«Talento Tech»

Desarrollo de Videojuegos

Unity 3D

Clase 08

Buenos Aires Ciudad

Clase N° 8 | Interacción con UI en 3D I

Temario:

- Implementación First Person
- Canvas en el espacio 3D.
- Ejemplo Práctico (Data Panel inGame)

Objetivos de la clase

Implementar una perspectiva en primera persona (First Person) en Unity.

- Configurar una cámara en primera persona para interactuar con un entorno 3D.

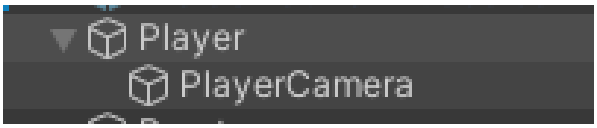
Configurar un Canvas en el espacio 3D.

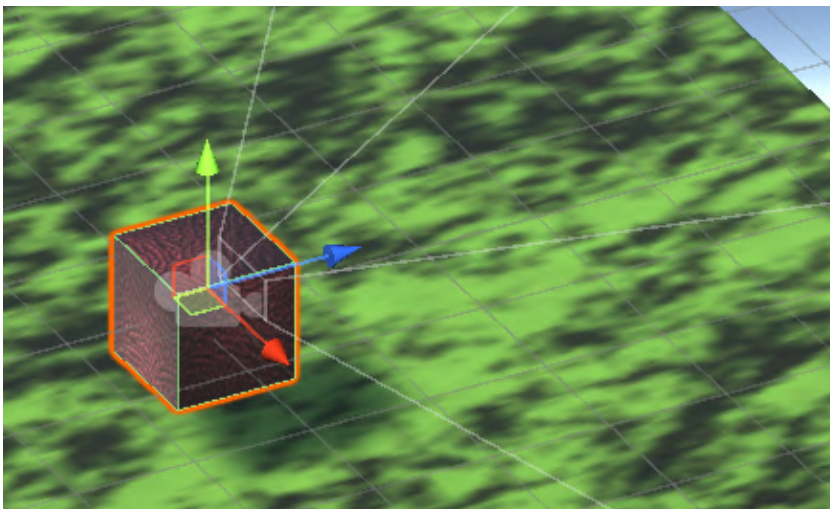
- Explorar cómo integrar elementos de interfaz de usuario (UI) directamente en el espacio 3D de la escena.
- Ajustar la posición y escala del Canvas para mantener la claridad y legibilidad.

First Person

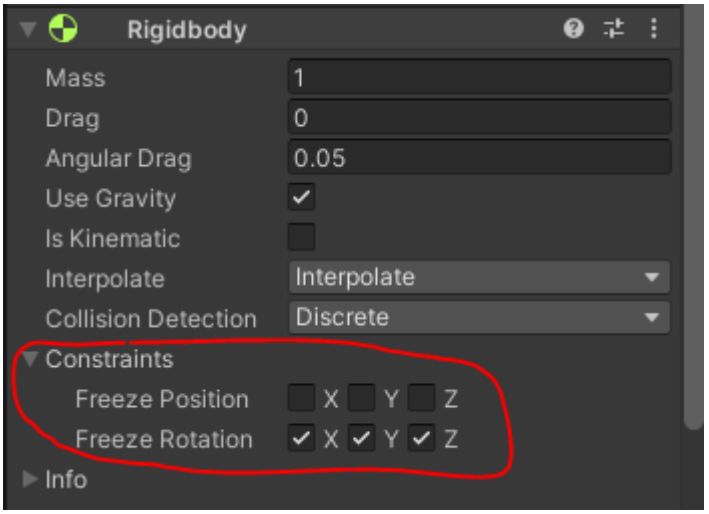
Hasta ahora, al menos que hayan descargado un Character Controller, hemos estado utilizando un estilo de cámara “Isométrico” como Diablo o Path of Exile. En la clase de hoy empezaremos a aplicar un estilo de “First Person” para crear más alternativas a la hora de diseñar nuestro UI.

