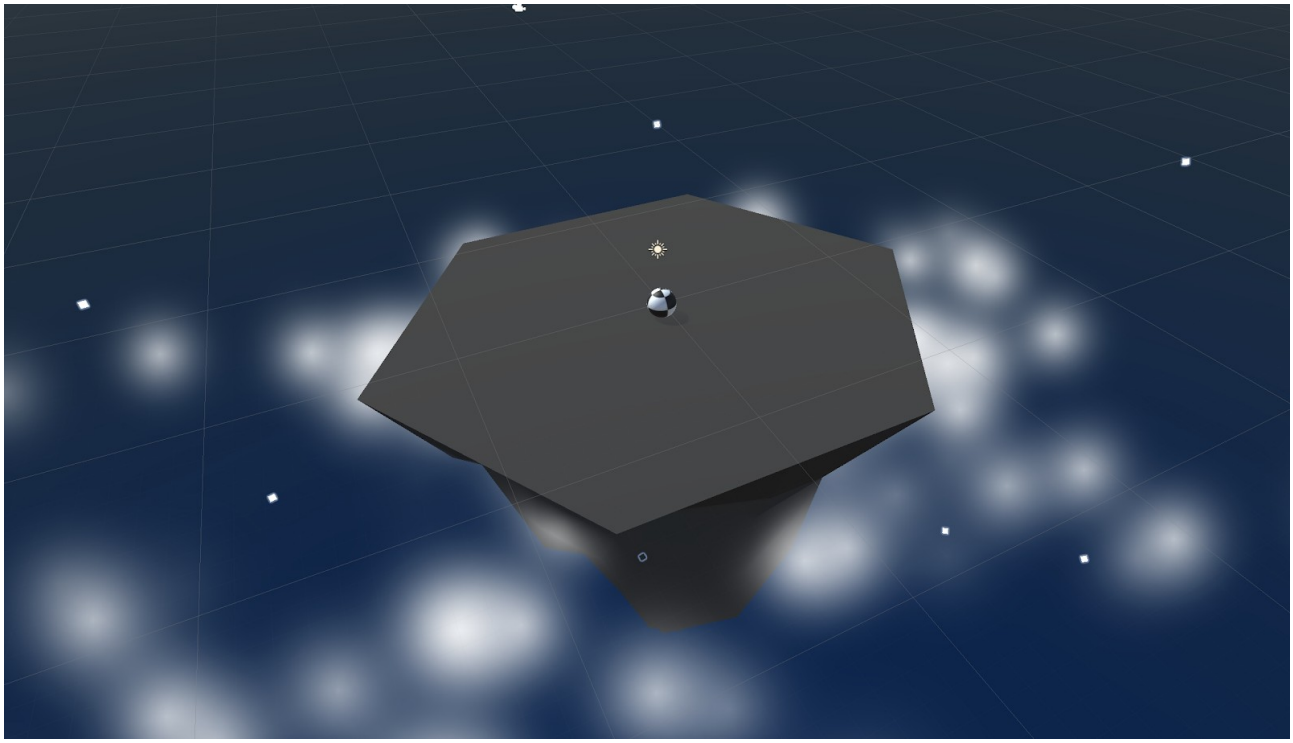
Prototipo 4



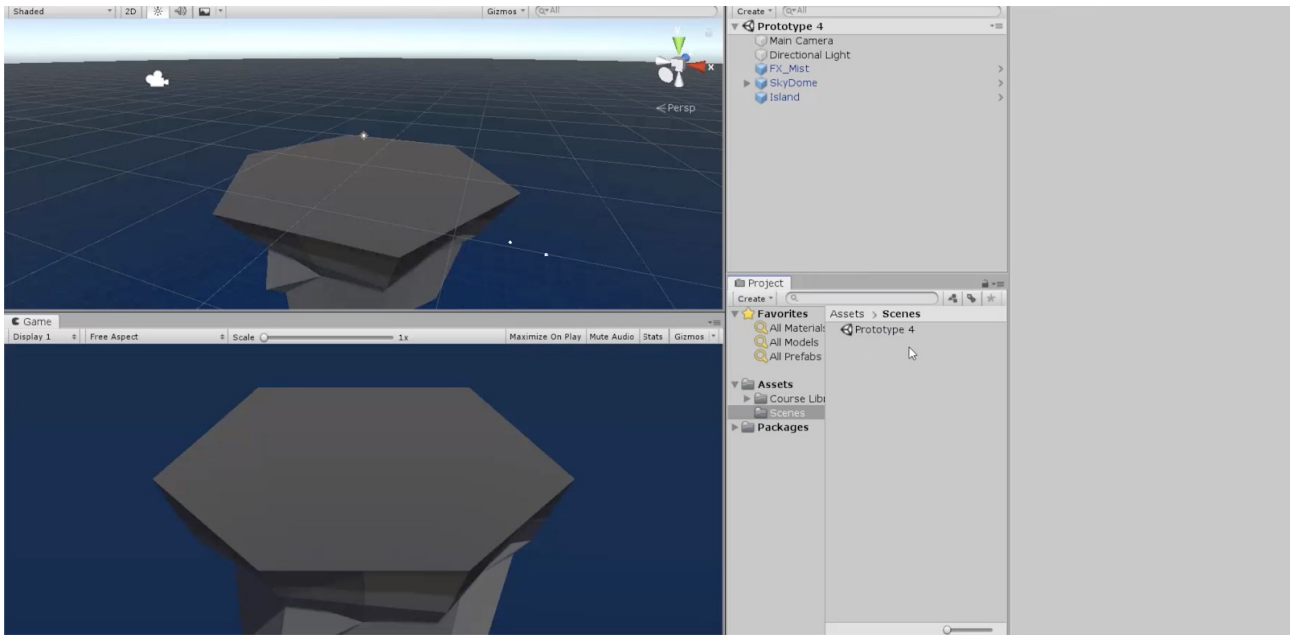
Mira hacia donde vas

Crearemos un nuevo prototipo y descargaremos los archivos iniciales. Notarás un hermoso efecto de isla, cielo y partículas... todos los cuales se pueden personalizar. A continuación, permitiremos que el jugador gire la cámara alrededor de la isla en un radio perfecto, proporcionando la vista de la escena. El jugador estará representado por una esfera, envuelta en una textura detallada de tu elección.
Finalmente agregaremos fuerza al jugador, permitiéndole moverse hacia adelante o hacia atrás en la dirección de la cámara.

