



PRÁCTICA 5 . Configuración Activos NetCode.

Programación en red, multijugador e IA

Juan Alberto Martínez Segura

Curso Especialización Desarrollo Videojuegos y Realidad Virtual

Índice

Introducción	2
Paso 1: Preparación del Proyecto	3
Paso 2: Importación de los Activos	4
1. Recibir los Activos	4
2. Importar el Paquete de Activos	5
Paso 3: Exploración de los Activos Importados	7
Paso 4: Organización de Prefabs	7
Paso 5: Configuración del Tanque	8
1. Prefab del Jugador	8
2. Configurar el Network Transform del Jugador	8
Paso 6: Añadir las ruedas del Tanque	9
1. Crear Objeto Hijo para las ruedas	9
2. Añadir el Sprite Renderer a las ruedas	9
3. Posicionar las ruedas	10
4. Añadir Network Transform a las ruedas	10
5. Añadir BoxCollider a las ruedas y ajustarlo	11
Paso 7: Añadir el Cuerpo del Tanque	11
1. Crear Objeto Hijo para el Cuerpo	11
2. Añadir el Sprite Renderer al Cuerpo	12
3. Posicionar el Cuerpo	12
Paso 8: Añadir el Pivote y la Torreta del Tanque	13
1. Crear el Pivote de la Torreta	13
2. Añadir la Torreta	13
Paso 9: Configurar el Orden de Renderización	14
1. Crear una Sorting Layer para el Jugador	14
2. Asignar la Sorting Layer y Orden a los Sprites	14
Paso 10: Personalizar el Color del Tanque	16
1. Cambiar el Color del Cuerpo	16
2. Aplicar el Mismo Color a la Torreta	17
Paso 11: Guardar y Salir del Modo Prefab	18
Paso 12: Probar el Tanque en el Juego	18
Paso 13: Desafío de Diseño (Opcional)	25
Personaliza tu Tanque o Crea un Nuevo Vehículo	25
Conclusión	27

Introducción

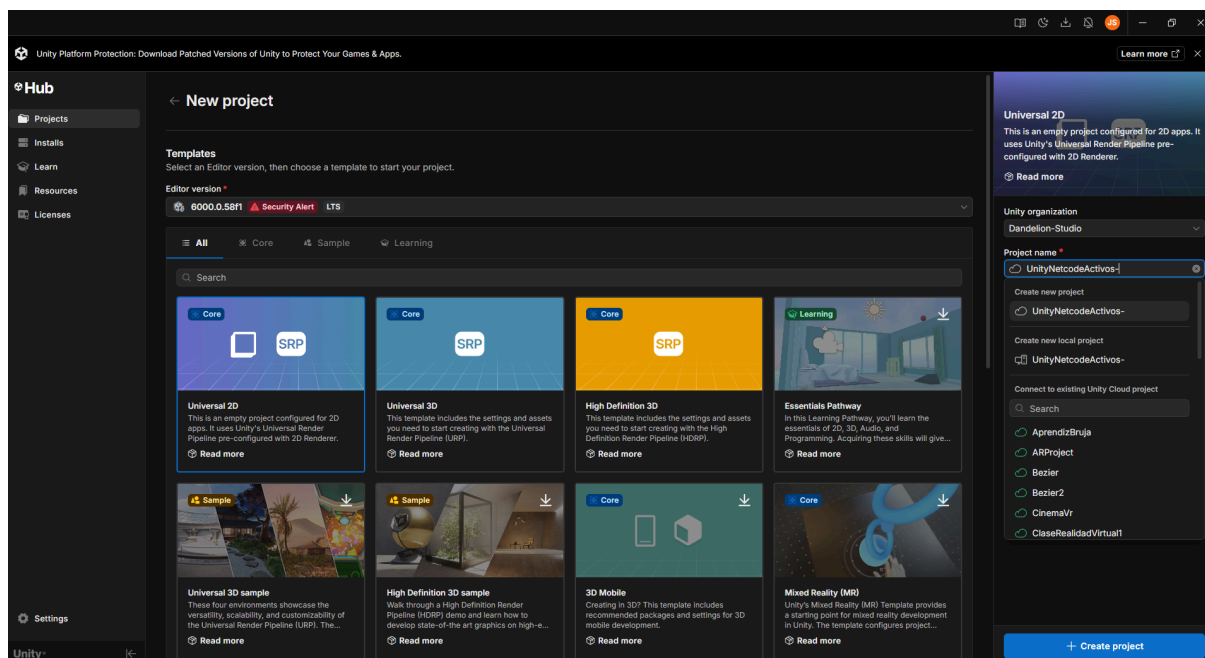
En esta práctica, vamos a mejorar la apariencia de nuestro juego importando y configurando los activos (assets) proporcionados. Vamos a transformar nuestro tanque de un simple círculo blanco a un vehículo detallado y funcional. Además, vamos a aprender cómo configurar correctamente los componentes de red para sincronizar los movimientos y acciones en un entorno multijugador.

Paso 1: Preparación del Proyecto

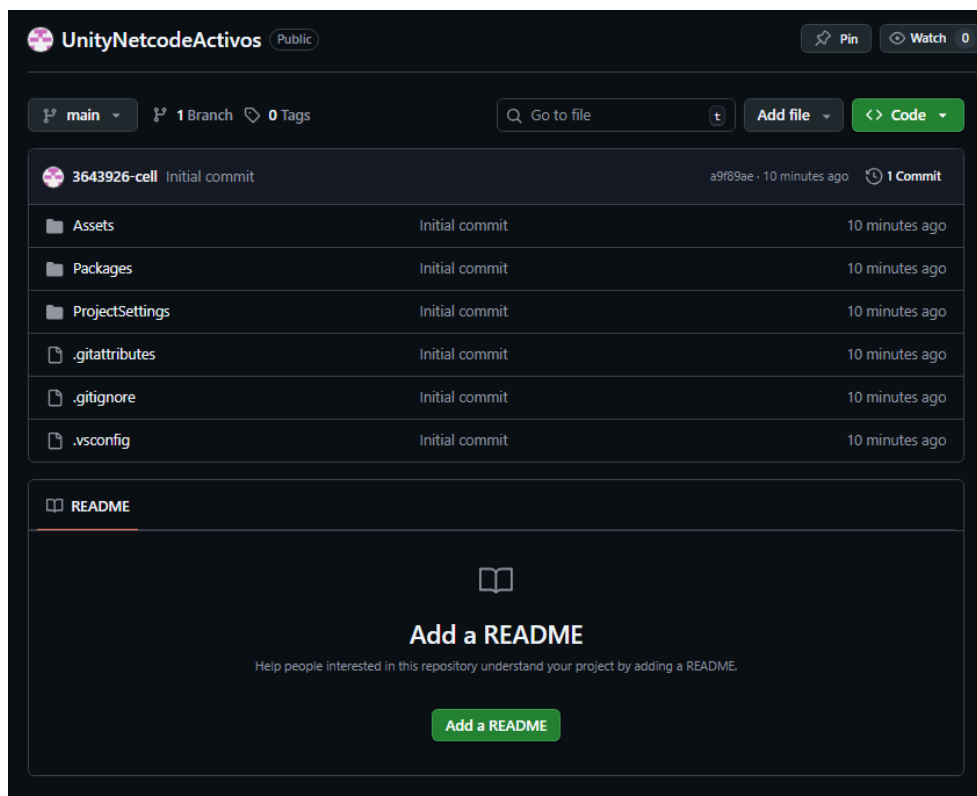
En nuestro caso vamos a crear un proyecto en Unity usando la plantilla Universal 2D.

He nombrado a mi proyecto UnityNetcodeActivos.

Una vez creado he realizado los pasos del proyecto anterior para poder partir de ese punto, de cara a resolver esta práctica.



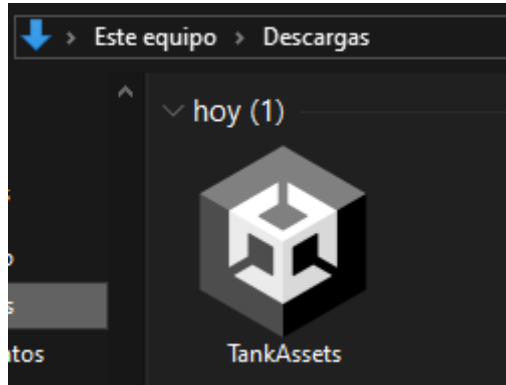
Además he aprovechado para subir el repositorio del proyecto inicializado y configurado a Github.



