



## **INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN ÁREA ENTORNOS VIRTUALES Y NEGOCIOS DIGITALES.**

**“Programación de videojuegos 1 unidad II”**

**Tutorial 2:**

# **Documentación del tutorial 2**

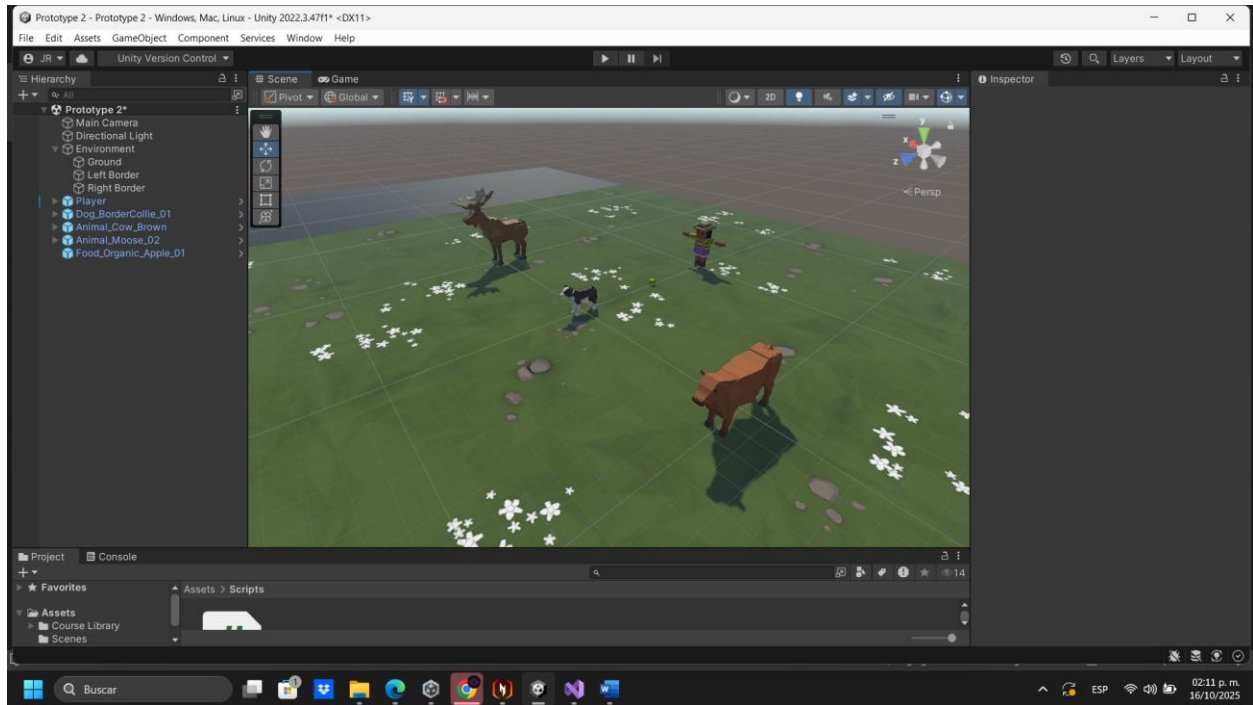
**Profesor: Gabriel Barrón Rodríguez.**

**Alumno: Jesús Alberto Arriaga Ramírez**  
**No. de Control: 1223100850**

**16 de octubre del 2025.**

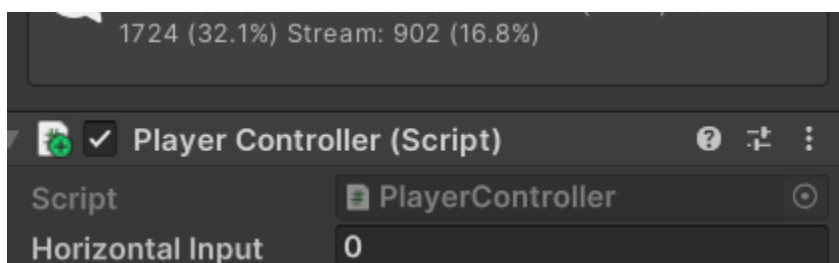
## Leccion 2 Objetos y escenario

*Vamos a colocar todos nuestros objetos en la Escena, incluyendo al jugador, los animales y la comida.*



