Guía de Laboratorio 7

Primera parte
Recolección de Recursos en Unity (MoneyBag, Meat y Wood)

Paso 1: Ajuste del recurso gráfico

1. Dirígete a la carpeta:
   Assets/Images/TinySwords/Resources/Resources
2. Selecciona el sprite correspondiente al objeto **MoneyBag** (por ejemplo, G_Idle_0).
3. En el **Inspector**, localiza la propiedad **Pixels Per Unit (PPU)** y cambia su valor de **100** (por defecto) a **64**.
4. Esto hará que la escala del sprite en el escenario sea coherente con los demás objetos del entorno.

Paso 2: Incorporar el objeto al escenario

1. Arrastra el sprite **G_Idle_0** desde el *Project* hasta la *Scene*.
2. En la **Jerarquía (Hierarchy)**, selecciona el objeto recién creado.
3. Cámbiale el nombre a **MoneyBag** para mantener un identificador claro y coherente.

Paso 3: Configurar colisiones

1. Con el objeto **MoneyBag** seleccionado, haz clic en **Add Component**.
2. Añade un componente del tipo **Box Collider 2D**.
3. Marca la casilla **Is Trigger**, lo que permitirá detectar colisiones sin bloquear físicamente el movimiento del jugador.
   Esto es importante porque el jugador debe poder atravesar el objeto mientras lo "recoge".

Paso 4: Crear y asignar la etiqueta (Tag)

1. En el menú superior, ve a **Edit → Project Settings → Tags and Layers**.
2. En la sección **Tags**, haz clic en **+** para añadir una nueva etiqueta.
3. Escribe **MoneyBag** como nombre del nuevo Tag.
4. Selecciona el objeto **MoneyBag** en la Jerarquía y asígnale la etiqueta recién creada desde el desplegable **Tag**.

Paso 5: Organización del proyecto

1. En la carpeta raíz del proyecto, dirígete a **Assets/Scripts**.
2. Crea dentro de ella una nueva carpeta llamada **Player** para mantener los scripts del jugador organizados.

Paso 6: Creación del script

1. Dentro de la carpeta **Assets/Scripts/Player**, crea un nuevo **MonoBehaviour Script** llamado **PlayerResourcesCollector**.
2. Abre el script haciendo doble clic (se abrirá en Visual Studio Code u otro editor configurado).

Paso 7: Programación del recolector de recursos

Sustituye el contenido del archivo por el siguiente código:

```
using UnityEngine;

public class PlayerResourcesCollector : MonoBehaviour
{
    private int money = 0;
    private int meat = 0;
    private int wood = 0;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("MoneyBag"))
        {
            Destroy(collision.gameObject);
            money++;
            Debug.Log("Tenemos " + money + " de dinero");
        }
    }
}
```

Explicación del código:

- **money, meat, wood** → variables privadas que almacenan la cantidad recolectada de cada recurso.
- **OnTriggerEnter2D()** → método que se ejecuta automáticamente cuando el jugador entra en contacto con un objeto marcado como *trigger*.
- **CompareTag("MoneyBag")** → verifica si el objeto con el que colisiona tiene la etiqueta "MoneyBag".
- **Destroy(collision.gameObject)** → elimina el objeto del escenario (simulando su recogida).
- **Debug.Log()** → muestra en la consola el número de recursos acumulados.

Paso 8: Asignación del script al jugador

1. En la **Jerarquía**, selecciona el objeto del jugador (por ejemplo, **Player**).
2. Arrastra el script **PlayerResourcesCollector** desde el *Project* al *Inspector* del jugador o utiliza el botón **Add Component → Scripts → PlayerResourcesCollector**.
   Esto vinculará el comportamiento de recolección al jugador.