Paso 2: Importación de los Activos

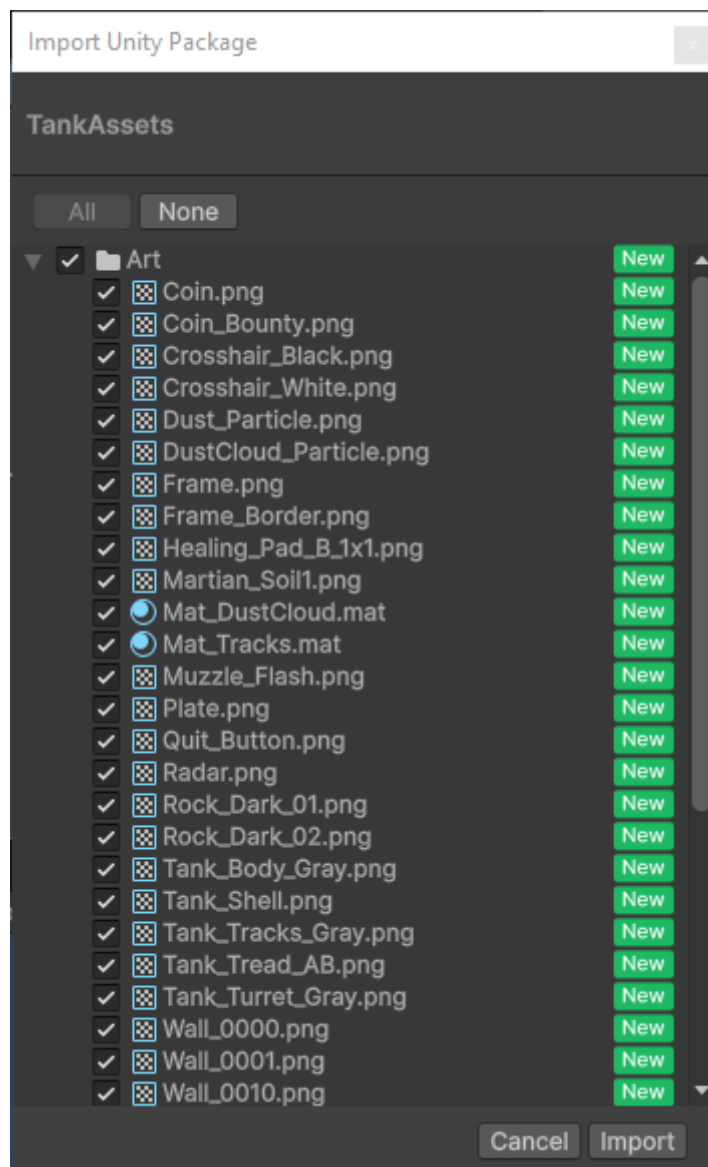
1. Recibir los Activos

Los activos necesarios para esta lección serán proporcionados por el instructor. Por eso debemos asegurarnos de tener el archivo .unitypackage disponible tras la descarga en el ordenador.

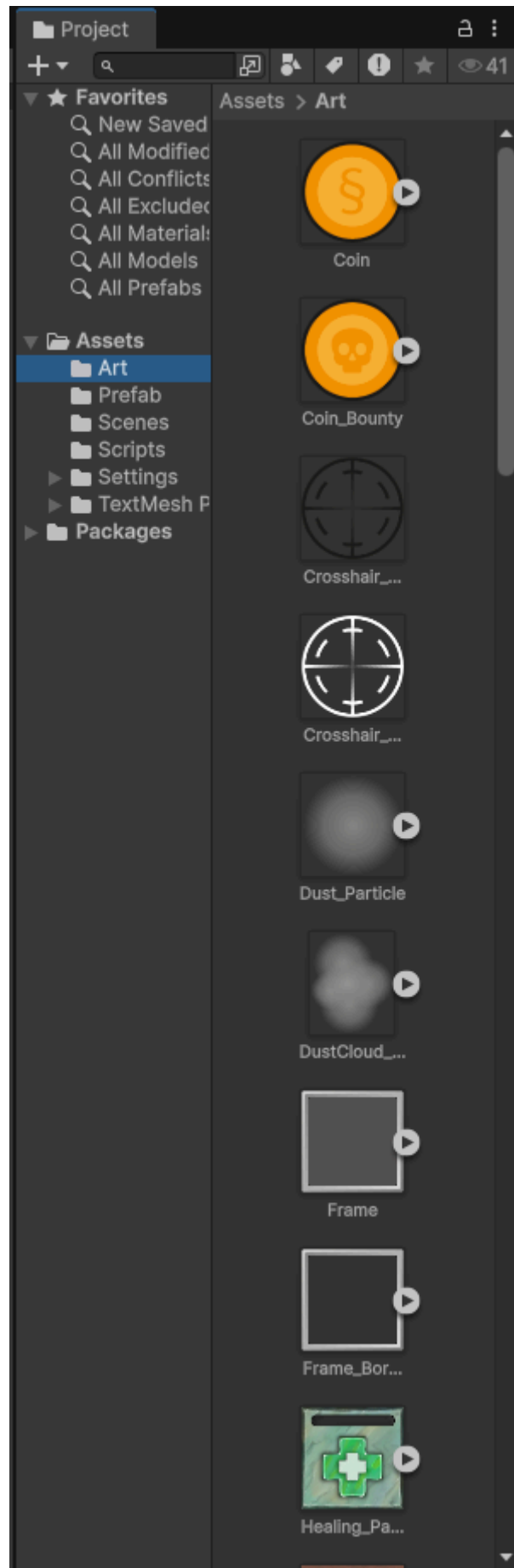


2. Importar el Paquete de Activos

Dentro de Unity y con el panel del Inspector activo y a la vista arrastramos el unityPackage dentro y se nos abrirá una ventana que nos permite decidir los Assets que queremos importar al proyecto desde el unityPackage. En este caso importamos todo.



Verificamos ahora en Project que todo se ha importado correctamente.



Paso 3: Exploración de los Activos Importados

En este caso comprobamos navegando en la carpeta Art que están todos los archivos que vamos a usar.

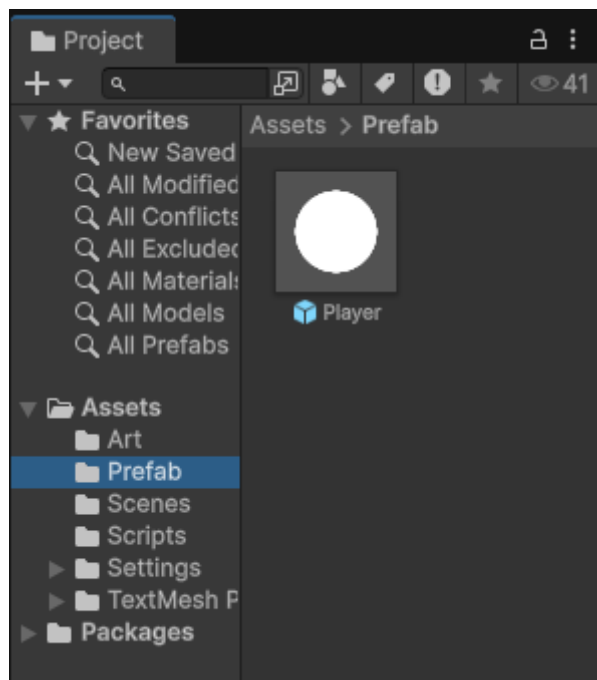
Explicación:

- **Activos Importados:**

- **TankTreads (ruedas del Tanque):** Las ruedas o cadenas que permitirán el movimiento del tanque.
- **TankBody(Cuerpo del Tanque):** La parte principal del tanque donde se sienta el conductor.
- **Turret (Torreta):** El cañón o arma que rotará para apuntar y disparar.
- **OtrosElementos:** Paredes modulares para construir el mapa y objetos de recogida para el juego.

Paso 4: Organización de Prefabs

Como estamos trabajando sobre el proyecto anterior, solo tenemos que verificar que tenemos creada una carpeta Prefab, y que tenemos un Player Prefab en su interior.