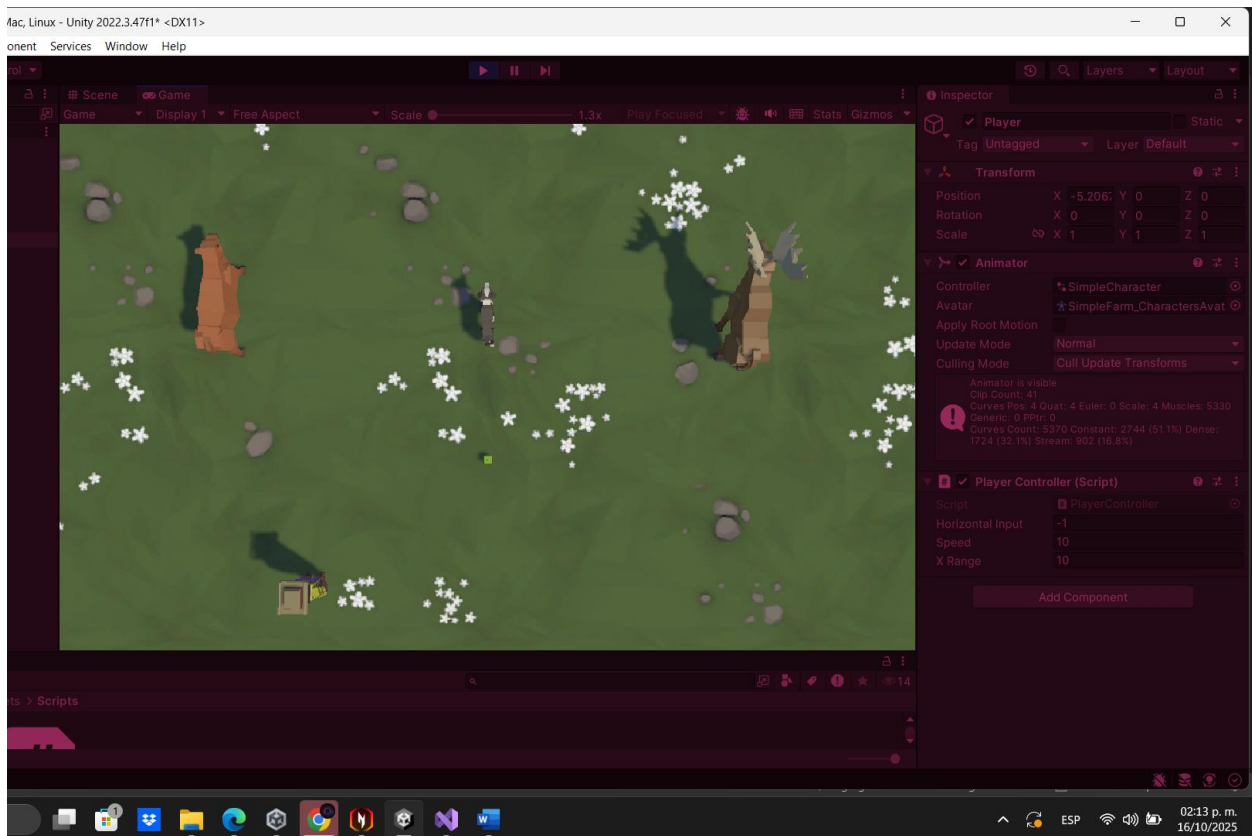
*Si queremos mover el jugador de izquierda a derecha, necesitamos una variable que siga la entrada del usuario.*

1. En la carpeta **Assets** crea una carpeta «Scripts» y un Script denominado «PlayerController» dentro.
2. **Adjunta** el Script al jugador para abrirlo.
3. En la parte superior de PlayerController.cs, define un nuevo **public float horizontalInput**.
4. En **Update()**, establece **horizontalInput = Input.GetAxis("Horizontal")**, y luego haz una prueba para asegurarte de que funciona en el Inspector.



Tenemos que utilizar la entrada horizontal para desplazar al jugador a la izquierda y a la derecha.

1. Define un nuevo **public float speed = 10.0f;**
2. En **Update()**, desplaza al jugador de lado a lado con base en **horizontalInput** y a **speed**.



(Captura de Player moviéndose con los controles).

Tenemos que evitar que el jugador se salga de la pantalla con un enunciado condicional.

1. En **Update()**, escribe un enunciado **condicional** para comprobar si la posición izquierda X del jugador es **menor que** un valor dado.
2. En el enunciado condicional, establece la posición del jugador a su posición actual, pero con una **ubicación X fija**.

```

void Update()
{
    if (transform.position.x < -xRange)
    {
        transform.position = new Vector3(-xRange, transform.position.y, transform.position.z);
    }

    if (transform.position.x > xRange)
    {
        transform.position = new Vector3(xRange, transform.position.y, transform.position.z);
    }

    horizontalInput = Input.GetAxis("Horizontal");
}

```

Tenemos que hacer que esto funcione también en el lado derecho, y luego limpiar nuestro código.

1. Repite este proceso para el **lado derecho** de la pantalla.
2. Define una nueva variable **xRange** y sustituye los valores codificados por estos.
3. Agrega **comentarios** a tu código.

```

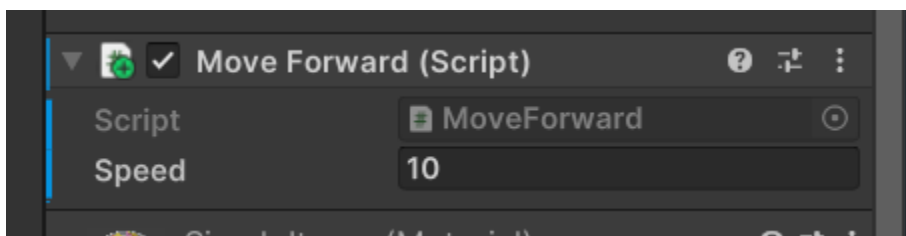
public float speed = 10.0f;
public float xRange = 10.0f;