Paso 9: Extensión a otros recursos

1. Repite el procedimiento anterior para crear los objetos **Meat** y **Wood**:
    - Agrega los sprites correspondientes al escenario.
    - Ajusta el **Pixels Per Unit** a 64.
    - Añade un **Box Collider 2D** con **Is Trigger** marcado.
    - Crea y asigna los Tags **Meat** y **Wood** respectivamente.
2. Actualiza el script del jugador con el siguiente código completo:

```
using UnityEngine;

public class PlayerResourcesCollector : MonoBehaviour
{
    private int money = 0;
    private int meat = 0;
    private int wood = 0;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("MoneyBag"))
        {
            Destroy(collision.gameObject);
            money++;
            Debug.Log("Tenemos " + money + " de dinero");
        }
        if (collision.gameObject.CompareTag("Meat"))
        {
            Destroy(collision.gameObject);
            meat++;
            Debug.Log("Tenemos " + meat + " de comida");
        }
        if (collision.gameObject.CompareTag("Wood"))
        {
            Destroy(collision.gameObject);
            wood++;
            Debug.Log("Tenemos " + wood + " de madera");
        }
    }
}
```

Funcionamiento:

- Cada bloque **if** detecta un tipo de recurso distinto.
- El objeto correspondiente se destruye tras la colisión.
- Se incrementa el contador del recurso.
- Se muestra en consola el total acumulado.

Paso 10: Prueba en el juego

1. Coloca los objetos **MoneyBag**, **Meat** y **Wood** en distintas posiciones dentro de la escena.
2. Asegúrate de que el jugador tenga un **Collider 2D** y un **Rigidbody2D** (sin marcar "Is Kinematic").
3. Ejecuta el juego (**Play**) y mueve el jugador hacia los objetos.
4. Observa en la **Consola** cómo aparecen los mensajes de recolección al tocar cada recurso.

Segunda Parte
Creación de Animaciones de Recursos en Unity

Paso 1: Ajustar la escala de los sprites

1. Dirígete a la carpeta:
   Assets/Images/TinySwords/Resources/Resources
2. Localiza los archivos de sprites:
   - G_Spawn (para MoneyBag)
   - M_Spawn (para Meat)
   - W_Spawn (para Wood)
3. Selecciona cada uno de estos sprites en el *Project*.
4. En el **Inspector**, busca la propiedad **Pixels Per Unit (PPU)** y cambia su valor a **64**.
   Esto asegura que las animaciones tengan una escala coherente con el resto de los elementos del juego.

Paso 2: Dividir los sprites en cuadros de animación

1. Con cada recurso (G_Spawn, M_Spawn, W_Spawn) seleccionado, haz clic en el botón **Sprite Editor** (si aparece un mensaje para aplicar cambios, presiona **Apply**).
2. Dentro del **Sprite Editor**, en la parte superior, selecciona la opción **Slice**.
3. Cambia el modo de corte a **Grid by Cell Count**.
4. Configura los valores de:
   - **Columnas:** 7
   - **Filas:** 1
5. Presiona el botón **Slice** para dividir la imagen en sus respectivos cuadros de animación.
6. Finalmente, haz clic en **Apply** (botón inferior derecho) para guardar los cortes realizados.
   Cada sprite quedará dividido en 7 fotogramas que se usarán para crear la animación.

Paso 3: Crear la animación del objeto MoneyBag

1. Abre el panel **Project** y selecciona la carpeta
   Assets/Images/TinySwords/Resources/Resources.
2. Expande el sprite G_Spawn para ver sus sub-imágenes (frames).

3. Selecciona todos los cortes (desde G_Spawn_0 hasta G_Spawn_6).
4. Arrástralos hacia el **Escenario (Scene)**.
5. Unity generará automáticamente una animación.
6. Cuando aparezca la ventana de guardar, crea una nueva carpeta dentro de **Assets/Animations** llamada **Resources**.
7. Guarda la animación con el nombre **SpawnMoneyBag** dentro de esa carpeta.
   Esta animación mostrará el efecto de aparición (spawn) del objeto MoneyBag.