La cámara girará uniformemente alrededor de un punto focal en el centro de la isla, siempre que el jugador indique una entrada horizontal. El jugador controlará una esfera texturizada y la moverá hacia adelante o hacia atrás en la dirección del punto focal de la cámara.

1. Crear proyecto y abrir escena.
Debemos comenzar un nuevo proyecto e importar los archivos iniciales.

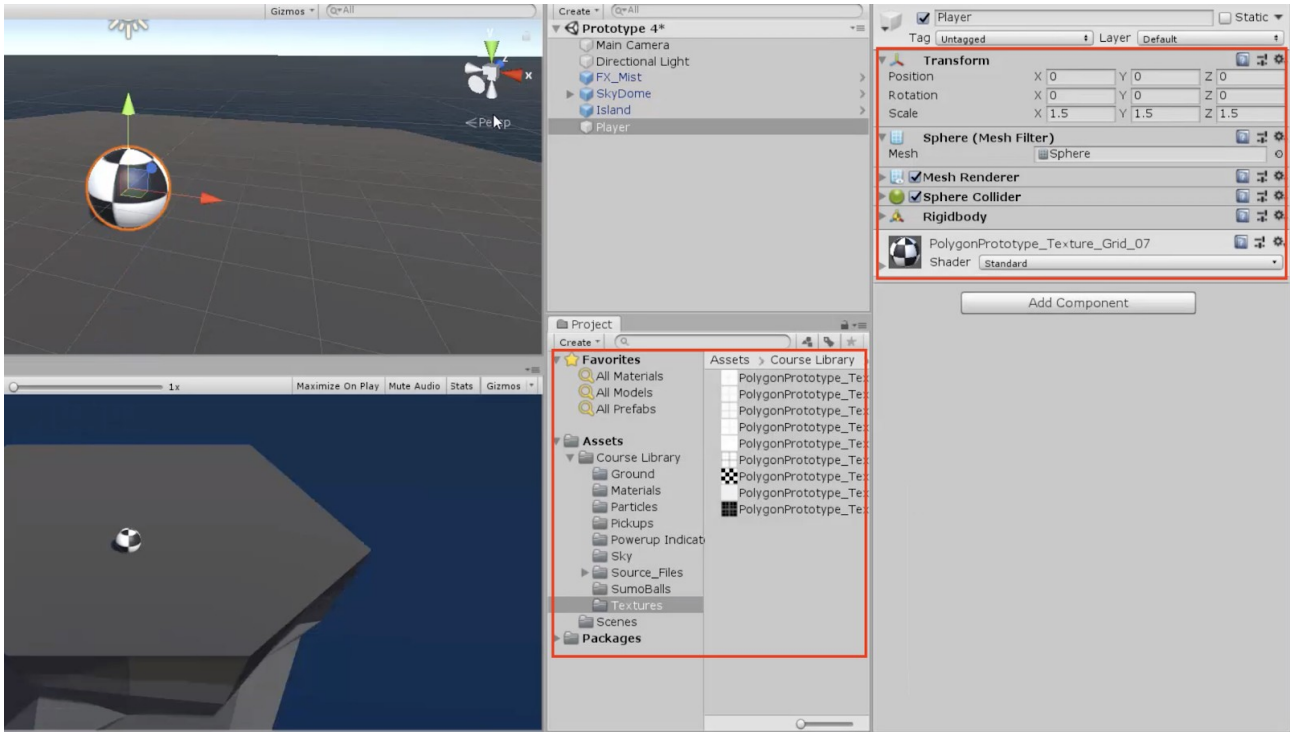
- Abre Unity Hub y crea un proyecto “Prototipo 4” vacío en el directorio del curso en la versión correcta de Unity.
- Haz clic para descargar los archivos iniciales del Prototipo 4, extrae la carpeta comprimida y luego importa el paquete .unity a tu proyecto.
- Abre la escena Prototype 4 y elimina la escena de muestra sin guardarla.
- Haz clic en Ejecutar para ver los efectos de partículas.



2. Configura al jugador y agrega una textura.

Tenemos una isla para que se desarrolle el juego y ahora necesitamos una esfera para que el jugador la controle y gire.

