Andrés Donaciano Martínez Guillén

POSICIONAMIENTO DEL JUGADOR

1. Crea un nuevo proyecto para el Prototipo 2

Lo primero que debemos hacer es crear un nuevo proyecto e importar los archivos de inicio del Prototipo 2.

1. Abra Unity Hub y cree un proyecto vacío " Prototipo 2 " en el directorio de su curso con la versión correcta de Unity.
2. Si no recuerda cómo hacerlo, consulte las instrucciones de la Lección 1.1 - Paso 1.
3. Haga clic para descargar los archivos de inicio del Prototipo 2 , extraiga la carpeta comprimida e importe el paquete .unity a su proyecto.
4. Si no recuerda cómo hacerlo, consulte las instrucciones de la Lección 1.1 - Paso 2.
5. Desde la ventana Proyecto, abra la escena Prototipo 2 y elimine SampleScene
6. En la parte superior derecha del Editor de Unity, cambie su Diseño de Predeterminado a su diseño personalizado

2. Agrega el jugador, los animales y la comida.

Coloquemos todos nuestros objetos en la escena, incluido el jugador, los animales y la comida.

1. Si lo desea, arrastre un material diferente desde Biblioteca del curso > Materiales hasta el objeto Tierra
2. Arrastre 1 objeto humano , 3 animales y 1 objeto de comida a la jerarquía
3. Cambia el nombre del personaje a “ Jugador ”, luego reposiciona los animales y la comida para que puedas verlos.
4. Ajuste la escala XYZ de la comida para que pueda verla fácilmente desde arriba

3. Configure los ajustes del proyecto para el Administrador de entrada

El código que agregarás en los siguientes pasos utiliza un sistema llamado Input Manager para gestionar los controles del reproductor. Si usas Unity 6.1 o una versión posterior, es probable que tu proyecto de Unity esté configurado para usar un sistema diferente llamado Input System . Para garantizar que el código de Input Manager funcione correctamente, debes cambiar una configuración del proyecto para que ambos sistemas se ejecuten simultáneamente.

Siga el vídeo y las instrucciones a continuación.

Instrucciones
1. Abra la configuración del **Player** :
Desde el menú principal, seleccione **Editar > Configuración del proyecto**, luego seleccione la categoría **Player** en el panel más a la izquierda.

2. Habilite la compatibilidad con ambos sistemas de entrada:
Encuentra la sección **Configuración** .
En el menú desplegable Manejo de entrada activa, seleccione **BOTH**.
Seleccione el botón Aplicar para confirmar el cambio.
El Editor de Unity se reiniciará automáticamente para aplicar esta configuración. Tras reiniciarse, el proyecto estará configurado correctamente para gestionar la entrada tanto del Gestor de Entrada original como del nuevo Sistema de Entrada.

4. Obtener la entrada horizontal del usuario

Si queremos mover el reproductor de izquierda a derecha, necesitamos una variable que rastree la entrada del usuario.

1. En su carpeta Activos , cree una carpeta “ Scripts ” y un script “ PlayerController ” dentro
2. Adjunte el script al reproductor y ábralo
3. En la parte superior de PlayerController.cs, declare un nuevo float horizontalInput público
4. En Update() , configure horizontalInput = Input.GetAxis(“Horizontal”) y luego pruebe para asegurarse de que funcione en el inspector.

```
public float horizontalInput;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
}
```

5. Mueve al jugador de izquierda a derecha

En realidad, tenemos que utilizar la entrada horizontal para trasladar al jugador de izquierda a derecha.

1. Declarar un nuevo flotante público speed = 10.0f;
2. En Update() , traslada al jugador de lado a lado según la entrada horizontal y la velocidad.

```
public float horizontalInput;
public float speed = 10.0f;

void Update()
{
  horizontalInput = Input.GetAxis("Horizontal");
  transform.Translate(Vector3.right * horizontalInput * Time.deltaTime * speed);
}
```

6. Mantener al jugador dentro de los límites del campo

Tenemos que evitar que el jugador se salga de la pantalla con una declaración si-entonces.

1. En Update() , escriba una declaración if que verifique si la posición X izquierda del jugador es menor que un valor determinado
2. En la declaración if, establezca la posición del jugador en su posición actual, pero con una ubicación X fija

```
void Update() {
  if (transform.position.x < -10) {
    transform.position = new Vector3(-10, transform.position.y, transform.position.z);
  }
}
```

7. Limpia tu código y variables

Necesitamos hacer que esto funcione también en el lado correcto y luego limpiar nuestro código.