Antes de iniciar, asegúrense de colocar una cámara bien alineada dentro del GameObject del player y desactivar la cámara que ya tenían.





También tenemos que freezar las rotaciones de nuestro personaje para asegurarnos de no voltearnos. Recordemos que esta es la opción “Constraints” dentro del componente de RigidBody



Script (Cámara)

```
public class CameraRotateScript : MonoBehaviour{
    private float x;
    private float y;
    public float sensitivity = -1f;
    private Vector3 rotate;

    // Start is called before the first frame update
    void Start() {
        Cursor.lockState = CursorLockMode.Locked;
    }

    // Update is called once per frame
    void Update(){
        y = Input.GetAxis("Mouse X");
        x = Input.GetAxis("Mouse Y");
        rotate = new Vector3(x, y * sensitivity, 0);
        transform.eulerAngles = transform.eulerAngles - rotate;
    }
}
```

En este código nos encontraremos con algunas cosas nuevas:

1)

```
        Cursor.lockState = CursorLockMode.Locked;
```

`Cursor`:

- Es una clase estática en Unity que controla el comportamiento del cursor (mouse) en la ventana del juego.

`lockState`:

- Es una propiedad de la clase `Cursor` que define cómo el cursor interactúa con la ventana del juego. Puede tomar valores de la enumeración `CursorLockMode`.

`CursorLockMode.Locked`:

- Es un modo en el que el cursor está bloqueado en el centro de la pantalla y no es visible. Aunque el cursor está físicamente inmovilizado, el movimiento del mouse sigue capturándose, permitiendo al jugador, por ejemplo, rotar una cámara o un personaje en un juego en primera o tercera persona.

2)

```
        transform.eulerAngles = transform.eulerAngles - rotate;
```

`transform.eulerAngles`:

- Representa las rotaciones del objeto en términos de ángulos de Euler (X, Y, Z) en grados.
- En este caso, toma los ángulos actuales del objeto (`transform.eulerAngles`) y les resta el valor calculado en `rotate`. Esto actualiza la rotación del objeto.

`transform.eulerAngles - rotate`:

- Resta el vector `rotate` para modificar la orientación del objeto:
    - **Eje X (`x`)**: Controla la inclinación (mirar hacia arriba o hacia abajo).
    - **Eje Y (`y * sensitivity`)**: Controla la rotación hacia los lados (girar hacia la izquierda o derecha).
    - **Eje Z (`0`)**: No se aplica ninguna rotación en este eje.

Con esto nuestra cámara estará lista para ser usada. Pero antes de dar Play, deberemos de chequear algo en nuestro Script del Player

Script (Player)

Puede llegar a pasar que si probamos el juego, nuestro Player/cámara rote, pero no cambie su dirección de movimiento.
Para solucionar esto, modificaremos el Script de Movimiento de nuestro personaje haciendo que e Vector3 que era “movement” sea modificado por la función “TransformDirection”

```
 float horizontalInput = Input.GetAxis("Horizontal");
 float verticalInput = Input.GetAxis("Vertical");

  Vector3 movement = new Vector3(horizontalInput, 0, verticalInput);
  Vector3 newDirection = transform.TransformDirection(movement);

 newDirection = newDirection.normalized;

 rb.velocity = new Vector3(newDirection.x * moveSpeed,rb.velocity.y,
newDirection.z * moveSpeed);
```

`transform.TransformDirection(movement)`:

- Convierte el vector `movement` (en el espacio local del objeto) a un vector en el espacio global.
- Esto significa que el movimiento ahora se alinea con la rotación actual del objeto.
- Por ejemplo, si el objeto rota, "adelante" (eje Z) seguirá siendo relativo a su orientación, no al mundo.