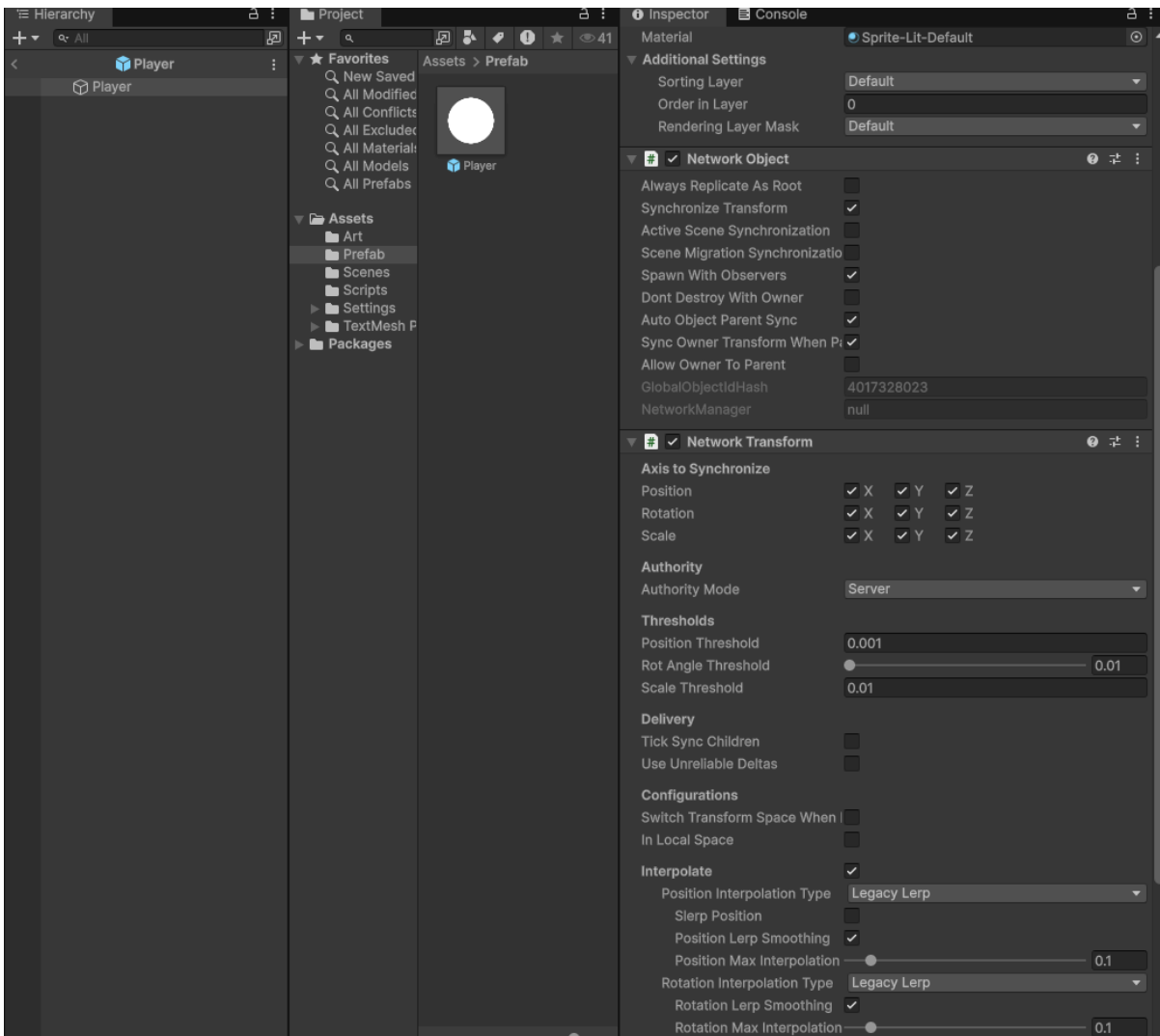
Explicación:

- **Prefabs:** Son objetos preconfigurados que podemos reutilizar en nuestras escenas. Mantenerlos organizados en una carpeta facilita su manejo y mantenimiento.

Paso 5: Configuración del Tanque

1. Prefab del Jugador

Lo primero que necesitamos hacer es asegurarnos también de que el Player Prefab tiene los componentes Network Object y Network Transform.



Explicación:

- NetworkObject: Marca este objeto para ser reconocido por el sistema de red de Unity.
- NetworkTransform: Permite sincronizar la posición y rotación del objeto a través de la red.

2. Configurar el Network Transform del Jugador

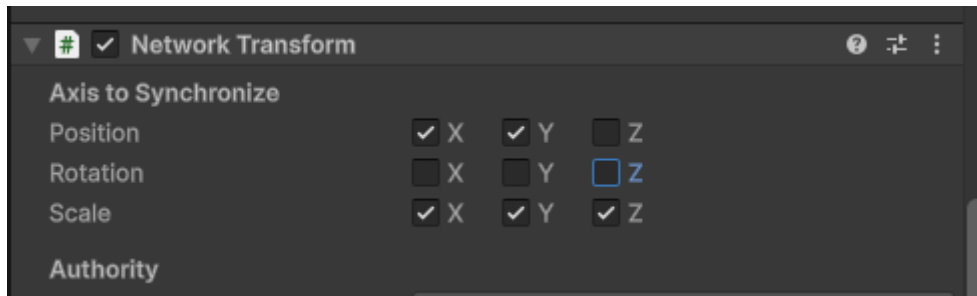
Debemos ir al inspector una vez abierto el prefab del Jugador, buscar el Network Transform y desmarcar las siguientes opciones:

- Sync Position Z
- Sync Rot Angle X
- Sync Rot Angle Y
- Sync Rot Angle Z

Y luego marcar las siguientes opciones:

■ Sync Position X

■ Sync Position Y



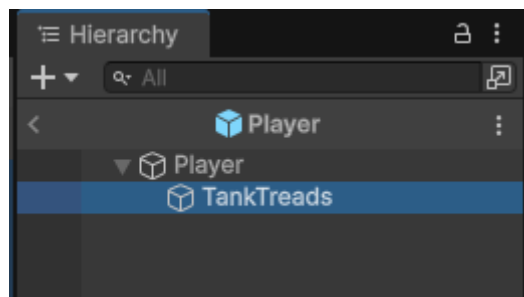
Explicación:

- Como estamos trabajando en 2D, solo necesitamos sincronizar las posiciones X e Y (horizontal y vertical).
- No sincronizamos la rotación en el objeto raíz porque manejaremos la rotación en los objetos hijos (torreta y ruedas).

Paso 6: Añadir las ruedas del Tanque

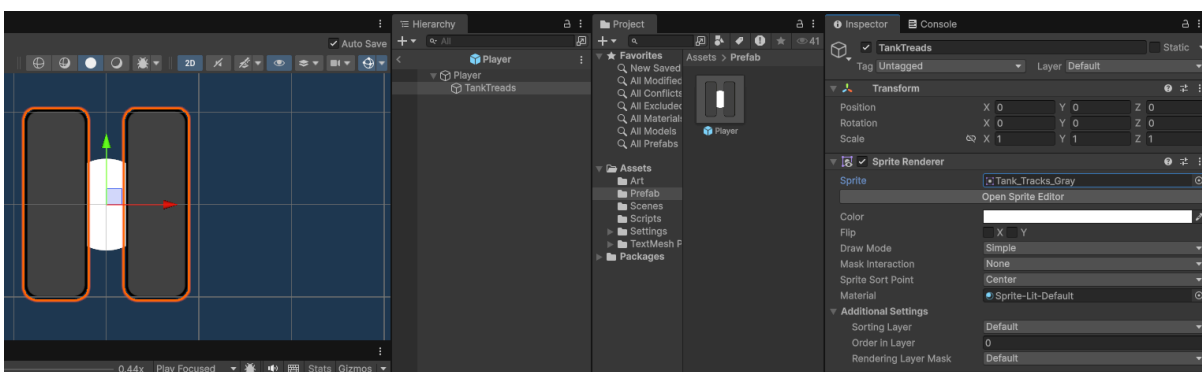
1. Crear Objeto Hijo para las ruedas

Seleccionamos el Player y creamos un objeto vacío nombrado a Tank Treads.



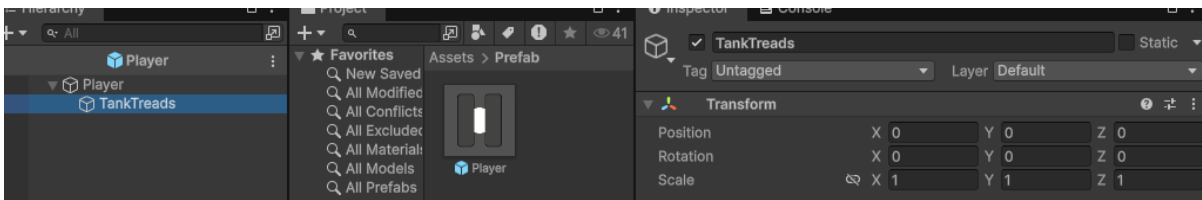
2. Añadir el Sprite Renderer a las ruedas

Al objeto Tank Treads le añadimos el componente Sprite Renderer, y en el campo Sprite de este componente le asignamos el Sprite correspondiente a las ruedas.



3. Posicionar las ruedas

Nos aseguramos de que las posiciones X, Y, Z en el Transform estén a 0.



4. Añadir Network Transform a las ruedas

A las ruedas le añadimos el componente Network Transform.

