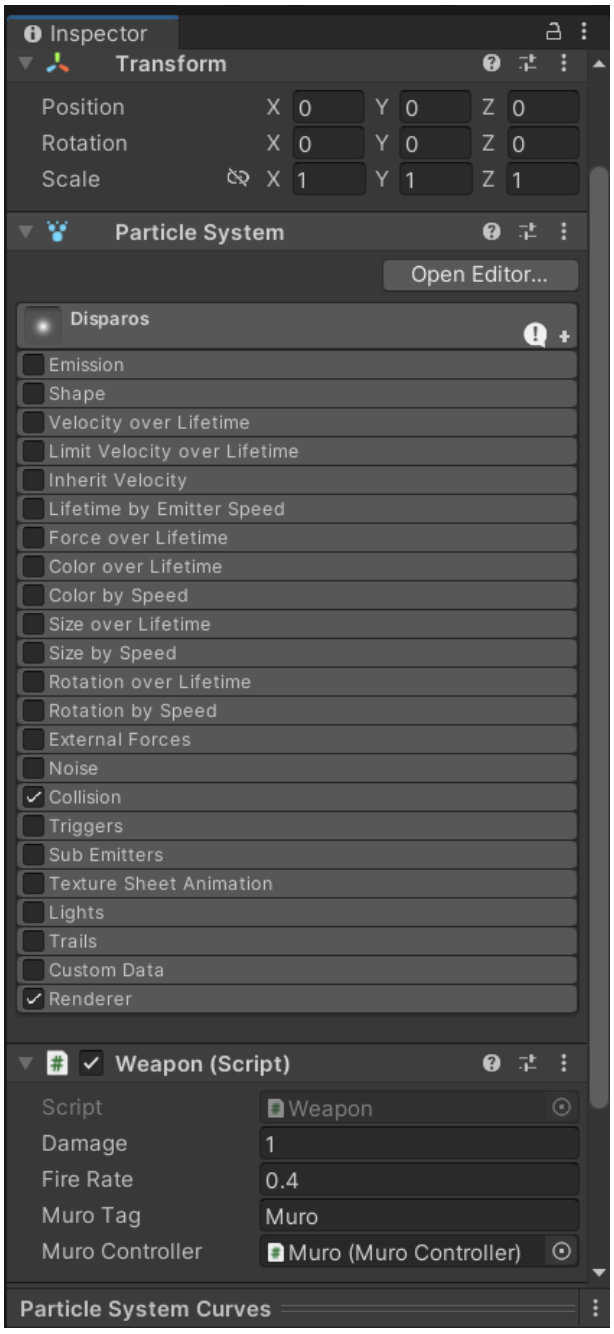

Figura 9: Inspector

Se configuro las propiedades siguientes:

- Disparos
- Collision

- Renderer
- También se agrego un script (Weapon) El cual tiene variables publicas como el daño del arma, tasa de duración del fuego, un tag para encontrar la colisión del Muro, y un script del MuroController ya que el muro tiene su resistencia y el momento en que explosiona.

6. Codigos

6.1. Move Camera

Código 1: MoveCamera.cs

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

public class MoveCamera : MonoBehaviour
{
    // Start is called before the first frame update

    //camera will follow this target
    public Transform target;
    //change this value to get desired smoothness
    public float smoothTime = 0.3f;
    // this value will change at the runtime depending on target movement
    private Vector3 velocity = Vector3.zero;

    public Vector3 offset;

    void Start()
    {

        offset = transform.position - target.position;
    }

    // Update is called once per frame
    void LateUpdate()
    {
        Vector3 targetPosition = target.position + offset;
        transform.position = Vector3.SmoothDamp(transform.position, targetPosition, ref velocity,
     ↪ smoothTime);
    }
}
```

6.2. Move Player

Código 2: MovePlayer.cs

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

public class MovePlayer : MonoBehaviour
{
    public float mouseSpeed = 2;
    public float speed = 5.0f;
    public float jumpSpeed = 5.0f;
    public float gravity = 20.8f;
    private Vector3 moveDirection;
    private CharacterController chCtrl;

    public GameObject CameraRot;
    private float X = 0;
    private float Y = 0;

    // Start is called before the first frame update
    void Start()
    {
        chCtrl = GetComponent<CharacterController>();
    }

    void FixedUpdate()
    {
        if (chCtrl.isGrounded)
        {
            moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0.0f, Input.GetAxis("Vertical"));
            moveDirection = transform.TransformDirection(moveDirection);
            moveDirection *= speed;
            if (Input.GetButton("Jump"))
            {
                moveDirection.y = jumpSpeed;
            }
        }
        moveDirection.y -= (gravity * Time.deltaTime);
        chCtrl.Move(moveDirection * Time.deltaTime);

        Y += Input.GetAxis("Mouse X") * mouseSpeed;
        X += Input.GetAxis("Mouse Y") * mouseSpeed;
        transform.eulerAngles = new Vector3(0, Y, 0);
        CameraRot.transform.eulerAngles = new Vector3(-X, Y, 0);
    }
}
```

6.3. Weapon

Este script esta enlazado con el sistema de partículas el cual rescata los eventos con las cuales el las partículas colisionan.