```

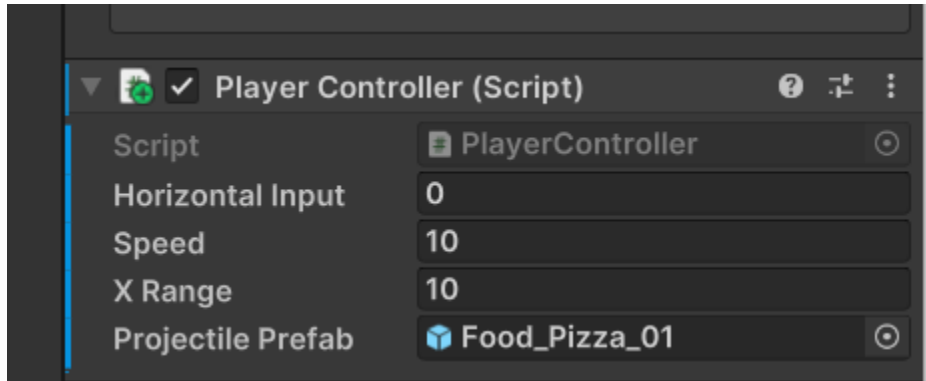
## Lección 2.2: Comida voladora.

Lo primero que debemos hacer es darle al proyectil algo de movimiento hacia adelante para que pueda recorrer la Escena cuando el jugador lo lance.

1. Crea un nuevo Script llamado «MoveForward», **adjúntalo** al objeto Food y ábrelo.
2. Define **public float speed** como una nueva variable;
3. En **Update()**, agrega **transform.Translate(Vector3.forward \* Time.deltaTime \* speed);**, luego **guarda**.
4. En el **Inspector**, establece la variable **speed** del proyectil, y luego haz una prueba.



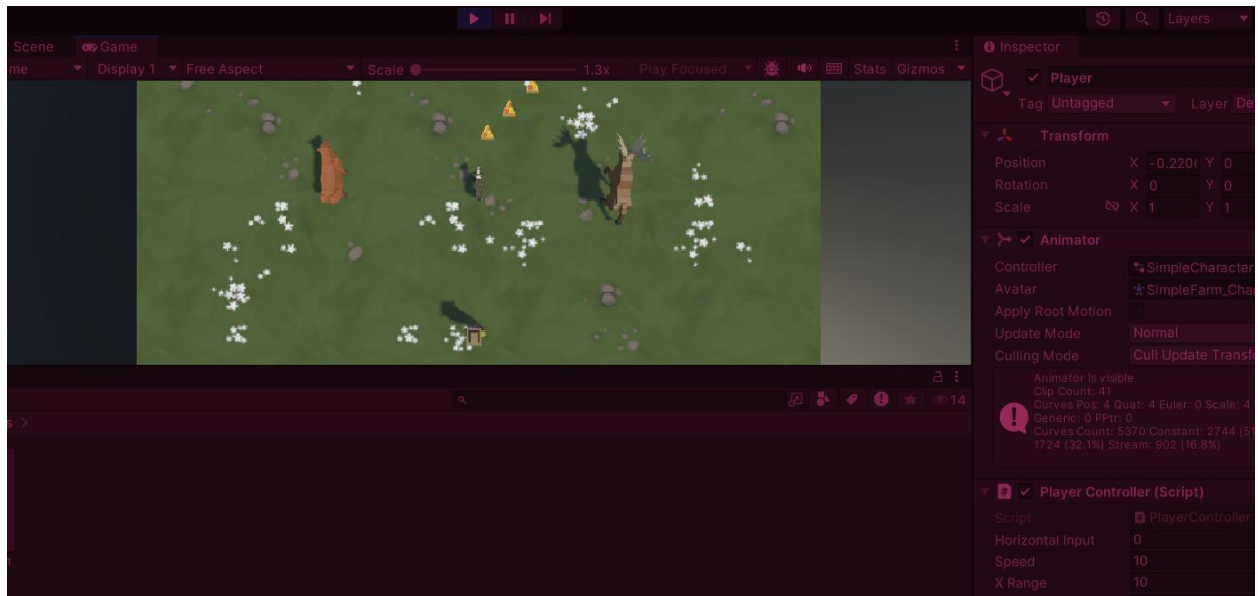
1. Crea una nueva carpeta llamada «Prefabs» y arrastra la comida y selecciona **Original Prefab**.
2. En PlayerController.cs, define ***public GameObject projectilePrefab;*** como una nueva variable.
3. **Selecciona** al jugador en Hierarchy y **arrastra** el objeto de la carpeta de Prefabs a **Projectile Prefab box** en el Inspector.
4. Haz una prueba **arrastrando** el proyectil a la Escena en tiempo de ejecución para comprobar que vuela.