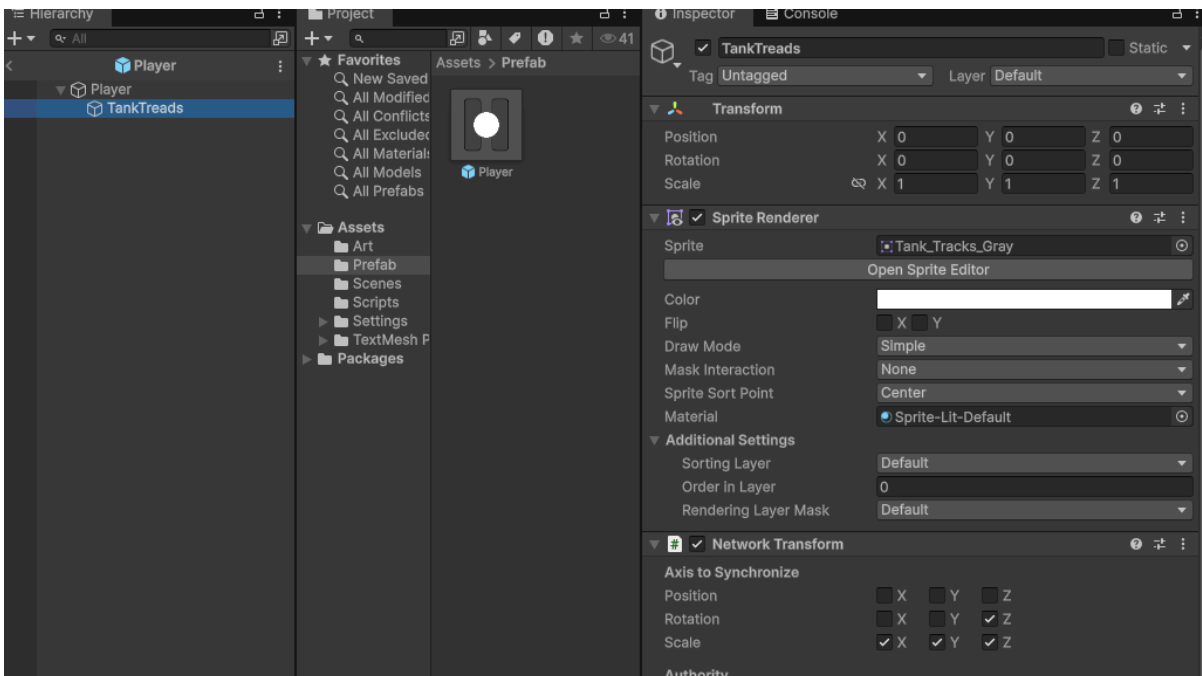
Lo configuramos de la siguiente manera:

Desmarcar:

- Sync Position X
- Sync Position Y
- Sync Position Z
- Sync Rot Angle X
- Sync Rot Angle Y

Marcar:

- Sync Rot Angle Z



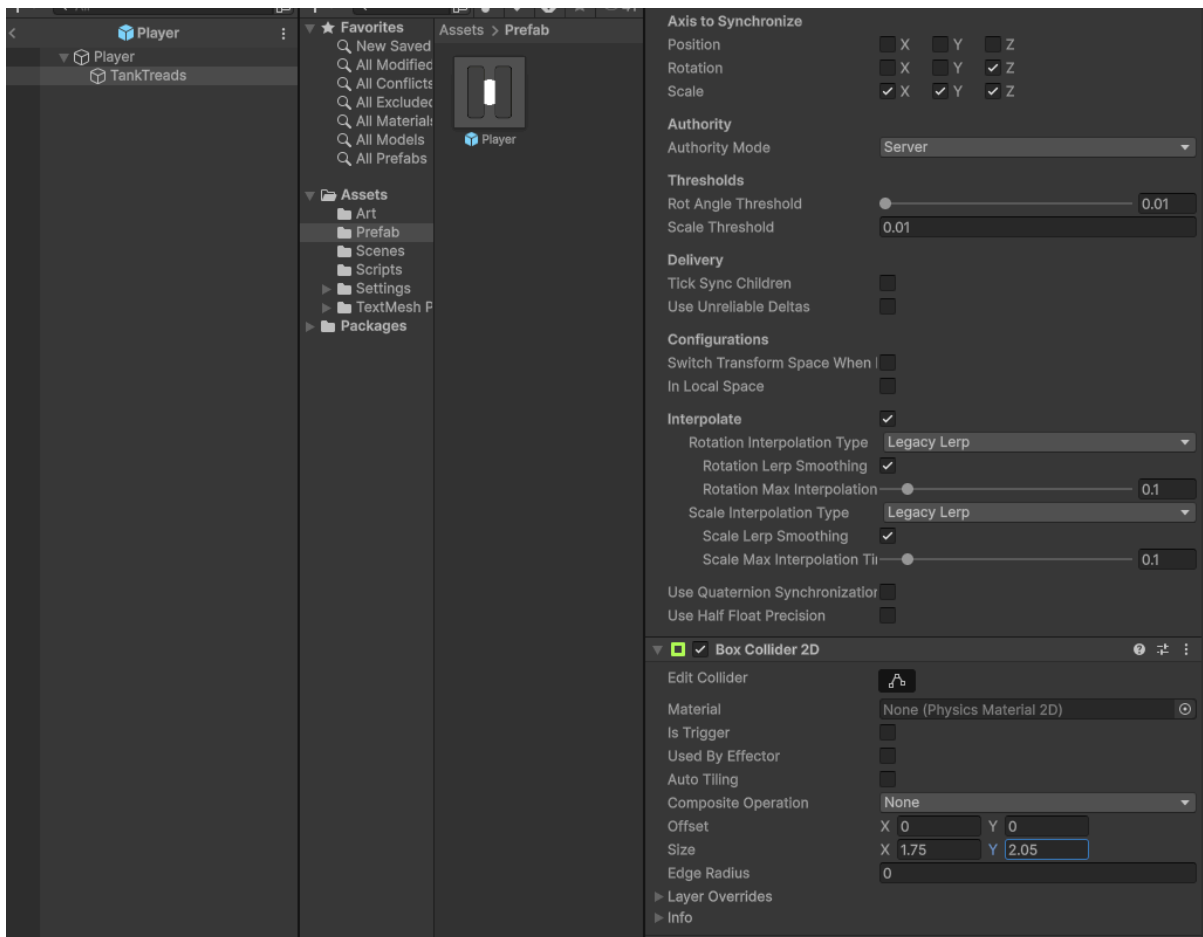
Explicación:

- Sincronizamos solo la rotación en Z de las ruedas porque queremos que la rotación (giro) se sincronice en la red, pero no necesitamos sincronizar su posición, ya que se moverán junto con el objeto Player.

5. Añadir BoxCollider a las ruedas y ajustarlo

Añadimos el componente Box Collider 2D a las ruedas y ajustamos los valores de Size de la siguiente manera:

- SizeX:1.75
- SizeY: 2.05



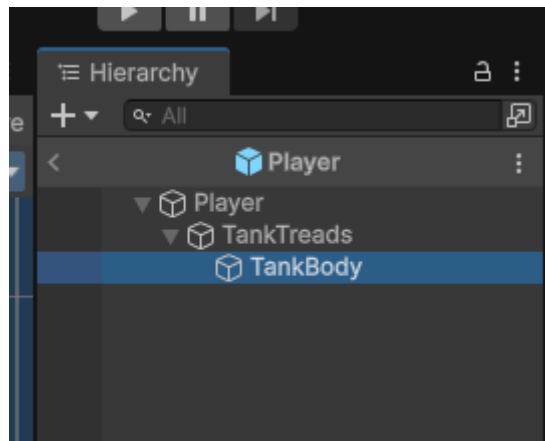
Explicación:

- El Box Collider 2D permite detectar colisiones con otros objetos en el juego.
- Ajustamos el tamaño para que coincida con las dimensiones visuales de las ruedas.