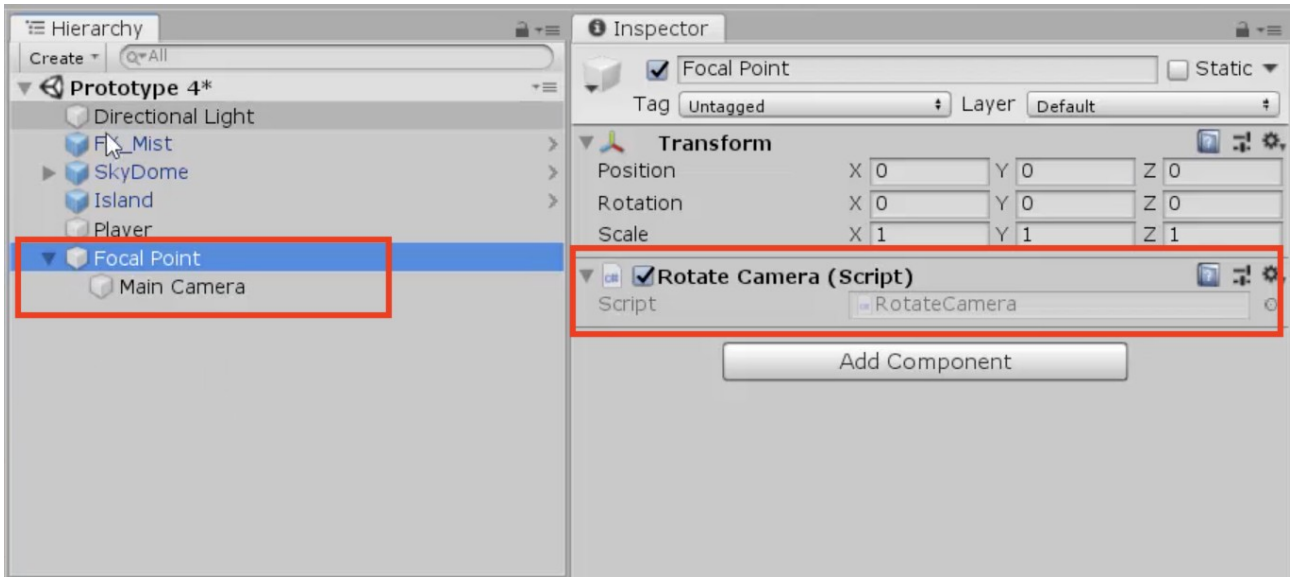
- En la ventana de jerarquía, crea 3D Object > Sphere
- Cámbiale el nombre a "Player", resetea su posición y aumente su escala XYZ a 1,5.
- Agrega un componente RigidBody al player
- Desde Library > Textures, arrastra una textura a la esfera.



3. Crea un punto focal para la cámara.

Si queremos que la cámara gire alrededor del juego de forma fluida y cinematográfica, debemos fijarla en el centro de la isla con un punto focal.

- Crea un nuevo Objeto Vacío y cámbiale el nombre a "Focal Point".
- Restablece su posición al origen (0, 0, 0) y convierte la cámara en un objeto secundario o hijo del mismo.
- Crea una nueva carpeta "Scripts" y un nuevo script "RotateCamera" dentro de ella.
- Adjunta el script "RotateCamera" al objeto vacío "Focal Point".



4. Gira el punto focal según la entrada del usuario.

Ahora que la cámara está conectada al punto focal, el jugador debe poder rotarla alrededor de la isla con el control de la entrada horizontal.

- Crea el código para rotar la cámara según la velocidad de rotación y la entrada horizontal.
- Modifica el valor de la velocidad de rotación para obtener la velocidad que desees.

```
public class RotateCamera : MonoBehaviour
{
    public float rotationSpeed;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        float horizontalInput = Input.GetAxis("Horizontal");
        transform.Rotate(Vector3.up, horizontalInput * rotationSpeed *
Time.deltaTime);
    }
}
```

5. Añade fuerza hacia adelante al jugador.

La cámara gira perfectamente alrededor de la isla, pero ahora necesitamos mover al jugador.

- Crea un nuevo script "PlayerController", aplícalo al objeto Player y ábrelo.
- Declara una nueva variable **public float speed** e inicialízala.
- Declara una nueva variable **private Rigidbody playerRb** e inicialízalo en Start()
- En Update(), declara una nueva variable **forwardInput** basada en la entrada "Vertical"

- Crea y llama un método **AddForce()** para mover al Player hacia adelante según **forwardInput.**

```
public class PlayerController : MonoBehaviour
{
    private Rigidbody playerRb;
    public float speed = 5.0f;
    // Start is called before the first frame update
    void Start()
    {
        playerRb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    void Update()
    {
        float forwardInput = Input.GetAxis("Vertical");
        playerRb.AddForce(Vector3.forward * speed * forwardInput);
    }
}
```

6. Muévete en dirección al punto focal.

Tenemos la pelota rodando, pero solo avanza y retrocede en una única dirección. En cambio, debería moverse en la dirección hacia la que mira la cámara (y el punto focal).

- Declara un nuevo **private GameObject focalPoint**; e inicialízalo en **Start(): focalPoint = GameObject.Find("Focal Point");**
- En la llamada AddForce, reemplaza **Vector3.forward** con **focalPoint.transform.forward**

```
public class PlayerController : MonoBehaviour
{
    private Rigidbody playerRb;
    public float speed = 5.0f;
    private GameObject focalPoint;
    // Start is called before the first frame update
    void Start()
    {
        playerRb = GetComponent<Rigidbody>();
        focalPoint = GameObject.Find("FocalPoint");
    }

    // Update is called once per frame
    void Update()
    {
        float forwardInput = Input.GetAxis("Vertical");
        //playerRb.AddForce(Vector3.forward * speed * forwardInput);
        playerRb.AddForce(focalPoint.transform.forward * speed * forwardInput);
    }
}
```

Sigue al jugador

El jugador puede rodar a su antojo... pero no tiene ningún propósito. Vamos a crear un enemigo para desafiar al jugador.