1. En PlayerController.cs, en ***Update()***, agrega un **enunciado condicional** que compruebe si se pulsa la barra espaciadora: ***if (Input.GetKeyDown(KeyCode.Space)) {***
2. En el enunciado condicional, añade un comentario que diga ***// Launch a projectile from the player.***

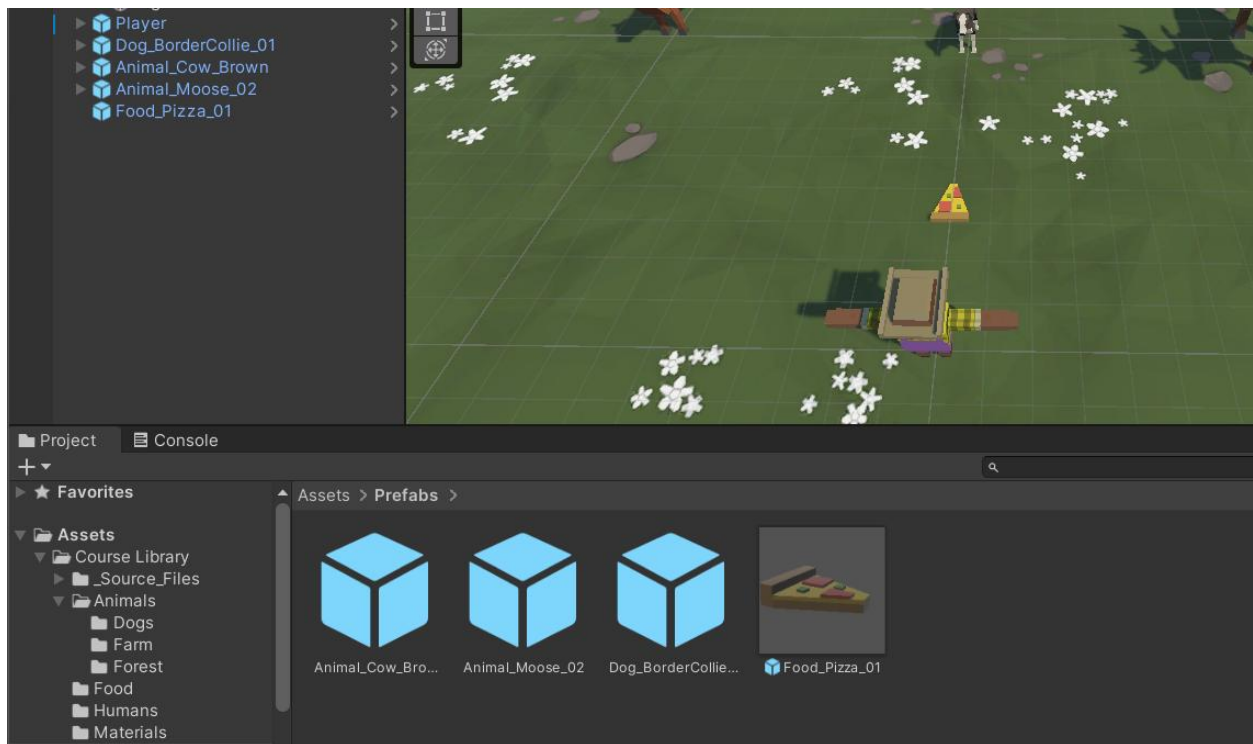
```
if (Input.GetKeyDown(KeyCode.Space))
{
    // Launch a projectile from the player
}
```

*Hemos creado el código que comprueba si el jugador pulsa la barra espaciadora, pero ahora tenemos que generar un proyectil cuando eso ocurra.*