Paso 7: Añadir el Cuerpo del Tanque

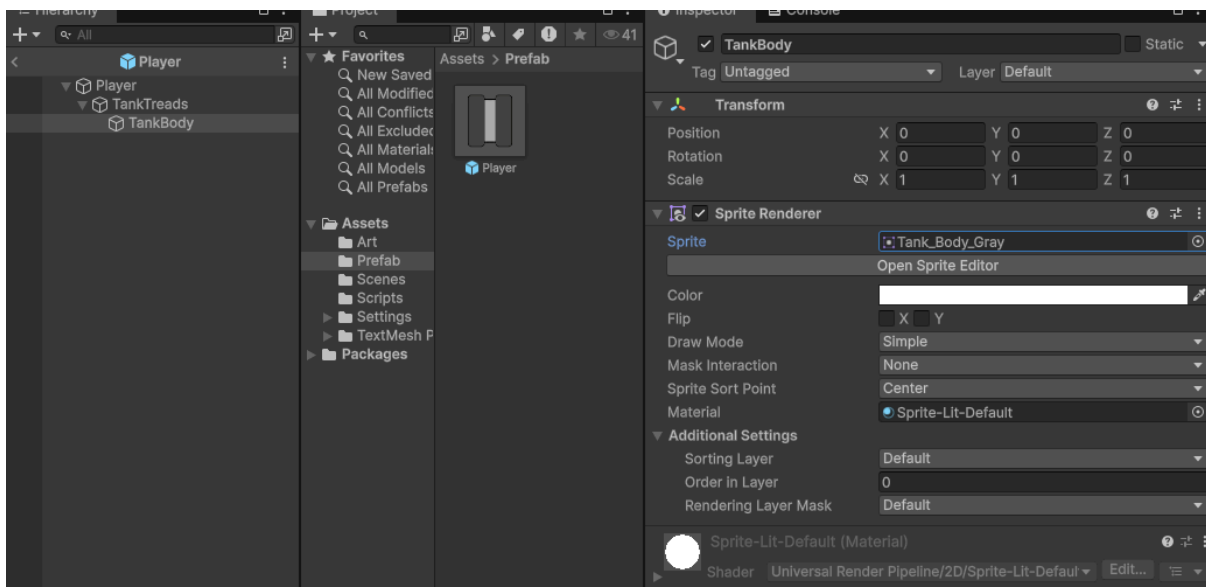
1. Crear Objeto Hijo para el Cuerpo

En la jerarquía seleccionamos las ruedas y creamos un objeto vacío que renombramos a Tank Body.



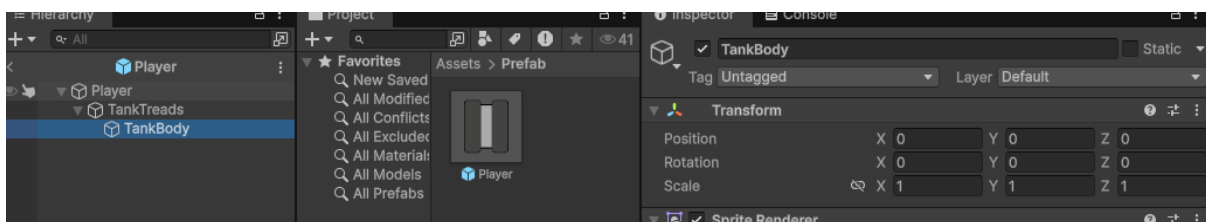
2. Añadir el Sprite Renderer al Cuerpo

Seleccionando Tank Body en la jerarquía, vamos al inspector y le añadimos el componente de Sprite Renderer. Después en Sprite añadimos el TankBody.



3. Posicionar el Cuerpo

Nos aseguramos de que las posiciones X, Y, Z en el Transform estén a 0.



Explicación:

- Al anidar Tank Body dentro de Tank Treads, garantizamos que el cuerpo se mueva junto con las ruedas.

Paso 8: Añadir el Pivote y la Torreta del Tanque

1. Crear el Pivote de la Torreta

En la jerarquía seleccionamos al Player y creamos un objeto vacío que renombramos a Turret Pivot.

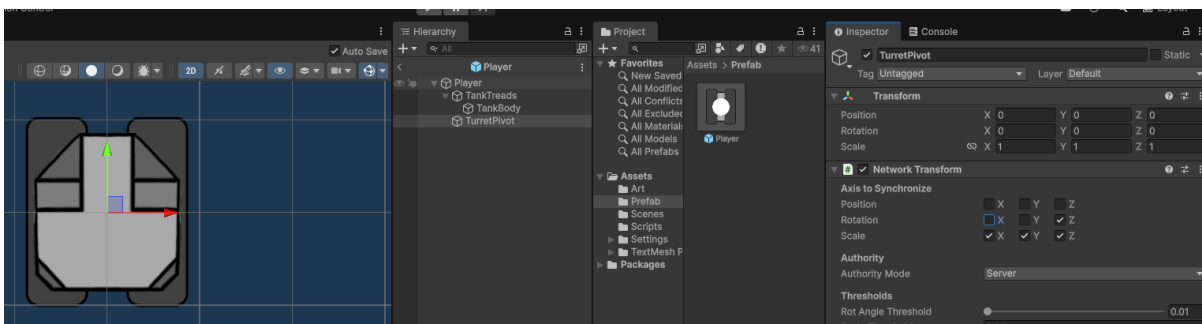
También le añadimos un componente Network Transform que configuramos así:

Desmarcar:

- Sync Position X
- Sync Position Y
- Sync Position Z
- Sync Rot Angle X
- Sync Rot Angle Y

Marcar:

- Sync Rot Angle Z



Explicación:

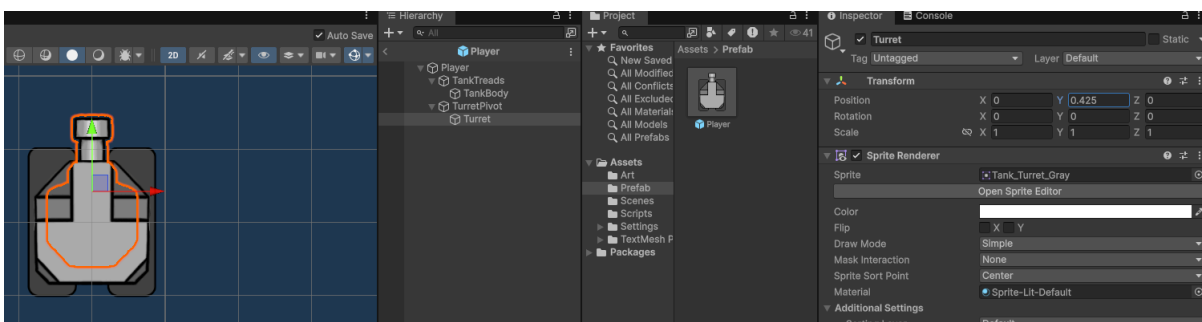
- El Turret Pivot es el punto alrededor del cual rotará la torreta. Sincronizamos su rotación en Z para que todos los jugadores vean hacia dónde apunta.

2. Añadir la Torreta

En la jerarquía seleccionamos Turret Pivot y creamos un objeto vacío que renombramos a Turret.

Le añadimos a este objeto un SpriteRenderer y le asignamos el sprite correspondiente.

Por último posicionamos la Torreta ajustando la posición en Y a 0.425.



Explicación:

- Ajustamos la posición Y para que la torreta se coloque correctamente sobre el cuerpo del tanque.

Paso 9: Configurar el Orden de Renderización

1. Crear una Sorting Layer para el Jugador

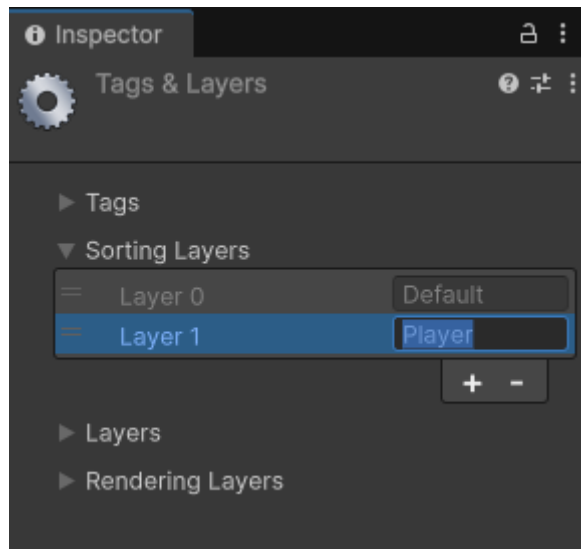
En el panel Inspector, con cualquier Sprite Renderer seleccionado, buscamos la opción Sorting Layer.

Hacemos clic en el menú desplegable y seleccionamos Add Sorting Layer

En la ventana de Tags and Layers, en Sorting Layers, hacemos clic en el botón +.

Añadimos una nueva capa llamada Player.