Paso 4: Repetir el proceso para Meat y Wood

1. Repite el mismo procedimiento con:
   - M_Spawn → crea la animación **SpawnMeat**.
   - W_Spawn → crea la animación **SpawnWood**.
2. Guarda ambas animaciones también dentro de la carpeta:
   Assets/Animations/Resources
   Cada animación representará visualmente la aparición del recurso correspondiente.

Paso 5: Desactivar la repetición (Loop)

1. Selecciona las tres animaciones (**SpawnMoneyBag**, **SpawnMeat**, **SpawnWood**) en el panel **Project**.
2. En el **Inspector**, desmarca la opción **Loop Time**.
   Esto evita que las animaciones se repitan en bucle. Solo se reproducirán una vez al aparecer el objeto.

Paso 6: Renombrar los controladores de animación

1. Unity crea automáticamente un **Animator Controller** para cada animación.
2. Ubica estos controladores en el mismo directorio donde guardaste las animaciones.
3. Cambia los nombres de los controladores a:
   - SpawnMoneyBagController
   - SpawnMeatController
   - SpawnWoodController

   Cada controlador será responsable de gestionar la animación del recurso correspondiente.

Paso 7: Asignar los controladores a los objetos

1. En la **Jerarquía (Hierarchy)**, selecciona el objeto **MoneyBag**.
2. Haz clic en **Add Component → Animator**.
3. En el campo **Controller**, asigna el archivo **SpawnMoneyBagController**.
4. Repite el mismo proceso para los objetos:
   - **Meat** → asignar **SpawnMeatController**

- **Wood** → asignar **SpawnWoodController**

De este modo, cada objeto tiene su propia animación de aparición configurada.

Paso 8: Limpieza de la jerarquía

1. Las animaciones que se arrastraron al escenario durante la creación pueden haber dejado instancias temporales (objetos animados).
2. Verifica en la **Jerarquía** si existen objetos sueltos con nombres como:
   - G_Spawn_0 o similares.
3. Elimínalos, ya que las animaciones ahora están vinculadas directamente a los objetos finales (MoneyBag, Meat, Wood).
   Esto mantiene la jerarquía organizada y evita duplicaciones innecesarias.

Paso 9: Verificar el sprite inicial

1. Selecciona cada uno de los objetos de recursos (**MoneyBag**, **Meat**, **Wood**) en la jerarquía.
2. En el **Inspector**, revisa el **Sprite Renderer**.
3. Asegúrate de que el **Sprite** mostrado corresponda al último fotograma de la animación respectiva (por ejemplo, G_Spawn_6).
4. Si no coincide, reemplázalo manualmente arrastrando el sprite correcto.
   Esto garantiza que el objeto se muestre con la apariencia final después de su animación de aparición.

Paso 10: Ajustar los colisionadores

1. Selecciona cada uno de los objetos (**MoneyBag**, **Meat**, **Wood**).
2. En el componente **Box Collider 2D**, haz clic en **Edit Collider**.
3. Ajusta el contorno del collider para que se adapte al tamaño real del sprite.
   - Evita que el collider sea más grande o más pequeño de lo necesario.
4. Desactiva la edición haciendo clic nuevamente en el botón **Edit Collider**.
   Esto mejora la precisión de las colisiones al recolectar los objetos.

Resultado final:

Ahora cada recurso (MoneyBag, Meat y Wood) tiene su propia animación de aparición configurada, correctamente escalada, sin bucles y vinculada a su objeto en la escena. Cuando se generen en el juego, se mostrará una animación corta que simula su “aparición mágica” antes de quedar disponibles para ser recolectados por el jugador.

Tercera Parte
Creación de Prefabs en Unity

Paso 1: Crear la carpeta de prefabricados

1. En el panel **Project**, dirígete a la carpeta raíz **Assets**.
2. Haz clic derecho y selecciona **Create → Folder**.
3. Asigna el nombre **Prefabs** a la nueva carpeta.
   Esta carpeta servirá como contenedor para todos los objetos prefabricados del proyecto.