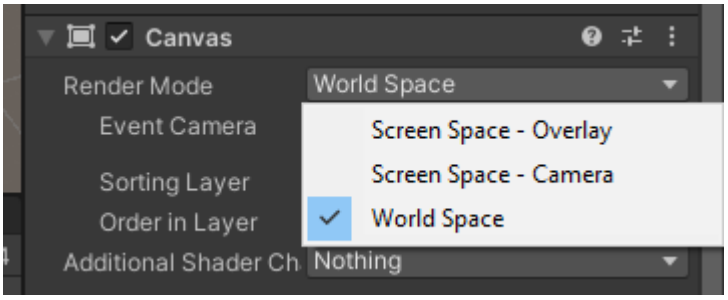
Con esto armado, al ir para “adelante” con la W, nuestro personaje estará siempre viendo en la dirección correcta.

UI en el juego

Usaremos un Canvas para crear un UI dentro del mismo juego. Este ya no estará fijo a la pantalla sino que nos lo podremos encontrar a medida que exploremos nuestro nivel.

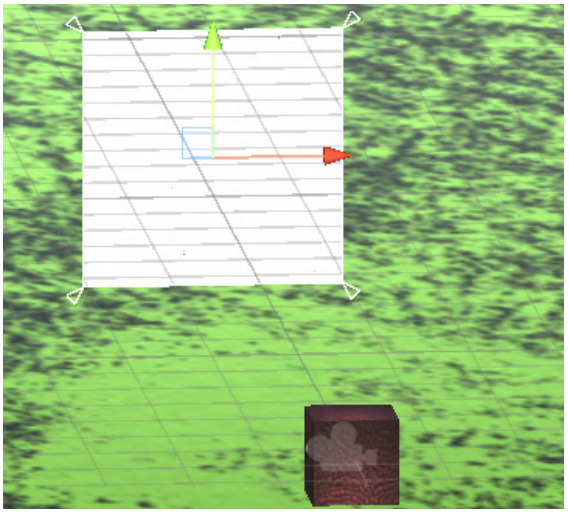
Seteo

Primero crearemos un canvas nuevo y cambiaremos la variable de Render Mode, de “Screen Space - Overlay” a “World Space”.

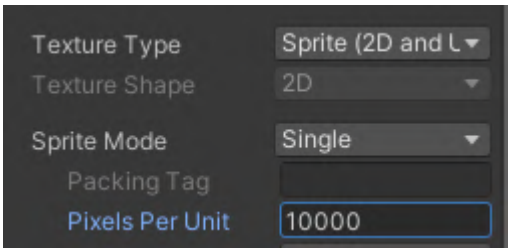


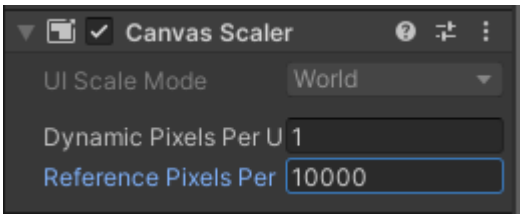
Seguiremos por cambiarle el tamaño algo mas chico y apropiado a la visión de nuestro personaje

Lo posicionaremos donde queramos y crearemos una UI->Image dentro para ya tener un “fondo blanco”



Si ven que su imagen se ve de una manera extraña prueben modificar la escala de píxeles tanto en la imagen como en el canvas que hicimos:





Tip: Si desean cambiar la escala, del canvas o la imagen, SIEMPRE asegúrense que sean la misma tanto el parent (canvas) como el child (imagen de fondo). Especialmente si desean rotarla.

Ahora, si lo deseamos, podemos colocar nuestro canvas para que nos de información dentro del juego. Probemos con un cartel de aviso al lado de una puerta:

Con esto tendremos un UI básico dentro de nuestro juego

Data In-Game

Vamos a darle uso a lo que venimos viendo. Para eso crearemos un “panel” in game que nos muestre datos de nuestro personaje.
Imaginemos que en nuestro juego, nuestro personaje todavía no posee la tecnología que le deja visualizar su estatus, por lo tanto, para saber cuanta vida, energía o mana tiene, debe encontrar los distintos paneles dentro del juego. Esto nos permitirá crear distintas interacciones con el mundo y variar la siempre “vida en pantalla”.