1. Repita este proceso para el lado derecho de la pantalla.
2. Declare la nueva variable xRange y luego reemplace los valores codificados con ellos
3. Añade comentarios a tu código

```
public float xRange = 10;

void Update()
{
  // Keep the player in bounds
  if (transform.position.x < -10 -xRange)
  {
    transform.position = new Vector3(-10 -xRange, transform.position.y, transform.position.z);
  }
  if (transform.position.x > xRange)
  {
    transform.position = new Vector3(xRange, transform.position.y, transform.position.z);
  }
}
```

2 – VUELO DE COMIDA

1. Haz que el proyectil vuele hacia adelante.

Lo primero que debemos hacer es darle al proyectil un poco de movimiento hacia adelante para que pueda atravesar la escena cuando el jugador lo lance.

1. Crea un nuevo script “ MoveForward ”, adjúntalo al objeto de comida y luego ábrelo
2. Declarar una nueva variable de velocidad flotante pública ;
3. En Update() , agregue transform.Translate(Vector3.forward * Time.deltaTime * speed); y luego guarde
4. En el Inspector , configure la variable de velocidad del proyectil y luego pruebe

```
public float speed = 40.0f;

void Update() {
  transform.Translate(Vector3.forward * Time.deltaTime * speed);
}
```

2. Convierte el proyectil en un prefab.

Ahora que nuestro proyectil tiene el comportamiento que queremos, necesitamos convertirlo en un prefab para que pueda reutilizarse en cualquier lugar y en cualquier momento, con todos sus comportamientos incluidos.

1. Crea una nueva carpeta “ Prefabs ”, arrastra tu comida a ella y elige Prefab original.
2. En PlayerController.cs, declare un nuevo GameObject público projectilePrefab; variable
3. Seleccione el Jugador en la jerarquía, luego arrastre el objeto desde su carpeta Prefabs al nuevo cuadro Prefab Proyectil en el inspector
4. Intente arrastrar el proyectil a la escena en tiempo de ejecución para asegurarse de que vuele.

3. Prueba de pulsación de la barra espaciadora

Ahora que tenemos un prefabricado de proyectil asignado a PlayerController.cs, el jugador necesita una forma de lanzarlo con la barra espaciadora.

1. En PlayerController.cs, en Update() , agregue una declaración if que verifique si se presiona la barra espaciadora:
2. if (Input.GetKeyDown(KeyCode.Space)) {
3. Dentro de la declaración if, agregue un comentario que diga que debe // Lanzar un proyectil desde el jugador

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

4. Lanzar proyectil al presionar la barra espaciadora

Hemos creado el código que prueba si el jugador presiona la barra espaciadora, pero ahora necesitamos generar un proyectil cuando eso sucede.

1. Dentro de la declaración if, use el método Instantiate para generar un proyectil en la ubicación del jugador con la rotación del prefabricado.

```
if (Input.GetKeyDown(KeyCode.Space))
{
  // Launch a projectile from the player
  Instantiate(projectilePrefab, transform.position, projectilePrefab.transform.rotation);
}
```

5. Convierte animales en prefabricados

El proyectil ahora es un prefabricado, pero ¿y los animales? También deben ser prefabricados para poder instanciarlos durante el juego.

2. Gire todos los animales en el eje Y 180 grados para que queden boca abajo.
3. Seleccione los tres animales en la jerarquía y Agregar componente > **MoveForward**
4. Edite sus valores de velocidad y pruebe para ver cómo se ve.
5. Arrastre los tres animales a la carpeta Prefabs y elija “Prefab original”.
6. Prueba arrastrando prefabricados a la vista de escena durante el juego

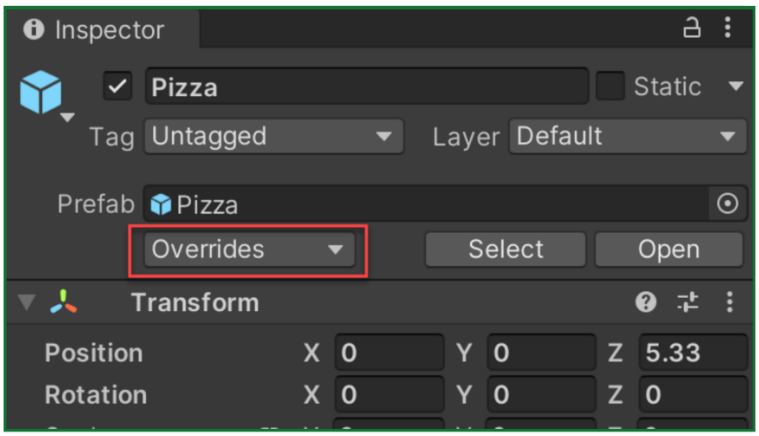
6. Destruye proyectiles fuera de la pantalla

Cada vez que generamos un proyectil, este se aleja del área de juego y se pierde en la eternidad. Para mejorar el rendimiento del juego, debemos destruirlos cuando salen de los límites.