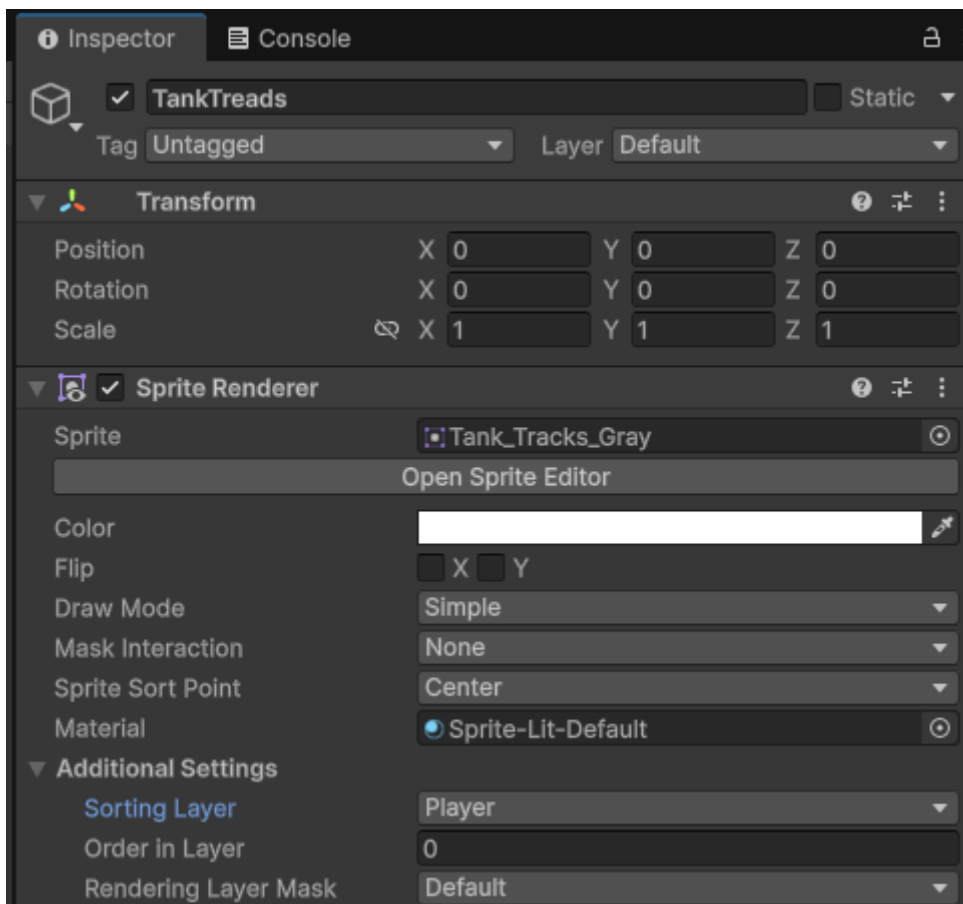
Por último cerramos la ventana.



2. Asignar la Sorting Layer y Orden a los Sprites

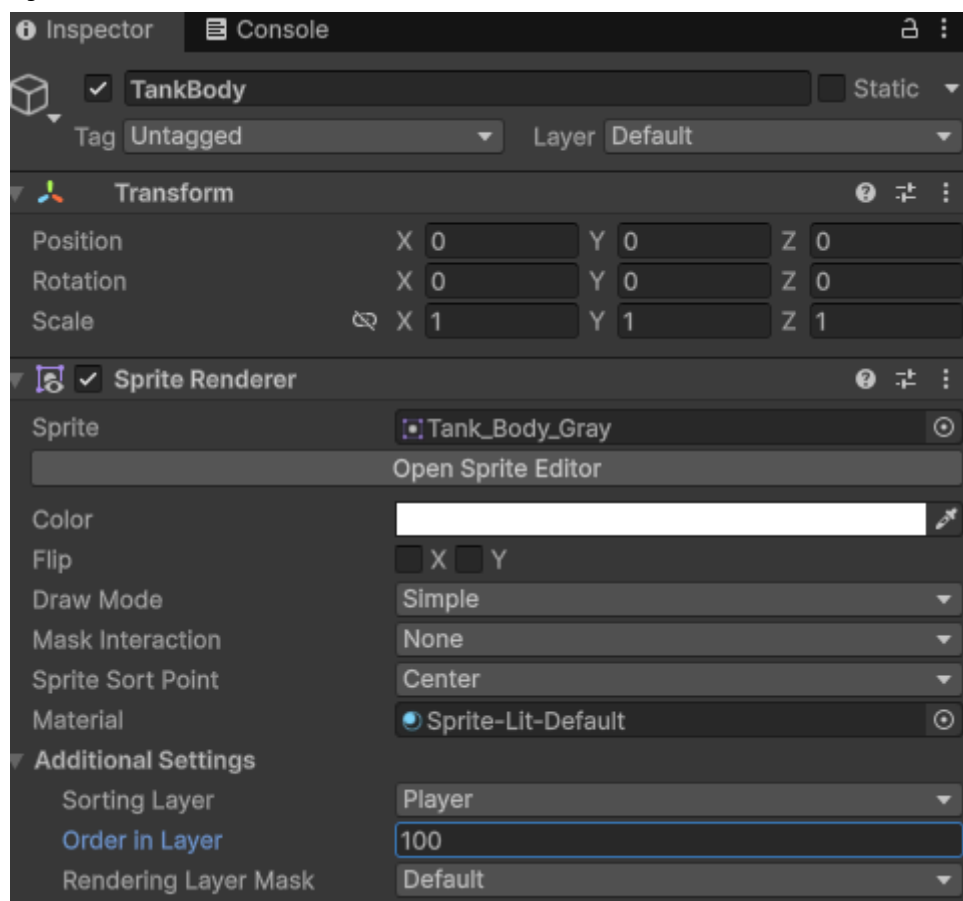
Tank Treads:

- Sorting Layer: Player
- Order in Layer:0



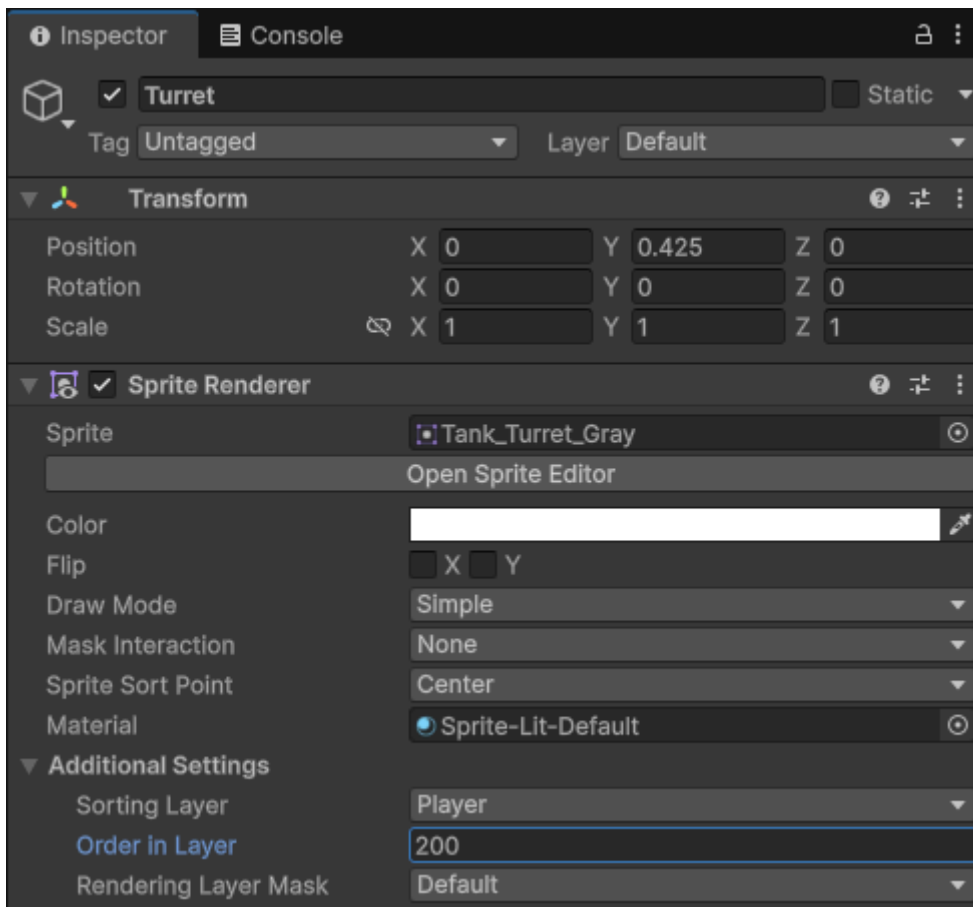
Tank Body:

- **Sorting Layer: Player**
- **Order in Layer:100**



Turret:

- Sorting Layer: Player
- Order in Layer:200



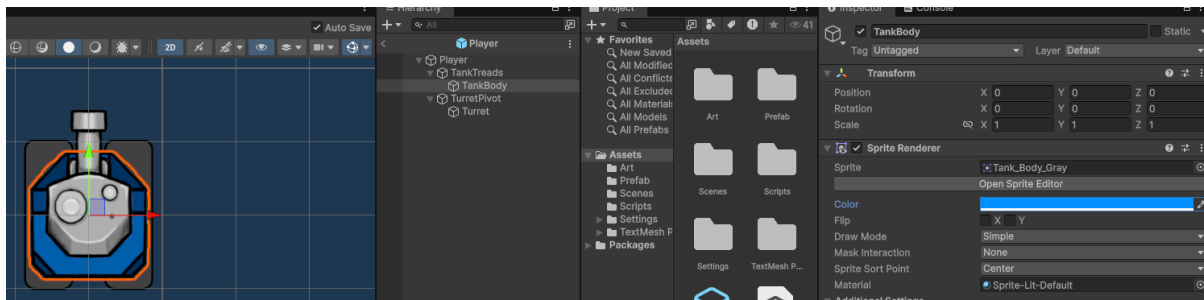
Explicación:

- El Sorting Layer y Order in Layer determinan el orden en que se dibujan los sprites en pantalla.
- Al asignar órdenes más altos, los sprites se dibujarán sobre aquellos con órdenes más bajos.
- Esto asegura que la torreta se dibuje sobre el cuerpo, y el cuerpo sobre las ruedas.