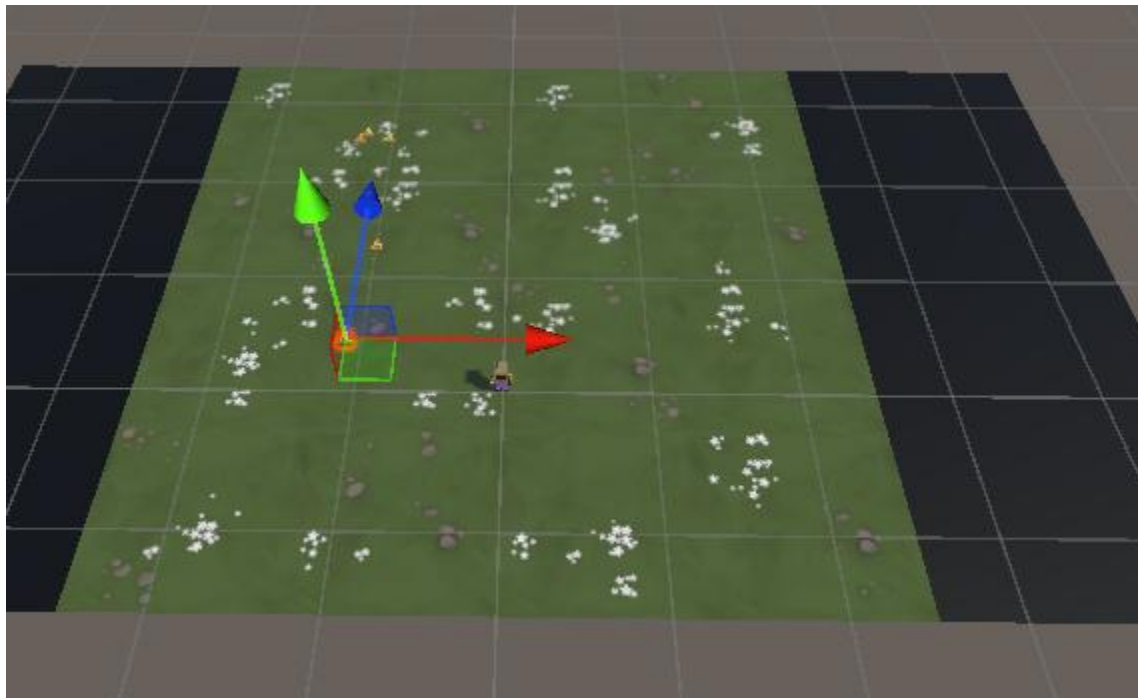


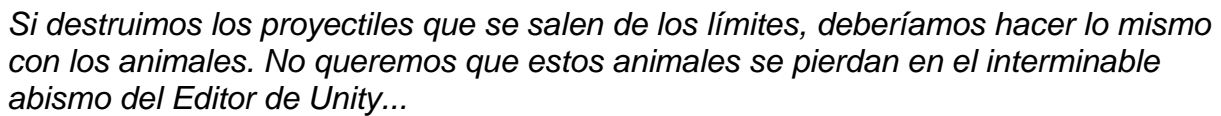
(captura lanzando proyectiles)

1. **Rota** todos los animales en el eje Y **180 grados** para que estén al final.
2. **Selecciona** los tres animales en Hierarchy y selecciona **Add Component > Move Forward**
3. Edita los valores **speed** y **haz una prueba** para revisar cómo se ve.
4. Arrastra los tres animales hacia la **carpeta Prefabs** y elige el Prefab original.
5. **Haz una prueba** arrastrando los Prefabs a la vista de Escena durante el juego.



*Cada vez que generamos un proyectil, este se desplaza más allá de la zona de juego hacia la eternidad. Para mejorar el rendimiento del juego, tenemos que destruirlos cuando salgan de los límites.*







1. Crea un **enunciado condicional else-if** para comprobar que los objetos estén debajo del **lowerBound**: **else if (transform.position.z > topBound)**.
2. **Aplica** el Script a todos los animales y usa **Override** en los Prefabs.



```

public class DestroyOutOfBounds : MonoBehaviour
{
    private float topBound = 30;
    private float lowerBound = -10;
    // 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 (transform.position.z > topBound)
        {
            Destroy(gameObject);
        } else if (transform.position.z < lowerBound)
        {
            Destroy(gameObject);
        }
    }
}

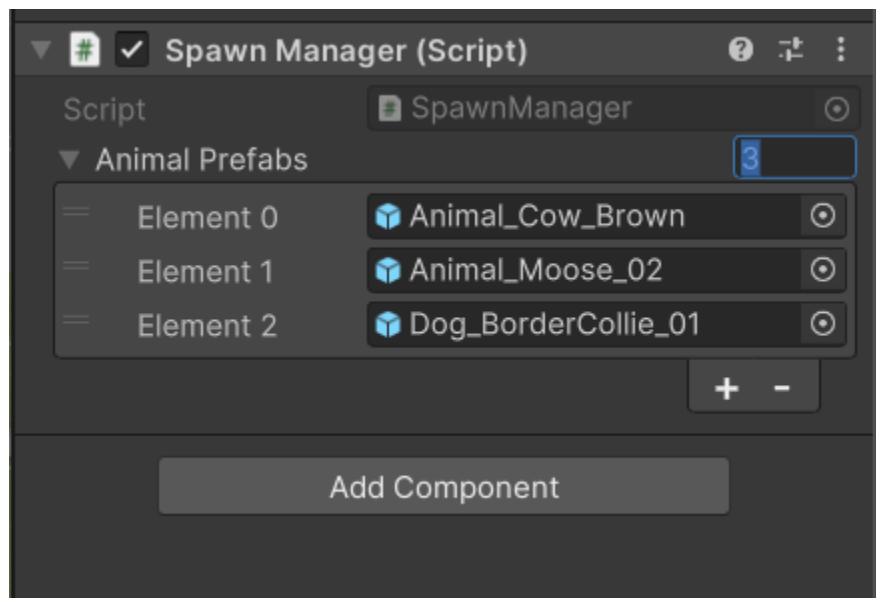
```

## Lección 2.3: Estampida de animales aleatorios

1. En Hierarchy, crea un **Empty object (Objeto vacío)** llamado «SpawnManager».
2. Crea un nuevo script llamado «SpawnManager», adjúntalo al **gestor de generación** y ábrelo.
3. Define ***public GameObject[] animalPrefabs;***
4. En el Inspector, cambia el **tamaño del arreglo** para que sea igual a la cantidad de animales, luego **asigna** los animales **arrastrándolos** desde la ventana Project hacia los espacios vacíos. **Nota:** asegúrate de arrastrarlos desde la **ventana Project**, no desde Hierarchy. Si vas a generar objetos, debes asegurarte de que estás usando Prefabs, que se guardan en la ventana Project.