Paso 2: Convertir los objetos en Prefabs

1. Asegúrate de tener configurados y correctamente animados los siguientes objetos dentro de la escena (**Hierarchy**):
   - EnemyTorchRed (enemigo)
   - MoneyBag (recurso de dinero)
   - Meat (recurso de comida)
   - Wood (recurso de madera)
2. Abre la carpeta **Assets/Prefabs** en el panel **Project**.
3. Arrastra cada uno de los objetos mencionados desde la **Jerarquía (Hierarchy)** hasta la carpeta **Prefabs**.
   Unity creará automáticamente un archivo *.prefab* para cada objeto, manteniendo todas sus propiedades, componentes, animaciones, colisionadores y etiquetas.
   Cada Prefab actúa como un modelo base del objeto, lo que te permitirá instanciarlo múltiples veces en diferentes escenas sin tener que configurarlo nuevamente.

Paso 3: Confirmar la vinculación del Prefab

1. Observa que cada objeto en la escena ahora aparece con su nombre en **azul**, lo que indica que está vinculado a un Prefab en el proyecto.
2. Si realizas cambios en la configuración del objeto en la escena, puedes aplicar esos cambios al Prefab seleccionando el objeto y haciendo clic en **Overrides → Apply All**.
   Esto actualiza el Prefab con las modificaciones más recientes.

Paso 4: (Opcional) Verificar las rutas y dependencias

1. Selecciona cada Prefab dentro de **Assets/Prefabs** y revisa en el **Inspector**:
   - Que el **Animator** mantenga su controlador asignado.
   - Que los **Colliders** sigan activos y correctamente ajustados.
   - Que el **Sprite Renderer** muestre la imagen final deseada.

2. Si alguno de los elementos no aparece correctamente, vuelve a arrastrar el componente o la animación correspondiente desde las carpetas originales (por ejemplo, desde **Assets/Animations/Resources**).

Resultado final:

Ahora cuentas con *Prefabs funcionales* de todos los elementos principales del juego:

- EnemyTorchRed (enemigo animado)
- MoneyBag, Meat y Wood (recursos con animaciones y colisionadores ajustados)

Estos *Prefabs* podrán ser instanciados dinámicamente mediante código (por ejemplo, desde un *spawner*), o reutilizados en distintas escenas del proyecto sin necesidad de reconstruir su configuración.

Cuarta Parte
UI – Interfaz del Inventario

Paso 1: Crear el Canvas principal

1. En el menú superior selecciona **GameObject → UI → Canvas**.
2. Unity creará automáticamente un objeto **Canvas** junto con un **EventSystem**.
3. Selecciona el objeto **Canvas** en la jerarquía.
4. En el **Inspector**, dentro del componente **Canvas Scaler**, cambia:
    - **UI Scale Mode:** Scale With Screen Size

Esto asegura que todos los elementos de la interfaz mantengan su proporción al adaptarse a diferentes resoluciones o dispositivos.

Paso 2: Crear el panel de madera (Wood)

1. Con el Canvas seleccionado, crea un nuevo elemento **UI → Image**.
2. Cámbiale el nombre a **Wood**.
3. En el **Inspector**, asigna en **Source Image** el sprite:
   Assets/Images/TinySwords/UI/Ribbons/Ribbon_Blue_Connection_Down.png
4. Activa la opción **Preserve Aspect** para mantener la proporción original del diseño del panel.
5. En **Anchor Presets**, selecciona la posición **Bottom Left (Inferior Izquierda)**.
6. Ajusta el tamaño del panel para que ocupe un área visible, por ejemplo:
    - **Width:** 200
    - **Height:** 100
    - **Posición X:** 50
    - **Posición Y:** 50

El panel "Wood" servirá como contenedor visual para mostrar el icono y la cantidad de madera recolectada.