Paso 10: Personalizar el Color del Tanque

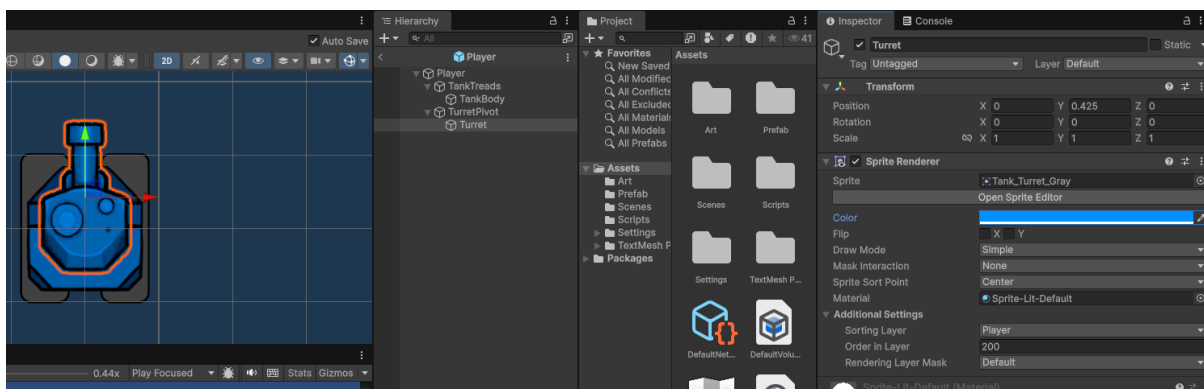
1. Cambiar el Color del Cuerpo

Seleccionamos Tank Body y en el Sprite Renderer, hacemos clic en el cuadro de color al lado de Color. Seleccionamos un color, por ejemplo, el azul.



2. Aplicar el Mismo Color a la Torreta

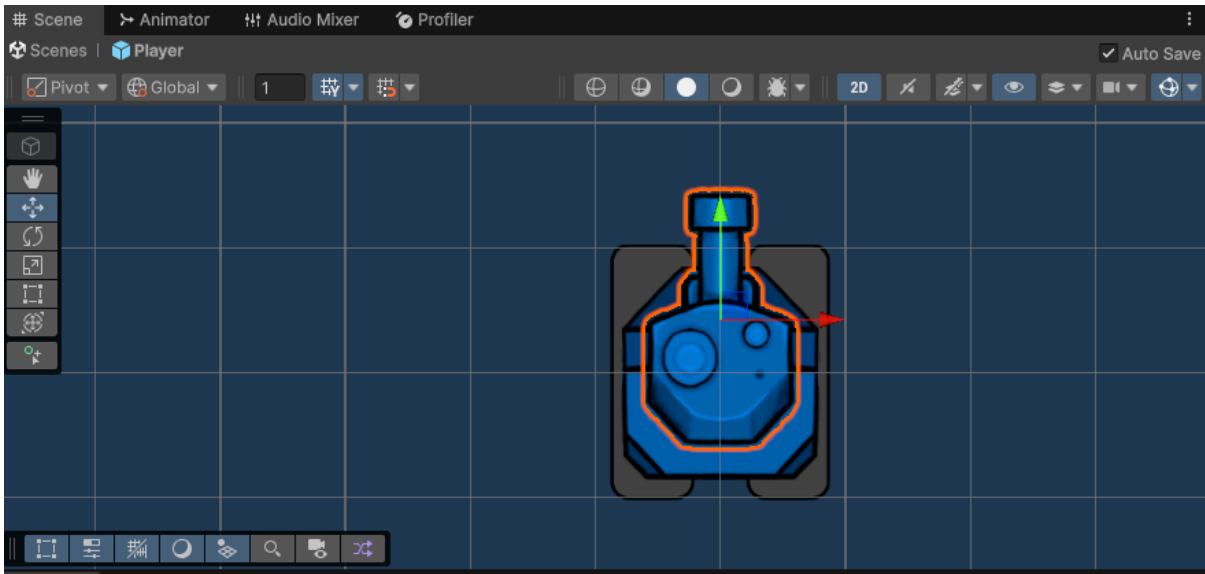
Del mismo modo copiamos el código de color hexadecimal anterior y se lo aplicamos a la Torreta para que tenga coherencia visual.



De momento dejamos el resto de piezas del tanque del color original.

Paso 11: Guardar y Salir del Modo Prefab

Nos aseguramos antes de salir del Prefab de que está guardado, si está activado AutoSave lo hará automáticamente.



Paso 12: Probar el Tanque en el Juego

Antes de poder probar si funciona el proyecto, he notado que nos falta meter un Script en el prefab de Player, o lo que es lo mismo, del tanque, para que podamos moverlo.

En mi caso he decidido hacer que pulsando las teclas W y S, vaya adelante y atrás respectivamente. Para girarlo se usan las teclas Q y E, pero para girar la torreta independientemente se usan las teclas I y P.

El código queda auto-explicado en la imagen. Hacemos las asignaciones pertinentes en el script, asignándolo previamente al objeto padre.

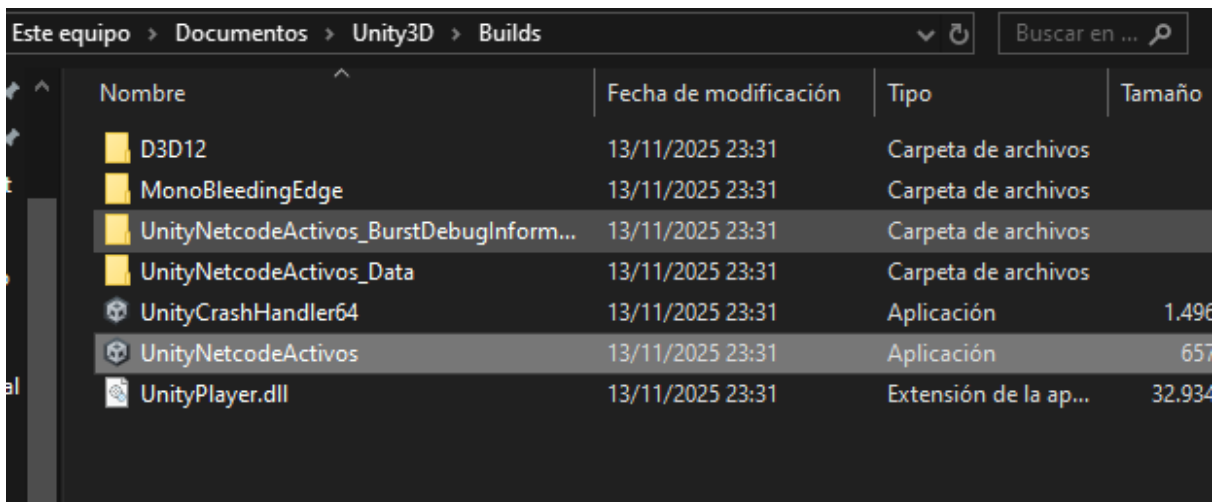
```

TankController2D.cs  JoinServer.cs
Assembly-CSharp  TankController2D  turretRotationSpeed

1  using UnityEngine;
2
3  Script de Unity (1 referencia de recurso) | 0 referencias
4  public class TankController2D : MonoBehaviour
5  {
6      [Header("Movimiento del tanque")]
7      public float moveSpeed = 5f; // Velocidad de avance / retroceso
8      public float rotationSpeed = 180f; // Velocidad de giro del tanque (cuerpo) en grados por segundo
9
10     [Header("Torreta")]
11     public Transform turret; // Referencia al punto de pivote de la torreta
12     public float turretRotationSpeed = 200f; // Velocidad de giro de la torreta
13
14     private Rigidbody2D _rb; // Referencia al Rigidbody2D del tanque
15     private float _moveInput; // Input de movimiento (W/S)
16     private float _rotationInput; // Input de rotación del cuerpo (Q/E)
17     private float _turretRotationInput; // Input de rotación de la torreta (I/P)
18
19     Mensaje de Unity | 0 referencias
20     void Awake()
21     {
22         // Obtenemos el componente Rigidbody2D
23         _rb = GetComponent<Rigidbody2D>();
24     }
25
26     Mensaje de Unity | 0 referencias
27     void Update()
28     {
29         // --- MOVIMIENTO ADELANTE / ATRÁS (CUERPO) ---
30         _moveInput = Input.GetAxisRaw("Vertical");
31
32         // --- ROTACIÓN DEL CUERPO (Q / E) ---
33         _rotationInput = 0f;
34
35         // Q = girar cuerpo a la izquierda (antihorario)
36         if (Input.GetKey(KeyCode.Q))
37             _rotationInput = 1f;
38
39         // E = girar cuerpo a la derecha (horario)
40         if (Input.GetKey(KeyCode.E))
41             _rotationInput = -1f;
42
43         // --- ROTACIÓN DE LA TORRETA (I / P) ---
44         _turretRotationInput = 0f;
45
46         // I = girar torreta a la izquierda
47         if (Input.GetKey(KeyCode.I))
48             _turretRotationInput = 1f;
49
50         // P = girar torreta a la derecha
51         if (Input.GetKey(KeyCode.P))
52             _turretRotationInput = -1f;
53     }
54
55     Mensaje de Unity | 0 referencias
56     void FixedUpdate()
57     {
58         // --- MOVIMIENTO DEL CUERPO SEGÚN DONDE MIRA ---
59         // La "punta" del tanque (el Sprite) en este caso mira hacia arriba
60         // Cambiaríamos a transform.right si el sprite mira a la derecha
61         Vector2 forward = transform.up;
62
63         Vector2 newPosition = _rb.position + forward * _moveInput * moveSpeed * Time.fixedDeltaTime;
64         _rb.MovePosition(newPosition);
65
66         // --- ROTACIÓN DEL CUERPO ---
67         float newRotation = _rb.rotation + _rotationInput * rotationSpeed * Time.fixedDeltaTime;
68         _rb.MoveRotation(newRotation);
69
70         // --- ROTACIÓN DE LA TORRETA ---
71         if (turret != null)
72         {
73             // Obtenemos la rotación actual en Z
74             float currentZ = turret.eulerAngles.z;
75
76             // Calculamos la nueva rotación
77             float newTurretZ = currentZ + _turretRotationInput * turretRotationSpeed * Time.fixedDeltaTime;
78
79             // Aplicamos la rotación solo en Z
80             turret.rotation = Quaternion.Euler(0f, 0f, newTurretZ);
81         }
82     }
83 }