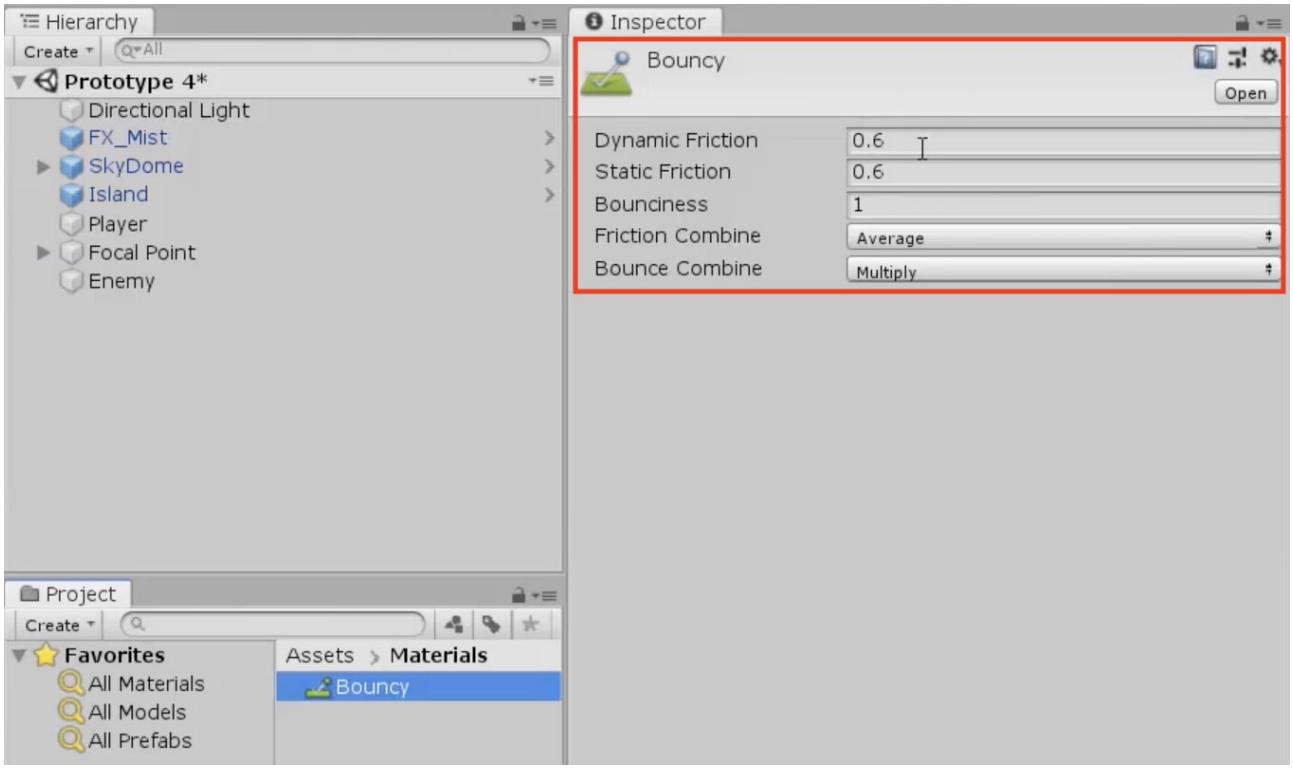
Primero le daremos al enemigo una textura de tu elección, luego le daremos la capacidad de hacer rebotar al jugador... potencialmente tirándolo por el precipicio. Por último, dejaremos que el enemigo persiga al jugador por la isla y aparezca en posiciones aleatorias.

Un enemigo texturizado y esférico aparecerá en la isla al principio, en una ubicación aleatoria determinada por una función personalizada. Perseguirá al jugador por la isla, haciéndolo rebotar hacia el borde si se acerca demasiado.

7. Añade un enemigo y un material físico.

Nuestra rotación de cámara y movimiento de jugador funcionan. A continuación vamos a configurar un enemigo y darle una características físicas especiales para hacer rebotar al jugador.

- Crea una nueva esfera, cámbiale el nombre a "Enemy", resetea su posición y arrastra una textura sobre ella.
- Agrégale un nuevo componente RigidBody y ajusta su escala XYZ, luego prueba como va.
- Crea una nueva carpeta "Physics Materials" y dentro haz **Create > Physics Material**, luego dale el nombre "**Bouncy**".
- Aumenta la propiedad rebote (Bounciness) a 1, cambia **Bounce Combine** a "**Multiply**", aplícalo al jugador y al enemigo, luego prueba



8. Crea un script para el enemigo para que siga al jugador.

El enemigo tiene el poder de hacer rebotar al jugador, pero sólo si el jugador se acerca. Debemos decirle al enemigo que siga la posición del jugador, persiguiéndolo por toda la isla.

- Crea un nuevo script de "Enemy" y añádelo al objeto Enemy.
- Declara 3 nuevas variables para **Rigidbody enemyRb;**, **GameObject player;**, y **public float speed;**
- Inicialice **enemyRb = GetComponent Rigidbody>();** y **player = GameObject.Find("Player");**

- En Update(), aplica AddForce hacia la dirección entre el jugador y el enemigo.

```
public class Enemy : MonoBehaviour
{
    public float speed = 3.0f;
    private Rigidbody enemyRb;
    private GameObject player;
    // Start is called before the first frame update
    void Start()
    {
        enemyRb = GetComponent<Rigidbody>();
        player = GameObject.Find("Player");
    }

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

        enemyRb.AddForce((player.transform.position - transform.position).normalized *
speed);
    }
}
```

9. Crea una variable lookDirection.

El enemigo ahora avanza hacia el jugador, pero nuestro código es un poco confuso. Vamos a limpiarlo un poco agregando una variable para el nuevo vector.