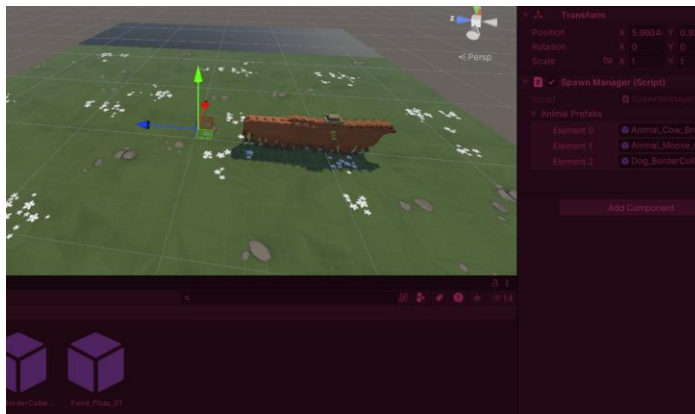
```
using System.Collections.Generic;
using UnityEngine;

Script de Unity | 0 referencias
public class SpawnManager : MonoBehaviour
{
    public GameObject[] animalPrefabs;
    // 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()
    {
    }
```



## 2. Genera un animal cuando se pulsa la tecla S

Hemos creado un arreglo y le hemos asignado nuestros animales, pero eso no sirve de mucho hasta que tengamos una forma de generarlos durante el juego. Vamos a crear una solución temporal para elegir y generar los animales.



Mostrar transcripción

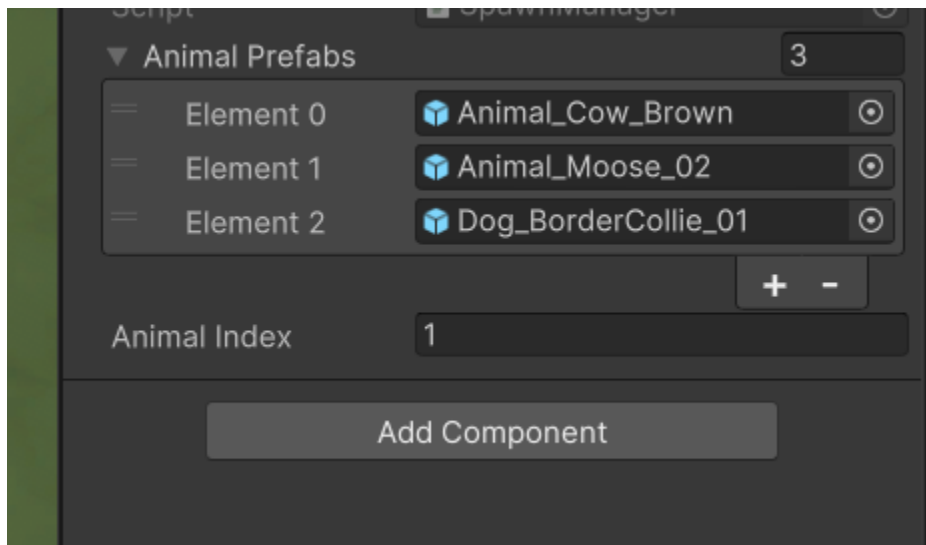
1. En **Update()**, escribe una sentencia if para **posicionar** un nuevo animal Prefab en la parte superior cuando la tecla **S** se pulsa.
2. Define una nueva variable **public int animalIndex** e incorpórala en la llamada. **Coloca una instancia** y haz una prueba editando el valor en el Inspector.

```
// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    if (Input.GetKeyDown(KeyCode.S))
    {
        Instantiate(animalPrefabs[0], new Vector3(0, 0, 20), animalPrefabs[0].transform.rotation);
    }
}
```

1. En **Update()**, escribe una sentencia if para **posicionar** un nuevo animal Prefab en la parte superior cuando la tecla **S** se pulsa.
2. Define una nueva variable **public int animalIndex** e incorpórala en la llamada. **Coloca una instancia** y haz una prueba editando el valor en el Inspector.

```
Script de Unity | 0 referencias
public class SpawnManager : MonoBehaviour
{
    public GameObject[] animalPrefabs;
    public int animalIndex;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Start()
    {

```



### 3. Generar animales aleatorios desde el arreglo

Podemos generar animales pulsando la tecla S, pero al hacerlo solo se genera un animal dentro del índice del arreglo que especifiquemos. Tenemos que aleatorizar la selección para que la tecla S pueda generar un animal aleatorio basado en el índice, sin nuestra especificación.

1. En la sentencia if que comprueba si se pulsa S, genera un int aleatorio en **animalIndex** entre 0 y la longitud del arreglo.
2. Elimina la variable global **animalIndex**, ya que solo se necesita localmente en la **sentencia if**.

```
// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    if (Input.GetKeyDown(KeyCode.S))
    {
        int animalIndex = Random.Range(0, 3);

        Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20), anima
    }
}
```

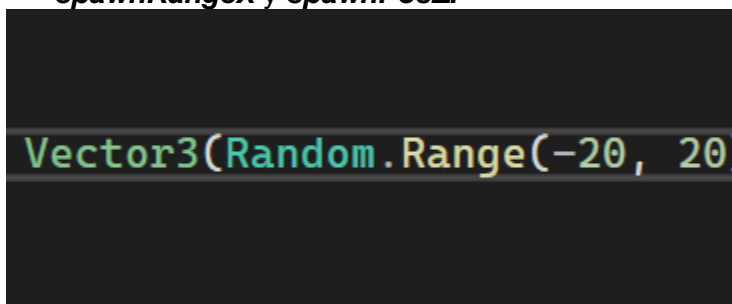


(Aparecen animales de manera aleatoria)

#### 4. Establece al azar la ubicación de la generación.

*Podemos pulsar S para generar animales aleatorios desde animalIndex, pero todos aparecen en el mismo lugar. Tenemos que aleatorizar su ubicación de generación, para que no avancen por la pantalla en línea recta.*

1. **Reemplaza** el valor de X para Vector3 por ***Random.Range(-20, 20)***, luego haz una prueba.
2. Dentro de la **sentencia if**, crea **Vector3 spawnPos** como una variable local.
3. En la parte superior de la clase, crea una nueva variable **privada float** para **spawnRangeX** y **spawnPosZ**.