Seteo

Primero, dentro de nuestra imagen crearemos 3 barras. Una de Vida, otra de Mana y otra de Energía.
Simplemente crearemos la primera y luego la copiaremos y modificaremos para las otras 2

Por ahora dejaremos las 3 deshabilitadas

Script

Seguiremos por crear un Script sencillo, con fines prácticos, y lo pondremos dentro nuestro objeto contenedor “Panel”.
Haremos que, cuando mi personaje esté cerca del objeto, si yo aprieto un botón, pasaré de “pantalla en pantalla” para chequear mi información.

```
    private List<GameObject> children = new List<GameObject>();
    private int currentIndex = 0;
    private bool playerIn = false;
    private void Start()
    {
        // Llenar la lista con los hijos del panel
        foreach (Transform child in gameObject.transform)
        {
            children.Add(child.gameObject);
        }

        // Asegurarse de que sólo el primer hijo esté activo al inicio
        UpdateVisibility();
    }


    private void Update()
    {

        if (Input.GetKeyDown(KeyCode.F) && playerIn)
        {
            Debug.Log("Cambio");
            AdvanceSlide();
        }
    }

    private void AdvanceSlide()
    {
        // Ocultar el actual y avanzar al siguiente
        if (children.Count == 0) return; // Evitar errores si no hay
hijos

        children[currentIndex].SetActive(false);
```

```
        currentIndex = (currentIndex + 1) % children.Count; // Ciclo
circular
        children[currentIndex].SetActive(true);
    }

    private void UpdateVisibility()
    {
        for (int i = 0; i < children.Count; i++)
        {
            children[i].SetActive(i == currentIndex);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            Debug.Log("Player Entro");
            playerIn = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            Debug.Log("Player salio");
            playerIn = false;
        }
    }
```

Explicación del Script

Variables

```
    private List<GameObject> children = new List<GameObject>();
    private int currentIndex = 0;
    private bool playerIn = false;
```

- **`children`**: Lista que contendrá los hijos del objeto donde está el script.
- **`currentIndex`**: Índice que señala cuál de los hijos está visible.
- **`playerIn`**: Variable booleana que indica si el jugador está dentro del rango del trigger asociado al objeto.

Método Start()

```
private void Start()
    {
        // Llenar la lista con los hijos del panel
        foreach (Transform child in gameObject.transform)
        {
            children.Add(child.gameObject);
        }

        // Asegurarse de que sólo el primer hijo esté activo al inicio
        UpdateVisibility();
    }
```

- **`foreach`**: Recorre todos los hijos del objeto donde está el script (`gameObject.transform`) y los agrega a la lista `children`.
- **`UpdateVisibility()`**: Activa únicamente el primer hijo (`currentIndex = 0`) y desactiva los demás.

Método Update()

```
    private void Update()
    {

        if (Input.GetKeyDown(KeyCode.F) && playerIn)
        {
            Debug.Log("Cambio");
            AdvanceSlide();
        }
    }
```

- **`Input.GetKeyDown(KeyCode.F)`**: Comprueba si se presiona la tecla F.
- **`playerIn`**: Verifica que el jugador esté dentro del rango del trigger.
- Si ambas condiciones se cumplen, se llama a `AdvanceSlide()` para avanzar a la siguiente "filmina" y registra el evento en la consola (`Debug.Log`).

Método AdvanceSlide()

```
    private void AdvanceSlide()
    {
        // Ocultar el actual y avanzar al siguiente
        if (children.Count == 0) return; // Evitar errores si no hay
hijos

        children[currentIndex].SetActive(false);

    currentIndex ++; // Incrementa el índice al siguiente elemento
if (currentIndex >= children.Count) // Si el índice supera el número de
elementos
{
    currentIndex = 0; // Reinicia el índice al principio
(comportamiento circular)
}



        children[currentIndex].SetActive(true);
    }
```

- Si no hay hijos en la lista (`children.Count == 0`), simplemente sale del método.
- **Avance circular**: Para avanzar al siguiente índice. Si llega al último elemento, vuelve al primero (comportamiento circular).
- Cambia la visibilidad desactivando el hijo actual y activando el siguiente.