- En Update(), declara una nueva variable **Vector3 lookDirection**
- Establece V**ector3 lookDirection = (player.transform.position - transform.position).normalized;**
- Usa la variable lookDirection en la llamada AddForce

```
void Update()
{
    Vector3 lookDirection = (player.transform.position -
transform.position).normalized;
    enemyRb.AddForce(lookDirection * speed);
}
```

10. Crea un gestor de generación para el enemigo (Spawn Manager).

Ahora que el enemigo está actuando exactamente como queremos, lo convertiremos en un prefab para que un administrador de generación pueda crear una instancia de él.

- Arrastra el objeto Enemy a la carpeta Prefabs (créala antes si no la tienes creada) para crear un nuevo Prefab, luego elimina el objeto Enemy de la escena.
- Crea un nuevo objeto vacío "Spawn Manager", añádele un nuevo script "SpawnManager" y ábrelo.
- Declara una nueva variable pública **public GameObject enemyPrefab** y luego asigna el prefab del enemigo desde la ventana del inspector.
- En Start(), crea una instancia de un nuevo **enemyPrefab** en una ubicación predeterminada.

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    // Start is called before the first frame update
    void Start()
    {
        Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
    }

    // Update is called once per frame
```

```
    void Update()
    {

    }
}
```

11. Genera aleatoriamente la posición de generación.

El enemigo aparece al principio, pero siempre aparece en el mismo lugar. Usando la clase Random, podemos generar al enemigo en una posición aleatoria.

- En SpawnManager.cs, en Start(), crea nuevos valores X y Z generados aleatoriamente
- Crea una nueva variable **Vector3 randomPos** con esas posiciones aleatorias de X y Z
- Incorpora la nueva variable **randomPos** en la llamada Instantiate
- Reemplaza los valores codificados con una variable **spawnRange**
- Inicia y reinicia el proyecto para asegurarte de que está funcionando.

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    private float spawnRange = 9;
    // Start is called before the first frame update
    void Start()
    {
        float spawnPosX = Random.Range(- spawnRange, spawnRange);
        float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
    }

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

    }
}
```

12. Haz que un método devuelva un punto de generación.

El código que usamos para generar una posición de inicio aleatoria está bien y lo usaremos mucho. Si queremos limpiar el script y usar este código más adelante, debemos almacenarlo en una función personalizada.

- Crea una nueva función **Vector3 GenerateSpawnPosition() { }**
- Copia y pegua las variables **spawnPosX** y **spawnPosZ** en el nuevo método.
- Agrega la línea para devolver **randomPos**; en tu nuevo método
- Reemplaza el código en su llamada Instantiate con el nuevo nombre de función: **GenerateSpawnPosition()**

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    private float spawnRange = 9;
    // Start is called before the first frame update
    void Start()
    {
        //float spawnPosX = Random.Range(- spawnRange, spawnRange);
        //float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        //Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
```

```
        Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
    }
    // Update is called once per frame
    void Update()
    {
    }
    private Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = Random.Range(-spawnRange, spawnRange);
        float spawnPosZ = Random.Range(-spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        return randomPos;
    }
}
```

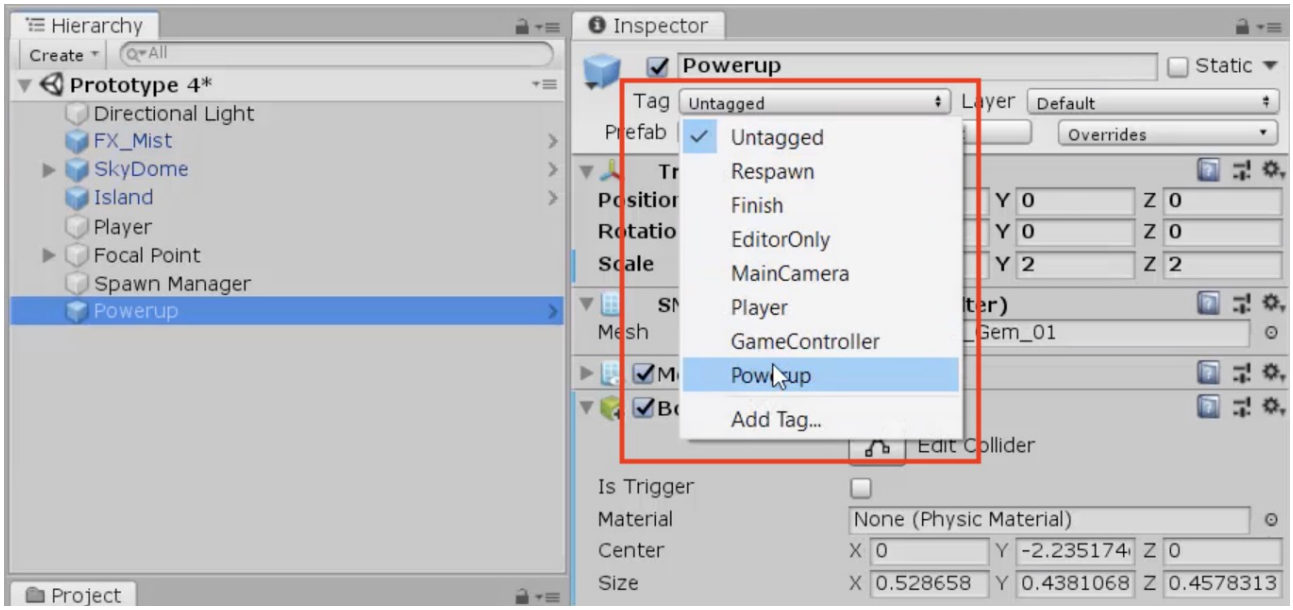
Potenciador y cuenta atrás

El enemigo persigue al jugador por la isla, pero el jugador necesita una mejor forma de defenderse... especialmente si añadimos más enemigos. En este apartado, vamos a crear un potenciador que le dará al jugador un impulso de fuerza temporal, alejando a los enemigos que entren en contacto. El potenciador aparecerá en una posición aleatoria en la isla y resaltará al jugador con un indicador cuando lo recoja. El indicador del potenciador y el potenciador en sí estarán representados por recursos de juego de tu elección.

El funcionamiento es el siguiente: aparecerá un potenciador en una posición aleatoria en el mapa. Una vez que el jugador colisiona con este potenciador, el potenciador desaparecerá y el jugador quedará resaltado con un indicador. El encendido durará una cierta cantidad de segundos después de recogerlo, lo que le otorga al jugador una súper fuerza que destruye a los enemigos.

13. Elige y prepara un potenciador.
Para agregar una mecánica de juego completamente nueva a este proyecto, introduciremos un nuevo objeto potenciador que le dará al jugador superpoderes temporales.

- Desde la Biblioteca, arrastra un objeto Powerup a la escena, cámbiale el nombre a "Powerup" y edita su escala y posición.
- Agrega un Box Collider al potenciador, haz clic en Edit Collider para asegurarte de que encaje y luego marque la casilla de verificación "Is Trigger".
- Crea una nueva etiqueta "Powerup" y aplíquela al potenciador.
- Arrastra el potenciador a la carpeta Prefabs para crear un nuevo "Original Prefab"



14. Destruye el potenciador en caso de colisión.
Como primer paso para que el potenciador funcione, lo haremos desaparecer cuando el jugador lo presione y configuraremos una nueva variable booleana para rastrear que el jugador lo obtuvo.

- En PlayerController.cs, agrega un nuevo método OnTriggerEnter()
- Agrega una estructura if que destruya el potenciador **other.CompareTag("Powerup")** en caso de colisión.
- Crea una nueva variable booleana **public bool hasPowerup;** y establece hasPowerup = true; cuando chocas con el Powerup.