## 5. Cambia la perspectiva de la cámara

Nuestro *Spawn Manager* está progresando muy bien, así que vamos a tomarnos un descanso y a jugar con la cámara. Cambiar la perspectiva de la cámara puede ofrecer una visión más adecuada para este juego de vista superior.

1. Alterna entre **Perspective** e **Isometric** en Scene View (Escena) para apreciar la diferencia
2. Selecciona la **cámara** y cambia **Projection** de «Perspective» a «Orthographic».



## Lección 2.4: Decisiones de colisiones

### 1. Haz un nuevo método para generar animales

Nuestro gestor de generación tiene buena pinta, pero todavía tenemos que pulsar **S** para que funcione. Si queremos que el juego genere animales automáticamente, tenemos que empezar por escribir nuestra primera función personalizada.

1. En **SpawnManager.cs**, crea la función **void SpawnRandomAnimal() {}** debajo de **Update()**
2. Corta y pega el código dentro de la **sentencia if** a la **nueva función**
3. Llama **SpawnRandomAnimal()**; si la tecla **S** se pulsa

```

Mensaje de Unity | 0 referencias
void Update()
{
    if (Input.GetKeyDown(KeyCode.S))
    {
        SpawnRandomAnimal();
    }
}

1 referencia
void SpawnRandomAnimal()
{
    int animalIndex = Random.Range(0, animalPrefabs.Length);
    Vector3 spawnPos = new Vector3(Random.Range(-spawnRangeX, spawnRangeX), 0, spawnPosZ);

    Instantiate(animalPrefabs[animalIndex], spawnPos, animalPrefabs[animalIndex].transform.rotation);
}

```

## 2. Genera animales en intervalos de tiempo

Guardamos el código de generación en una función personalizada, pero aún tenemos que pulsar la tecla S. Tenemos que generar los animales en intervalos, para que aparezcan aleatoriamente cada pocos segundos.

1. En **Start()**, usa **InvokeRepeating** para generar animales en función de un intervalo de tiempo, luego **haz una prueba**.
2. Elimina la **sentencia if** que comprueba si la tecla **S** se pulsó
3. Define **private startDelay** y **spawnInterval** como nuevas variables y haz una prueba en el juego para corregir los valores de las variables

```

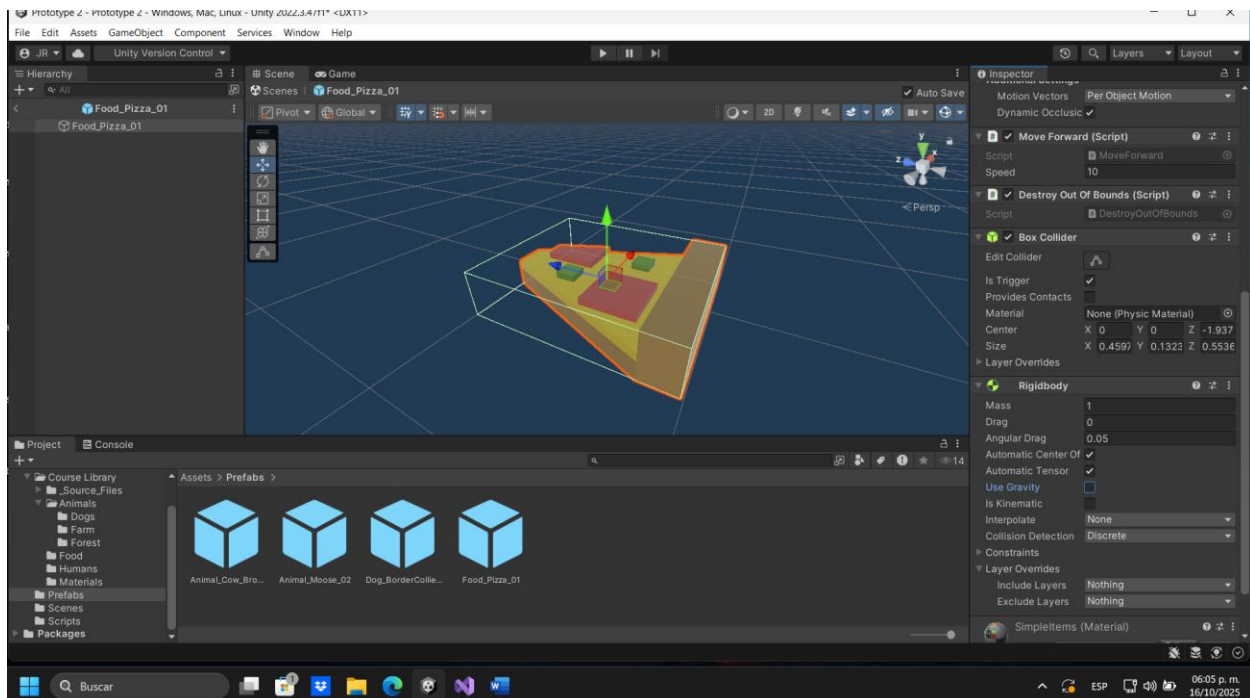
// Start is called before the first frame update
Mensaje de Unity | 0 referencias
void Start()
{
    InvokeRepeating("SpawnRandomAnimal", 2, 1.5f);
}