Paso 3: Agregar el texto del contador de madera

1. Con el objeto **Wood** seleccionado, crea un nuevo elemento **UI → Text - TextMeshPro**.
2. Cámbiale el nombre a **WoodText**.
3. Ajusta su posición para que quede **centrado dentro del panel Wood**.
4. En el **Inspector**, modifica los siguientes valores:
    - **Texto inicial:** 00
    - **Font Size:** 36
    - **Alignment:** Center
    - **Color:** blanco o gris claro (#E0E0E0).
5. Asegúrate de que el texto sea legible y esté visualmente equilibrado dentro del panel.

Este texto representará la cantidad actual de madera disponible para el jugador.

Paso 4: Añadir el ícono del recurso

1. Con el panel **Wood** seleccionado, crea un nuevo elemento hijo **UI → Image**.
2. Cámbiale el nombre a **WoodIcon**.
3. En **Source Image**, asigna el sprite:
   Assets/Images/TinySwords/Resources/Resources/W_Idle.png
4. Ajusta su tamaño para que el ícono quede **debajo del texto** y **dentro del panel Wood**.
    - **Width:** 64
    - **Height:** 64
    - **Posición Y:** -20
5. Activa **Preserve Aspect** para conservar la relación original del ícono.

El ícono refuerza visualmente el tipo de recurso que se muestra, facilitando su identificación.

Paso 5: Repetir el procedimiento para los otros recursos

1. Duplica el objeto **Wood** dos veces (Ctrl + D).
2. Renombra las copias como **Money** y **Meat**.
3. Modifica en cada uno:
    - En la imagen principal (**Image**), cambia el **Source Image** por el panel o cinta que desees mantener (puede ser el mismo o uno de color distinto).
    - En el texto, cambia los nombres a **MoneyText** y **MeatText** respectivamente, manteniendo el formato de texto centrado.
    - En el ícono interno, reemplaza las imágenes por:
        - **MoneyIcon →**
          Assets/Images/TinySwords/Resources/Resources/G_Idle_0.png
        - **MeatIcon →**
          Assets/Images/TinySwords/Resources/Resources/M_Idle_0.png

4. Ajusta sus posiciones para que los tres paneles (Wood, Money y Meat) estén alineados horizontalmente o verticalmente según la preferencia de diseño.

De este modo, la interfaz mostrará simultáneamente los tres tipos de recursos, cada uno con su contador e ícono representativo.

Paso 6: Agrupar los paneles en un contenedor común

1. Crea un nuevo objeto vacío (**Create Empty Parent**) dentro del Canvas.
2. Nómbralo **Inventory**.
3. Arrastra los objetos **Wood**, **Money** y **Meat** dentro de **Inventory** para agruparlos.
4. En el **Inspector**, ajusta el **Rect Transform** de **Inventory** y ancla su posición a la esquina **inferior izquierda (Bottom Left)**.
5. Ajusta la posición general del grupo para que quede visible, por ejemplo:
    - **Posición X:** 60
    - **Posición Y:** 60

Agrupar los elementos facilita el control del inventario como una sola unidad, permitiendo moverlo, ocultarlo o escalarlo conjuntamente.

Paso 7: Crear el controlador de interfaz (UI Manager)

1. Crea un nuevo objeto vacío en la jerarquía (fuera del Canvas).
2. Nómbralo **UIManager**.
3. En el panel **Project**, dirígete a **Assets/Scripts**.
4. Crea una nueva carpeta llamada **UI**.
5. Dentro de esa carpeta, crea un nuevo script C# con el nombre **UIManager.cs**.
6. Arrastra el script al objeto **UIManager** en la jerarquía para asignarlo como componente.

El UIManager será responsable, en etapas posteriores, de actualizar los contadores del inventario conforme el jugador recolecte recursos.

