Práctica: Movimiento del Jugador Paso a Paso

Objetivo:

Implementar gradualmente los controles de movimiento del jugador, comprendiendo cada instrucción. Además, explorar el uso de prefabs descargados para enriquecer la escena.

Fase 1: Configurar el entorno

1. **Crear un nuevo proyecto:**
   - Tipo: 3D (Core).
   - Nombre: Movimiento_Prefabs.
   - Elimina el objeto Main Camera y el objeto Directional Light si están presentes.
2. **Crear el escenario**:
   - Crea un **Plane** y cámbiale el nombre a "Suelo".
   - Escálalo a `(10, 1, 10)` para que sea suficientemente grande.
   - Añade una **Directional Light** para iluminar la escena.
3. **Crear el jugador**:
   - Crea un **Cube** y renómbralo "Jugador".
   - Posición inicial: `(0, 0.5, 0)`.
   - Añade un **Rigidbody** al cubo para que pueda interactuar con físicas.
   - Crea un material (por ejemplo, azul) y aplícalo al jugador.
   - Coloca una Main Camera en la escena, como hija del jugador para que lo siga automáticamente.
     - Ajusta su posición para tener una buena vista (por ejemplo, (0, 10, -10)).
4. **Preparar el script**:
   - Crea un script llamado `ControlJugador`.
   - Asigna este script al jugador.

Fase 2: Movimiento horizontal

1. **Agregar el código básico**:
   - Abre el script `ControlJugador` y añade el siguiente código:

```
using UnityEngine;

public class ControlJugador : MonoBehaviour
{
    public float velocidad = 5f;  // Velocidad del movimiento

    void Update()
    {
        // Movimiento hacia adelante y atrás
```

```
        float moverAdelanteAtras = Input.GetAxis("Vertical") *
velocidad * Time.deltaTime;
        transform.Translate(0, 0, moverAdelanteAtras);
    }
}
```

2. **Probar el movimiento**:
    - Ejecuta la escena (`Play`) y usa las teclas **W** y **S** (o flechas arriba y abajo) para mover el jugador hacia adelante y atrás.
        - ¿Qué hace `Input.GetAxis("Vertical")`?
        - ¿Por qué multiplicamos por `Time.deltaTime`?

Fase 3: Añadir rotación horizontal

1. **Expandir el script**:
    - Modifica el método `Update` para incluir rotación:

```
void Update()
{
    // Movimiento hacia adelante y atrás
    float moverAdelanteAtras = Input.GetAxis("Vertical") * velocidad *
Time.deltaTime;
    transform.Translate(0, 0, moverAdelanteAtras);

    // Rotación izquierda y derecha
    float rotarIzquierdaDerecha = Input.GetAxis("Horizontal") *
velocidad * Time.deltaTime;
    transform.Rotate(0, rotarIzquierdaDerecha, 0);
}
```

2. **Probar la rotación**:
    - Usa las teclas **A** y **D** (o flechas izquierda y derecha) para rotar el cubo.
        - ¿Cómo afecta la rotación al movimiento del jugador?
        - Modificar el script para ajustar la velocidad de movimiento

---

Fase 4: Añadir saltos

1. **Modificar el script**:
    - Añade una nueva variable para la fuerza del salto y un bloque de código para implementar el salto:

```
public float fuerzaSalto = 5f; // Fuerza del salto
private Rigidbody rb;          // Referencia al Rigidbody

void Start()
{
    rb = GetComponent<Rigidbody>(); // Obtener el componente Rigidbody
}

void Update()
{
```

```
    // Movimiento hacia adelante y atrás
    float moverAdelanteAtras = Input.GetAxis("Vertical") * velocidad *
Time.deltaTime;
    transform.Translate(0, 0, moverAdelanteAtras);

    // Rotación izquierda y derecha
    float rotarIzquierdaDerecha = Input.GetAxis("Horizontal") *
velocidad * Time.deltaTime;
    transform.Rotate(0, rotarIzquierdaDerecha, 0);

    // Saltar
    if (Input.GetKeyDown(KeyCode.Space) && Mathf.Abs(rb.velocity.y) <
0.01f)
    {
        rb.AddForce(Vector3.up * fuerzaSalto, ForceMode.Impulse);
    }
}
```

2. **Probar el salto**:
    - Pulsa **Espacio** para saltar.
    - Asegúrate de que el jugador no pueda saltar más de una vez en el aire.
3. **Explicación**:
    - `Rigidbody.AddForce` aplica una fuerza en la dirección especificada.
    - `Mathf.Abs(rb.velocity.y) < 0.01f` asegura que el jugador esté en el suelo antes de saltar.

Fase 5: Descargar y personalizar prefabs

1. **Descargar prefabs**:
    - Descarga un pack gratuito desde la Asset Store (por ejemplo, árboles, muebles, o vehículos).
2. **Añadir prefabs a la escena**:
    - Arrastra varios prefabs desde el panel **Project** al **Hierarchy**.
    - Colócalos estratégicamente en la escena.
3. **Personalizar los prefabs**:
    - Cambia materiales:
        - Duplica un material (`Ctrl + D`) y cámbiale el color o la textura.
        - Asigna el nuevo material a una instancia.
    - Ajusta las transformaciones:
        - Cambia la escala (`Scale`) para hacer objetos más grandes o pequeños.
        - Rota los objetos para variar su orientación.

Fase 6: Generación dinámica de prefabs (opcional)

1. **Crear un generador de objetos**:
    - Crea un objeto vacío (`Empty GameObject`) llamado "Generador".
    - Añade un script llamado `GeneradorPrefabs` con este código:

```
using UnityEngine;

public class GeneradorPrefabs : MonoBehaviour
{
    public GameObject prefab;  // Prefab a generar
    public int cantidad = 5;   // Número de objetos a generar

    void Start()
    {
        for (int i = 0; i < cantidad; i++)
        {
            Vector3 posicionAleatoria = new Vector3(
                Random.Range(-5, 5),
                0.5f,
                Random.Range(-5, 5));
            Instantiate(prefab, posicionAleatoria,
Quaternion.identity);
        }
    }
}
```

2. **Configurar el generador**:
    - Asigna un prefab descargado al campo `Prefab` en el Inspector.
    - Ejecuta la escena para ver cómo los objetos se generan aleatoriamente.