```
public class PlayerController : MonoBehaviour
{
    private Rigidbody playerRb;
    public float speed = 5.0f;
    private GameObject focalPoint;
    public bool hasPowerup=false;
    // Start is called before the first frame update
    void Start()
    {
        playerRb = GetComponent<Rigidbody>();
        focalPoint = GameObject.Find("FocalPoint");
    }

    // Update is called once per frame
    void Update()
    {
        float forwardInput = Input.GetAxis("Vertical");
        //playerRb.AddForce(Vector3.forward * speed * forwardInput);
        playerRb.AddForce(focalPoint.transform.forward * speed * forwardInput);
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("PowerUp"))
        {
            hasPowerup = true;
            Destroy(other.gameObject);
        }
    }
}
```

15. Prueba el enemigo y el potenciador.

El potenciador solo entrará en juego en una circunstancia muy particular: cuando el jugador tiene un potenciador y choca con un enemigo, por lo que primero probaremos esa condición tan específica.

- Crea una nueva etiqueta "Enemy" y aplícala al prefab Enemy.
- En PlayerController.cs, agrega la función OnCollisionEnter()
- Crea la estructura if con la prueba de doble condición para la etiqueta enemiga y el valor booleano hasPowerup.
- Crea un Debug.Log para asegurarte de que esté funcionando.

```
 private void OnCollisionEnter(Collision collision)
 {
     if (collision.gameObject.CompareTag("Enemy") && hasPowerup)
     {
         Debug.Log("Collided with " + collision.gameObject.name + " with powerup set
to " + hasPowerup);
     }
 }
```

16. Aplica fuerza adicional en el choque con el potenciador.

Con las condiciones para el potenciador configuradas, ahora estamos listos para programar la habilidad del potenciador real: cuando el jugador choca con un enemigo, el enemigo debería salir volando

- En OnCollisionEnter() declara una nueva variable local para obtener el componente Rigidbody del enemigo.
- Declara una nueva variable para obtener la dirección hacia donde alejarse del jugador.
- Agrega una fuerza de impulso al enemigo, usando una nueva variable powerupStrength

```
private void OnCollisionEnter(Collision collision)
{
    //if (collision.gameObject.CompareTag("Enemy") && hasPowerup)
```

```
    //{
    //    Debug.Log("Collided with " + collision.gameObject.name + " with powerup set
to " + hasPowerup);
    //}
    if (collision.gameObject.CompareTag("Enemy") && hasPowerup)
    {
        Rigidbody enemyRigidbody = collision.gameObject.GetComponent<Rigidbody>();
        Vector3 awayFromPlayer = (collision.gameObject.transform.position -
transform.position);
        Debug.Log("Player collided with " + collision.gameObject + " with powerup set
to " + hasPowerup);
        enemyRigidbody.AddForce(awayFromPlayer * powerupStrength, ForceMode.Impulse);
    }
}
```

17. Crear una rutina de cuenta atrás para el potenciador.

No sería justo para los enemigos si el potenciador durara para siempre, por lo que programaremos un temporizador de cuenta atrás que comienza cuando el jugador recoge el potenciador, eliminando la capacidad del potenciador cuando finaliza el cronómetro.

- Agrega un nuevo **IEnumerator PowerupCountdownRoutine () {}**
- Dentro de PowerupCountdownRoutine, espera 7 segundos y luego desactiva el potenciador.
- Cuando el jugador choca con el encendido, inicia la rutina.

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("PowerUp"))
    {
        hasPowerup = true;
        Destroy(other.gameObject);
        StartCoroutine(PowerupCountdownRoutine());
    }
}

IEnumerator PowerupCountdownRoutine()
{
    yield return new WaitForSeconds(7);
    hasPowerup = false;
}
```

18. Agrega un indicador de potenciador.

Para que este juego sea mucho más jugable, debe quedar claro cuándo el jugador tiene o no el potenciador, por lo que programaremos un indicador visual para mostrárselo al usuario.