```

## 3. Agrega un componente Collider y Trigger

Los animales se generan perfectamente y el jugador puede dispararles proyectiles, pero no ocurre nada cuando colisionan. Si queremos que los proyectiles y los animales se destruyan al colisionar, tenemos que darles unos componentes conocidos: los «Colliders».

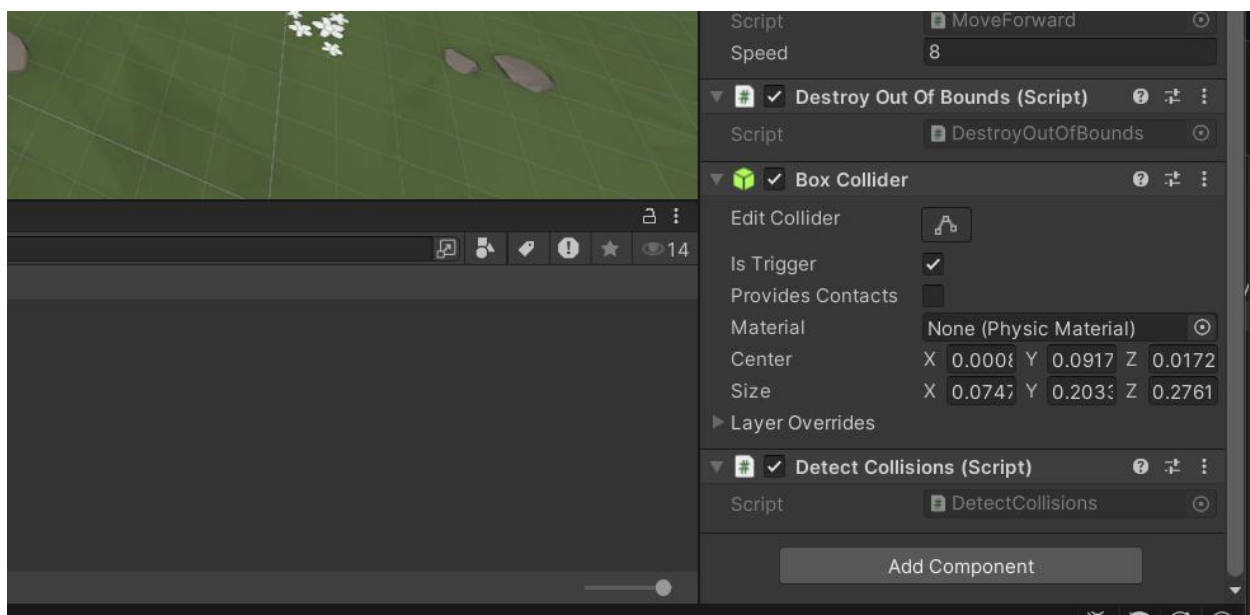
1. Haz doble clic en uno de los **animales Prefabs**, y **Add Component** (Agregar componente) > **Box Collider**.
2. Haz clic en **Edit Collider (Editar Collider)**, luego **arrastra** las manillas del colisionador para que abarquen el objeto.
3. Marca la casilla «**Is Trigger**».
4. Repite este proceso en cada uno de los **animales** y en el **proyectil**.
5. Agrega un **Rigidbody component** al proyectil y desmarca la casilla Use Gravity (Usar gravedad).



#### 4. Destruye objetos al colisionar.

Ahora que los animales y el proyectil tienen Box Colliders con triggers, necesitamos codificar un nuevo script para destruirlos al impactar.

1. Crea un nuevo script llamado **DetectCollisions.cs** y agrégalo a cada animal Prefab, luego ábrelo
2. Antes del final `}` agrega la función **OnTriggerEnter** usando **autocompletar**
3. En **OnTriggerEnter**, escribe **Destroy(gameObject);**, luego haz un prueba
4. En **OnTriggerEnter**, escribe **Destroy(other.gameObject);**





## 5.Desencadena un mensaje de «Game Over»

*El jugador puede defender su campo contra los animales por el tiempo que quiera, pero tenemos que avisarle que ha perdido con un mensaje de «Game Over» si algún animal esquiva al jugador.*

1. En DestroyOutOfBounds.cs, en la **condicional else if** que comprueba si los animales alcanzaron el fondo de la pantalla, agrega un mensaje de «Game Over»: ***Debug.Log("Game Over!")***.
2. Limpia tu código con **comentarios**.
3. Si usas Visual Studio, haz clic en *Edit > Advanced > Format document* para corregir los problemas de sangría. (En **Mac**, haz clic en *Edit > Format > Format Document*).