1. Crea el script “ DestroyOutOfBounds ” y aplícalo al proyectil
2. Agregue una nueva variable flotante privada topBound e inicialícela = 30;
3. Escribe el código para destruir si está fuera de los límites superiores if (transform.position.z > topBound) {
4. Destroy(gameObject); }
5. En el menú desplegable Inspector **Overrides** , haga clic en Aplicar todo para aplicarlo al prefab

```
private float topBound = 30;

void Update() {
  if (transform.position.z > topBound) {
    Destroy(gameObject);  }}
```



7. Destruir animales fuera de la pantalla

Si destruimos los proyectiles que salen de los límites, probablemente deberíamos hacer lo mismo con los animales. No queremos que se pierdan en el abismo infinito del Editor de Unity...

1. Cree una declaración else-if para verificar si los objetos están por debajo de lowerBound :
   else if (transform.position.z < lowerBound)
2. Aplique el script a todos los animales y luego anule los prefabricados.

```
private float topBound = 30;
private float lowerBound = -10;

void Update() {
  if (transform.position.z > topBound)
  {
    Destroy(gameObject);
  } else if (transform.position.z < lowerBound) {
    Destroy(gameObject);
  } }
```

Nota: siempre que modifique un GO desde el panel Hierarchy, tiene que usar OVERRIDE para actualizar todas las instancias del PREFAB.

3 - ESTAMPIDA ALEATORIA DE ANIMALES

1. Crea un administrador de spawn

Si vamos a realizar toda esta compleja generación de objetos, deberíamos tener un script dedicado a administrar el proceso, así como un objeto al cual adjuntarlo.

1. En la jerarquía, cree un objeto vacío llamado “ **SpawnManager** ”
2. Cree un nuevo script llamado “ **SpawnManager** ”, adjúntelo al **SpawnManager** y ábralo.

3. Declarar nuevo GameObject[ ] público animalPrefabs;
4. En el Inspector, modifica el tamaño del Array para que coincida con el número de animales y, luego, asígnalos arrastrándolos desde la ventana del Proyecto a las ranuras vacías. Nota: Asegúrate de arrastrarlos desde la ventana del Proyecto , no desde la Jerarquía. Si vas a generar objetos, asegúrate de usar Prefabs, que se almacenan en la ventana del Proyecto.

2. Genera un animal si se presiona S

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

1. En Update() , escriba una declaración if-then para crear una instancia de un nuevo prefab de animal en la parte superior de la pantalla si se presiona S
2. Declare un nuevo int público animalIndex e incorpórelo en la llamada Instantiate , luego pruebe editar el valor en el Inspector

```
public GameObject[] animalPrefabs;
public int animalIndex;

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20),
    animalPrefabs[animalIndex].transform.rotation);
  }
}
```

3. Generar animales aleatorios a partir de una matriz

Podemos generar animales presionando S, pero al hacerlo solo se genera un animal en el índice de la matriz que especifiquemos. Necesitamos aleatorizar la selección para que S pueda generar un animal aleatorio según el índice, sin nuestra especificación.

1. En la instrucción if que verifica si se presiona S, genere un int animalIndex aleatorio entre 0 y la longitud de la matriz
2. Elimine la variable global animalIndex , ya que solo se necesita localmente en la declaración if

```
public GameObject[] animalPrefabs;
public int animalIndex;

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    int animalIndex = Random.Range(0, animalPrefabs.Length);
    Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20),
            animalPrefabs[animalIndex].transform.rotation); }}
```

4. Aleatoriza la ubicación de aparición

Podemos presionar S para generar animales aleatorios de animalIndex, ¡pero todos aparecen en el mismo lugar! Necesitamos aleatorizar su posición de aparición para que no avancen por la pantalla en línea recta.

1. Cree la variables SpawnRangeX , spawnPosZ de tipo float y privadas con en valor de 20.
2. Reemplace el valor X para Vector3 con Random.Range(-20, 20) , luego pruebe
3. Dentro de la declaración if , cree una nueva variable local Vector3 spawnPos
4. En la parte superior de la clase, cree variables flotantes privadas para spawnRangeX y spawnPosZ

```
private float spawnRangeX = 20;
private float spawnPosZ = 20;

void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    // Randomly generate animal index and spawn position
    Vector3 spawnPos = new Vector3(Random.Range(-spawnRangeX, spawnRangeX),
    0, spawnPosZ);
    int animalIndex = Random.Range(0, animalPrefabs.Length);
    Instantiate(animalPrefabs[animalIndex], spawnPos,
    animalPrefabs[animalIndex].transform.rotation); }}
```

5. Cambiar la perspectiva de la cámara

Nuestro Spawn Manager está avanzando bien, así que tomémonos un descanso y juguemos con la cámara. Cambiar la perspectiva de la cámara podría ofrecer una vista más apropiada para este juego de arriba hacia abajo.