Método UpdateVisibility()

```
    private void UpdateVisibility() {
        for (int i = 0; i < children.Count; i++) {
            children[i].SetActive(i == currentIndex);
        }
    }
```

- Itera por todos los hijos en la lista `children`.
- Activa el hijo cuyo índice coincide con `currentIndex` y desactiva los demás.

Métodos de Trigger

```
    private void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player")) {
            Debug.Log("Player Entro");
            playerIn = true;
        }
    }
    private void OnTriggerExit(Collider other) {
        if (other.CompareTag("Player")) {
            Debug.Log("Player salio");
            playerIn = false;
        }
    }
```

`OnTriggerEnter(Collider other)`:

- Se llama cuando un objeto entra en el área del trigger del objeto que tiene este script.
- Si el objeto entrante tiene la etiqueta `"Player"`, registra el evento en la consola y cambia `playerIn` a `true`.

`OnTriggerExit(Collider other)`:

- Se llama cuando el objeto sale del área del trigger.
- Si el objeto saliente tiene la etiqueta `"Player"`, registra el evento en la consola y cambia `playerIn` a `false`.

Al terminar, tendremos un panel InGame que nos permitirá cambiar de imágenes al acercarnos. Bastará generar referencias a los respectivos valores y ya tendremos un panel que muestre nuestro status InGame.

Sumergido en 3D:

Con los cimientos del juego en su lugar, el cliente quiere que Nexus brinde una experiencia aún más inmersiva. Hasta ahora, el enfoque ha estado en mecánicas y sistemas generales, pero ahora es momento de situar al jugador dentro del mundo de Nexus de una manera más directa: *"Queremos que sientan que son parte del mundo, no solo observadores desde afuera."*

El equipo de TalentoLab recibe un nuevo desafío: implementar una perspectiva en primera persona y aprovechar el **Canvas en el espacio 3D** para crear elementos interactivos y visualmente integrados al entorno. Esto incluye la creación de un sistema que muestre información contextual al jugador en tiempo real, como un **Panel de Datos in-game**, que permita visualizar detalles importantes del entorno o del progreso de la partida.

Ejercicios prácticos:

La solicitud del cliente:
El cliente quiere que el equipo reorganice los eventos existentes utilizando un **EventManager**. Este nuevo sistema debe ser capaz de gestionar eventos actuales y futuros con mayor eficiencia, asegurando que todo el juego funcione como un ecosistema bien integrado.

Luigi te envió un mensaje del cliente: "El panel de datos que han diseñado es un gran comienzo, pero necesitamos que sea más funcional e informativo. Queremos que este panel refleje en tiempo real las estadísticas del jugador, como su vida, maná, energía y oro. Este detalle no solo añade inmersión, sino que también permite a los jugadores tomar decisiones estratégicas en tiempo real."

Elizabeth recalca que el sistema debe ser dinámico y capaz de adaptarse a los cambios en las estadísticas del jugador durante la partida. Por ejemplo:

Cuando el jugador pierde vida tras un enfrentamiento, el panel debe actualizarse al instante.
Si recolecta oro o recupera energía, estos cambios deben reflejarse inmediatamente en pantalla.
Este ejercicio no es solo una implementación técnica, sino una pieza clave para construir una experiencia inmersiva y conectada al universo de Nexus.

Aclaración: La información a mostrar en el panel es a gusto del estudiante

Materiales y recursos adicionales.

Image:
https://docs.unity3d.com/2022.3/Documentation/Manual/script-Image.html

Canvas:
https://docs.unity3d.com/es/2018.4/Manual/class-Canvas.html

Preguntas para reflexionar.

1. ¿Por qué creen que para esta clase preferimos adoptar una cámara First Person?
2. ¿Qué beneficios puede traer un UI Ingame?
3. ¿Conocen ejemplos de algún juego que lo use?

Próximos pasos.

En la próxima clase Seguiremos trabajando con el UI y empezaremos a conocer la herramienta de maquetado “ProBuilder”.

Buenos Aires Ciudad