- Desde la Biblioteca, arrastra un objeto de potenciado a la escena, cámbiale el nombre a "Powerup Indicator" y edita su escala.
- Desmarca la casilla de verificación "Active" en el inspector.
- En PlayerController.cs, declara una nueva variable **public GameObject powerupIndicator** y luego asigna la variable **Powerup Indicator** en el inspector.
- Cuando el jugador colisiona con el potenciador, configura el objeto indicador en Activo, luego configúralo en Inactivo cuando expire el potenciador.
- En Update(), establece la posición del indicador en la posición del jugador + un valor de offset

```
public GameObject powerupIndicator;
// Start is called before the first frame update
void Start()
{
    playerRb = GetComponent<Rigidbody>();
    focalPoint = GameObject.Find("FocalPoint");
```

```
    }

    // Update is called once per frame
    void Update()
    {
        float forwardInput = Input.GetAxis("Vertical");
        //playerRb.AddForce(Vector3.forward * speed * forwardInput);
        playerRb.AddForce(focalPoint.transform.forward * speed * forwardInput);
        powerupIndicator.transform.position = transform.position + new Vector3(0, -
0.5f, 0);
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("PowerUp"))
        {
            hasPowerup = true;
            Destroy(other.gameObject);
            powerupIndicator.gameObject.SetActive(true);
            StartCoroutine(PowerupCountdownRoutine());
        }
    }
    private void OnCollisionEnter(Collision collision)
    {
        //if (collision.gameObject.CompareTag("Enemy") && hasPowerup)
        //{
        //    Debug.Log("Collided with " + collision.gameObject.name + " with powerup
set to " + hasPowerup);
        //}
        if (collision.gameObject.CompareTag("Enemy") && hasPowerup)
        {
            Rigidbody enemyRigidbody = collision.gameObject.GetComponent<Rigidbody>();
            Vector3 awayFromPlayer = (collision.gameObject.transform.position -
transform.position);
            Debug.Log("Player collided with " + collision.gameObject + " with powerup
set to " + hasPowerup);
            enemyRigidbody.AddForce(awayFromPlayer * powerupStrength,
ForceMode.Impulse);
        }
    }
    IEnumerator PowerupCountdownRoutine()
    {
        yield return new WaitForSeconds(7);
        hasPowerup = false;
        powerupIndicator.gameObject.SetActive(false);
    }
}
```

Bucles For para crear oleadas

Tenemos todas las características de un gran juego: un jugador que rueda y gira la cámara, un potenciador que otorga súper fuerza y un enemigo que persigue al jugador hasta el final. Vamos a hacer unas últimas modificaciones.

Primero, mejoraremos el gestor de generación de enemigos, permitiéndole generar múltiples enemigos y aumentar su número cada vez que se derrote una oleada.

Por último, generaremos el potenciador con cada oleada, dándole al jugador la oportunidad de luchar contra una horda cada vez mayor de enemigos.

El Spawn Manager funcionará en oleadas, generando múltiples enemigos y un nuevo potenciador con cada iteración. Cada vez que los enemigos llegan a cero, se genera una nueva ola y aumenta el número de enemigos.

19. Escribe un bucle for para generar 3 enemigos.

Deberíamos desafiar al jugador generando más de un enemigo. Para hacerlo, repetiremos la creación de instancias del enemigo con un bucle.

- En SpawnManager.cs, en Start(), reemplaza la creación de instancias única con un bucle for que genere 3 enemigos.
- Mejora el código moviendo el bucle for a una nueva función vacía SpawnEnemyWave(), luego llama a esa función desde Start()

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    private float spawnRange = 9;
    // Start is called before the first frame update
    void Start()
    {
        //float spawnPosX = Random.Range(- spawnRange, spawnRange);
        //float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        //Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        SpawnEnemyWave();
    }
    // Update is called once per frame
    void Update()
    {
    }
    private Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = Random.Range(-spawnRange, spawnRange);
        float spawnPosZ = Random.Range(-spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        return randomPos;
    }
    void SpawnEnemyWave()
    {
        for (int i = 0; i < 3; i++)
        {
            Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        }
    }
}
```

20. Dale un parámetro al bucle for.

En este momento, SpawnEnemyWave genera exactamente 3 enemigos, pero si vamos a aumentar dinámicamente la cantidad de enemigos que aparecen durante el juego, debemos poder pasar información a ese método.

- Agrega un parámetro ***int enemiesToSpawn*** a la función ***SpawnEnemyWave***
- Reemplaza ***i*** < __ con ***i*** < ***enemiesToSpawn***
- Agrega esta nueva variable a la llamada de función en Start(): ***SpawnEnemyWave(___);***

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    private float spawnRange = 9;
    // Start is called before the first frame update
    void Start()
    {
        //float spawnPosX = Random.Range(- spawnRange, spawnRange);
        //float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        //Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        SpawnEnemyWave(3);
    }
    // Update is called once per frame
    void Update()
    {
    }
    private Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = Random.Range(-spawnRange, spawnRange);
        float spawnPosZ = Random.Range(-spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        return randomPos;
    }
    void SpawnEnemyWave(int enemiesToSpawn)
    {
        for (int i = 0; i < enemiesToSpawn; i++)
        {
            Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        }
    }
}
```

21. Destruye a los enemigos si se caen.

Una vez que el jugador se deshace de todos los enemigos necesitamos destruirlos y generar una nueva oleada :