Código 3: Weapon.cs

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

public class Weapon : MonoBehaviour
{
    public int damage = 1;
    public float fireRate;
    private ParticleSystem particleSystem;

    private List<ParticleCollisionEvent> particleCollisionEvents;
    private bool fireCooldown = false;
    public string muroTag;

    public MuroController muroController;

    private void Start()
    {
        particleSystem = GetComponent<ParticleSystem>();
        particleCollisionEvents = new List<ParticleCollisionEvent>();
    }

    private void OnParticleCollision(GameObject other)
    {
        ParticlePhysicsExtensions.GetCollisionEvents(particleSystem,other,particleCollisionEvents);

        for (int i = 0; i < particleCollisionEvents.Count; i++)
        {
            var collider = particleCollisionEvents[i].colliderComponent;
            if (collider.CompareTag(muroTag))
            {
                muroController.health -= damage;
                muroController.SetHealthText();
                //var Health = collider.GetComponent<HealthController>();
                //Health.ApplyDamage(damage);
            }
        }
    }
    public void Fire()
    {
        if (fireCooldown) return;
        particleSystem.Emit(1);
        fireCooldown = true;
        StartCoroutine(StopCooldownAfterTime());
    }

    private IEnumerator StopCooldownAfterTime()
    {
        yield return new WaitForSeconds(fireRate);
        fireCooldown = false;
```

```
52    }
53 }
```

6.4. WeaponController

Este script recibe las acciones cuando el usuario dispara si lo hace invoca los métodos de la clase Weapon.

Código 4: WeaponController.cs

```
1  using Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class WeaponController : MonoBehaviour
6  {
7     public Weapon weapon;
8     public string muroTag;
9
10    private bool fire;
11    // Start is called before the first frame update
12    void Start()
13    {
14       weapon.muroTag = muroTag;
15    }
16
17    // Update is called once per frame
18    void Update()
19    {
20       if (Input.GetMouseButtonDown(0))
21       {
22          fire = true;
23       }
24       if(Input.GetMouseButtonUp(0))
25       {
26          fire = false;
27       }
28       if (fire)
29       {
30          weapon.Fire();
31       }
32    }
33 }
```

6.5. MuroController

Este script contiene la vida útil del muro junto con el text que indica cuando de vida le queda. También tiene un método el cual instancia el prefabs Explosion (System Particles)c cuando la vida

útil del muro llegue a cero.

Código 5: MuroController.cs

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

public class MuroController : MonoBehaviour
{


    public TextMeshProUGUI healthText;
    public GameObject destroyText;
    public GameObject particleSystemExplotion;
    public GameObject muro;
    public Material Wall2;
    public Material Wall3;
    public int health = 10;
    private int sethealth;
    public bool isPlaying = false;
    void Start()
    {
        //explotion = particleSystemExplotion.GetComponent<ParticleSystem>();
        //var emission = explotion.emission;
        //emission.enabled = true;
        sethealth = health;
        SetHealthText();
    }
    public void SetHealthText()
    {
        healthText.text = "Wall Health: " + health.ToString();
        if (health <= 0)
        {
            healthText.text = "";
            destroyText.SetActive(true);
            isPlaying = true;
            Destruction();
        }else if(health == sethealth / 2)
        {
            muro.GetComponent<MeshRenderer>().material = Wall2;
        }else if(health == (int)sethealth * 0.2)
        {
            muro.GetComponent<MeshRenderer>().material = Wall3;
        }

    }
    public void Destruction()
    {
        Debug.Log("Destruction!!!!!");
        Destroy(muro.gameObject);
```

```
49          Instantiate(particleSystemExplotion, muro.transform.position, Quaternion.identity);
50          ActiveEmission();
51      }
52
53      public void ActiveEmission()
54      {
55
56          //var emission = explotion.emission;
57          //emission.enabled = false;
58          //emission.SetBursts(new ParticleSystem.Burst[] { new ParticleSystem.Burst(0,5,5,0.010f)});
59          ParticleSystem explotion = particleSystemExplotion.GetComponent<ParticleSystem>();
60          if (isPlaying)
61          {
62              Debug.Log("Play");
63              explotion.Play();
64              //explotion.Pause();
65              isPlaying = false;
66          }
67      }
68  }
```

Referencias

- **Grabación de Explicación:** https://drive.google.com/file/d/1wfCVLNyj-komjHYEWByOV3VwBqG77cAn/view?usp=sharing
- https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html
- https://sketchfab.com/3d-models/pistol-34ebc77c59eb47ef89e0463469237dac
- https://www.youtube.com/watch?v=mTBDT_TTFbM
- https://www.youtube.com/watch?v=pl253L-yCD0
- https://www.youtube.com/watch?v=OdpS4ALrGFI
- https://www.youtube.com/watch?v=q4vMuQKely8
- https://www.youtube.com/watch?v=cvQiQglPI18