1. Alterne entre la vista en perspectiva y la vista isométrica en la vista de escena para apreciar la diferencia
2. Seleccione la cámara y cambie la Proyección de “Perspectiva” a “Ortográfica”

4 - DECISIONES DE COLISION

1. Crea un nuevo método para generar animales.

Nuestro Gestor de Aparición se ve bien, ¡pero seguimos presionando S para que funcione! Si queremos que el juego genere animales automáticamente, debemos empezar por escribir nuestra primera función personalizada.

1. En SpawnManager.cs , cree una nueva función void SpawnRandomAnimal() {} debajo de Update()
2. Cortar y pegar el código de la declaración if-then a la nueva función
3. Llamar a SpawnRandomAnimal(); si se presiona S

```
void Update() {
  if (Input.GetKeyDown(KeyCode.S)) {
    SpawnRandomAnimal();
    int animalIndex … (Cut and Pasted Below) }}

void SpawnRandomAnimal() {
  int animalIndex = Random.Range(0, animalPrefabs.Length);
  Vector3 spawnpos = new Vector3(Random.Range(-xSpawnRange, xSpawnRange), 0, zSpawnPos);
  Instantiate(animalPrefabs[animalIndex], new Vector3(0, 0, 20) spawnpos,
    animalPrefabs[animalIndex].transform.rotation);
 }
```

2. Desovar los animales a intervalos cronometrados

Hemos guardado el código de generación en una función personalizada, ¡pero seguimos presionando S! Necesitamos generar los animales con un temporizador para que aparezcan aleatoriamente cada pocos segundos.

1. Crear variables startDelay=2 como float y spawnInterval=1.5 como float también.
2. En Start() , use InvokeRepeating para generar los animales en función de un intervalo y luego pruebe .
3. Eliminar la declaración if-then que prueba si se presiona S
4. Declare nuevas variables privadas startDelay y spawnInterval , luego pruebe y ajuste los valores de las variables.

```
private float startDelay = 2;
private float spawnInterval = 1.5f;

void Start() {
  InvokeRepeating("SpawnRandomAnimal", startDelay, spawnInterval); }

void Update() {
if (Input.GetKeyDown(KeyCode.S)) {
    SpawnRandomAnimal(); } }
```

3. Agregar componentes de colisionador y disparador

Los animales se generan a la perfección y el jugador puede dispararles proyectiles, ¡pero no ocurre nada si chocan! Si queremos que los proyectiles y los animales se destruyan al colisionar, necesitamos darles componentes conocidos: los "colisionadores".

1. Haga doble clic en uno de los prefabricados de animales y luego en Agregar componente > Box Collider .
2. Asegúrese de que la opción “Guardar automáticamente” esté habilitada en la esquina superior derecha de la vista de escena .
3. Haga clic en Editar Collider y luego arrastre los controladores del collider para abarcar el objeto.
4. Marque la casilla de verificación “ **Is trigger**”.
5. Utilice el menú desplegable **Overrides** para aplicar este disparador a todos los prefabricados de este animal.

6. Repita este proceso para cada uno de los animales y el proyectil .
7. Agregue un componente RigidBody al proyectil y desmarque "**Use gravity**"

4. Destruir objetos en caso de colisión.

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

1. Cree un nuevo script DetectCollisions.cs , agréguelo a cada prefabricado de animal y luego ábralo
2. Antes del } final agregue la función OnTriggerEnter usando autocompletar
3. En OnTriggerEnter , coloque Destroy(gameObject) ;, luego pruebe
4. En OnTriggerEnter , coloque Destroy(other.gameObject);

```
void OnTriggerEnter(Collider other) {
  Destroy(gameObject);
  Destroy(other.gameObject); }
```

5. Activar un mensaje de "Game Over"

El jugador puede defender su campo contra los animales durante el tiempo que desee, pero debemos avisarle cuando haya perdido con un mensaje de "Juego terminado" si algún animal logra superar al jugador.

1. En DestroyOutOfBounds.cs, en la condición else-if que verifica si los animales llegan al final de la pantalla, agregue un mensaje de Juego terminado: Debug.Log("Game Over!")
2. Limpia tu código con comentarios
3. Si usa Visual Studio, haga clic en Editar > Avanzado > Formato de documento para corregir cualquier problema de sangría
   (en una Mac , haga clic en Editar > Formato > Formato de documento)

```
void Update() {
  if (transform.position.z > topBound)
  {
    Destroy(gameObject);
  } else if (transform.position.z < lowerBound)
  {
    Debug.Log("Game Over!");
    Destroy(gameObject);
  }
}
```