- En Enemy.cs, destruye a los enemigos si su posición es menor que un cierto valor -Y
- En SpawnManager.cs, declara una nueva variable pública ***public int enemyCount***
- En Update(), establece ***enemyCount = FindObjectsOfType<Enemy>().Length;***
- Escribe la estructura if que indica que si ***enemyCount == 0*** entonces ***SpawnEnemyWave***

Fichero Enemy.cs

```
public class Enemy : MonoBehaviour
{
    public float speed = 3.0f;
    private Rigidbody enemyRb;
    private GameObject player;
    // Start is called before the first frame update
    void Start()
    {
        enemyRb = GetComponent<Rigidbody>();
        player = GameObject.Find("Player");
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 lookDirection = (player.transform.position -
transform.position).normalized;
        enemyRb.AddForce(lookDirection * speed);
        if (transform.position.y < -10) {
            Destroy(gameObject);
        }
    }
}
```

Fichero SpawnManager.cs

```
    void Start()
    {
        //float spawnPosX = Random.Range(- spawnRange, spawnRange);
        //float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        //Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        SpawnEnemyWave(3);
    }
    // Update is called once per frame
    void Update()
    {
        enemyCount = FindObjectsOfType<Enemy>().Length;
        if (enemyCount == 0) {
            SpawnEnemyWave(1);
        }
    }
    private Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = Random.Range(-spawnRange, spawnRange);
        float spawnPosZ = Random.Range(-spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        return randomPos;
    }
    void SpawnEnemyWave(int enemiesToSpawn)
    {
        for (int i = 0; i < enemiesToSpawn; i++)
        {
            Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        }
    }
}
```

22. Aumenta el número de enemigos con cada oleada.

Ahora que controlamos la cantidad de enemigos que aparecen, deberíamos aumentar su número en oleadas. Cada vez que el jugador derrota una oleada de enemigos, deberían surgir más para ocupar su lugar.

- Declara una nueva variable ***public int waveNumber = 1;***, luego úsala en la llamada a SpawnEnemyWave: ***SpawnEnemyWave(waveNumber);***
- En la declaración if que prueba si quedan 0 enemigos, incrementa el número de la oleada en 1.

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    private float spawnRange = 9;
    public int enemyCount;
    public int waveNumber = 1;
    // Start is called before the first frame update
    void Start()
    {
        //float spawnPosX = Random.Range(- spawnRange, spawnRange);
        //float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        //Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        //SpawnEnemyWave(3);
        SpawnEnemyWave(waveNumber);
    }
    // Update is called once per frame
    void Update()
    {
        enemyCount = FindObjectsOfType<Enemy>().Length;
        if (enemyCount == 0) {
            //SpawnEnemyWave(1);
            waveNumber++;
            SpawnEnemyWave(waveNumber);
        }
    }
    private Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = Random.Range(-spawnRange, spawnRange);
        float spawnPosZ = Random.Range(-spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        return randomPos;
    }
    void SpawnEnemyWave(int enemiesToSpawn)
    {
        for (int i = 0; i < enemiesToSpawn; i++)
        {
            Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        }
    }
}
```

23. Genera potenciadores con nuevas oleadas.

Nuestro juego está casi completo, pero nos falta algo. Los enemigos continúan apareciendo con cada oleada, pero el potenciador se usa una vez y desaparece para siempre, dejando al jugador vulnerable. Necesitamos generar el potenciador en una posición aleatoria con cada oleada, para que el jugador tenga la oportunidad de contraatacar.

- En SpawnManager.cs, declara una nueva variable pública ***public GameObject powerupPrefab***, asignale el prefab del potenciador en el inspector y elimínalo de la escena.
- En Start(), crea una instancia de un nuevo potenciador.
- Antes de la llamada ***SpawnEnemyWave()***, crea una instancia de un nuevo potenciador.

```
public class SpawnManager : MonoBehaviour
{
    public GameObject enemyPrefab;
    private float spawnRange = 9;
    public int enemyCount;
    public int waveNumber = 1;
    public GameObject powerupPrefab;
    // Start is called before the first frame update
    void Start()
    {
        //float spawnPosX = Random.Range(- spawnRange, spawnRange);
        //float spawnPosZ = Random.Range( - spawnRange, spawnRange);
        //Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        //Instantiate(enemyPrefab, new Vector3(0, 0, 6),
enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, randomPos, enemyPrefab.transform.rotation);
        //Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        //SpawnEnemyWave(3);
        SpawnEnemyWave(waveNumber);
        Instantiate(powerupPrefab, GenerateSpawnPosition(),
powerupPrefab.transform.rotation);
    }
    // Update is called once per frame
    void Update()
    {
        enemyCount = FindObjectsOfType<Enemy>().Length;
        if (enemyCount == 0) {
            //SpawnEnemyWave(1);
            waveNumber++;
            Instantiate(powerupPrefab, GenerateSpawnPosition(),
powerupPrefab.transform.rotation);
            SpawnEnemyWave(waveNumber);
        }
    }
    private Vector3 GenerateSpawnPosition()
    {
        float spawnPosX = Random.Range(-spawnRange, spawnRange);
        float spawnPosZ = Random.Range(-spawnRange, spawnRange);
        Vector3 randomPos = new Vector3(spawnPosX, 0, spawnPosZ);
        return randomPos;
    }
    void SpawnEnemyWave(int enemiesToSpawn)
    {
        for (int i = 0; i < enemiesToSpawn; i++)
        {
            Instantiate(enemyPrefab, GenerateSpawnPosition(),
enemyPrefab.transform.rotation);
        }
    }
}
```