Quinta parte
Abrir y cerrar inventario

1. Abrimos el script “Assets/Scripts/UI/UIManager.cs” y agregamos el siguiente código:

```
using UnityEngine;

public class UIManager : MonoBehaviour
{
    public GameObject invetory;
    public static UIManager Instance { get; private set; }

    private void Awake()
    {
```

```
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    public void OpenOrCloseInvetory()
    {
        invetory.SetActive(!invetory.activeSelf);
    }
}
```

2. Asignamos el grupo “Invetory” al UI Mager Script en su propiedad “Invetory”.

3. Abrimos el script “Assets/Scripts/Player/Player.cs” y codificamos:

```
using System;
using UnityEngine;

public class Player : MonoBehaviour
{
    public float speed = 5;
    Rigidbody2D rb2d;
    Vector2 movementInput;
    Animator animator;
    void Start()
    {
        rb2d = GetComponent<Rigidbody2D>();
        animator = GetComponent<Animator>();
    }

    void Update()
    {
        movementInput.x = Input.GetAxisRaw("Horizontal");
        movementInput.y = Input.GetAxisRaw("Vertical");

        movementInput = movementInput.normalized;

        animator.SetFloat("Horizontal", Math.Abs(movementInput.x));
        animator.SetFloat("Vertical", Math.Abs(movementInput.y));

        CheckFlip();

        OpenCloseInvetory();
    }
```

```
    private void FixedUpdate()
    {
        rb2d.linearVelocity = movementInput * speed;
    }

    void CheckFlip()
    {
        if ((movementInput.x > 0 && transform.localScale.x < 0) ||
        (movementInput.x < 0 && transform.localScale.x > 0))
        {
            transform.localScale = new Vector3(
                transform.localScale.x * -1,
                transform.localScale.y,
                transform.localScale.z
            );
        }
    }

    void OpenCloseInvetory()
    {
        if (Input.GetKeyDown(KeyCode.I))
        {
            UIManager.Instance.OpenOrCloseInvetory();
        }
    }
}
```

Sexta Parte

Mostrar recursos en el Inventario

1. Abrimos el script “Assets/Scripts/UI/UIManager.cs” y agregamos el siguiente código:

```
using TMPro;
using UnityEngine;

public class UIManager : MonoBehaviour
{
    public GameObject invetory;

    public TMP_Text moneyCountText;
    public TMP_Text woodCountText;
    public TMP_Text meatCountText;

    public static UIManager Instance { get; private set; }

    private void Awake()
```

```
    {
        if (Instance == null)
        {
            Instance = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    public void OpenOrCloseInvetory()
    {
        invetory.SetActive(!invetory.activeSelf);
    }

    public void UpdateMoney(int value)
    {
        moneyCountText.text = value.ToString();
    }
    public void UpdateWood(int value)
    {
        woodCountText.text = value.ToString();
    }
    public void UpdateMeat(int value)
    {
        meatCountText.text = value.ToString();
    }
}
```

2. Asignamos a las propiedades del script UI Manager Script cada uno de los textos.

3. Editamos “Assets/Scripts/Player/PlayerResourcesCollector.cs”

```
using UnityEngine;

public class PlayerResourcesCollector : MonoBehaviour
{
    private int money = 0;
    private int meat = 0;
    private int wood = 0;

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.CompareTag("MoneyBag"))
        {
            Destroy(collision.gameObject);
            money++;
            Debug.Log("Tenemos " + money + " de dinero");
```

```
            UIManager.Instance.UpdateMoney(money);
        }
        if (collision.gameObject.CompareTag("Meat"))
        {
            Destroy(collision.gameObject);
            meat++;
            Debug.Log("Tenemos " + meat + " de comida");
            UIManager.Instance.UpdateMeat(meat);
        }
        if (collision.gameObject.CompareTag("Wood"))
        {
            Destroy(collision.gameObject);
            wood++;
            Debug.Log("Tenemos " + wood + " de madera");
            UIManager.Instance.UpdateWood(wood);
        }
    }
}
```