```

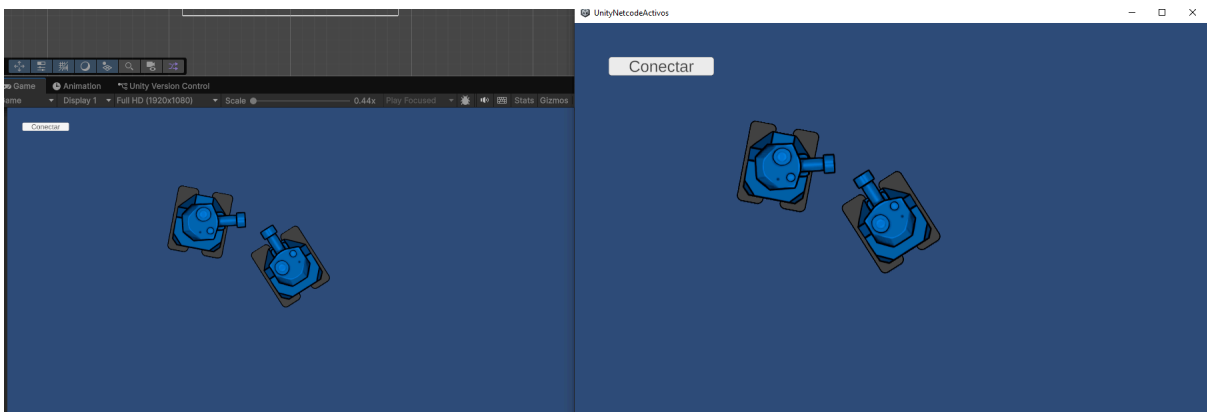
Después en File, podemos hacer Build and Run después de haber creado una carpeta para las Builds, y podemos así probar nuestro juego.



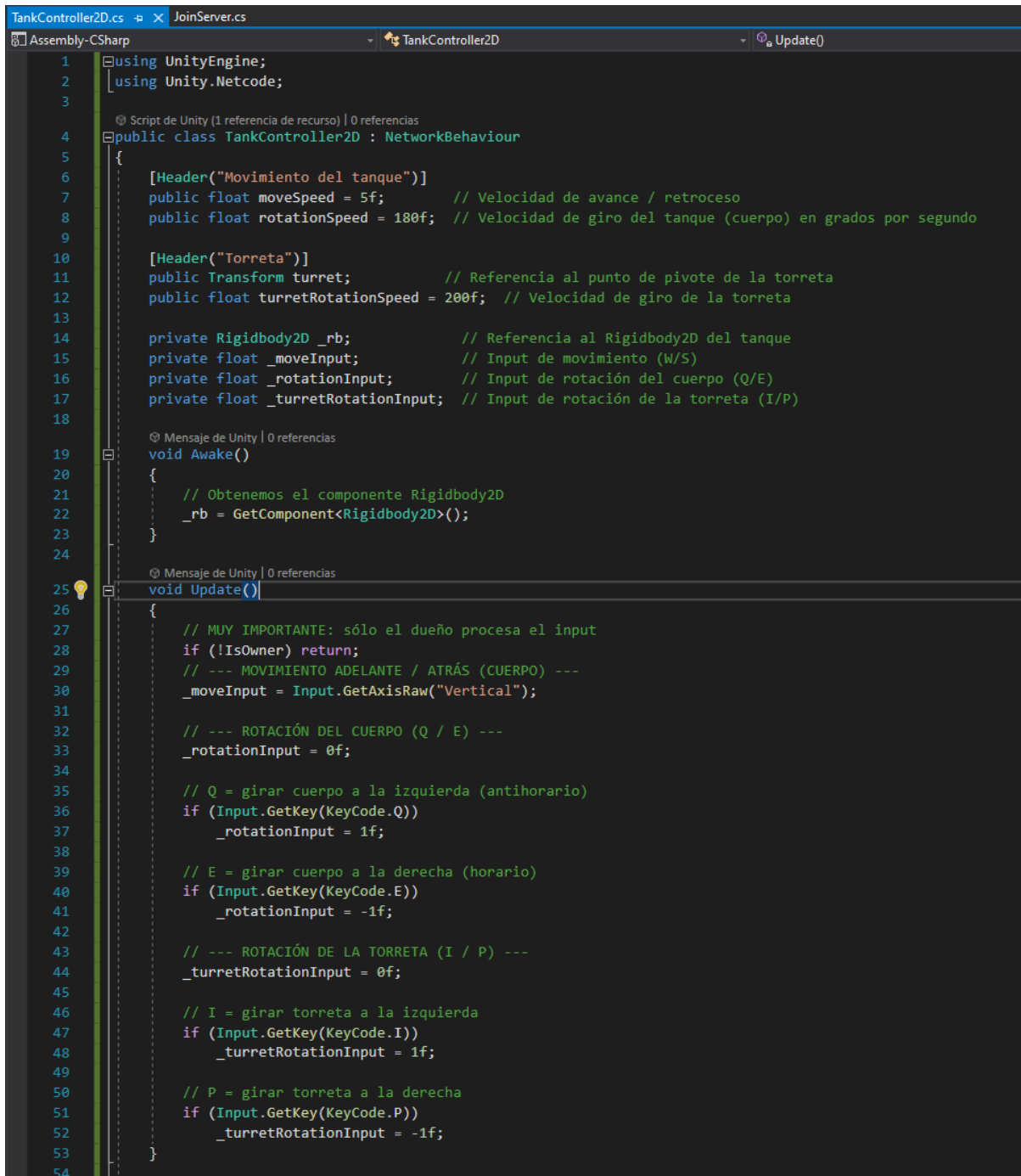
Cuando se ejecute debemos recordar pulsar en Start Host para iniciar el juego como anfitrión.



Luego probamos a conectar otro cliente y ver la sincronización en red.



Aunque el movimiento es el correcto, en este punto nos damos cuenta de que el script de movimiento que llevan los tanques, hace que desde cada cliente del juego se puedan mover todos los tanques a la vez, por lo que debemos modificar levemente el script de la siguiente manera:



```

1  using UnityEngine;
2  using Unity.Netcode;
3
4  [Script de Unity (1 referencia de recurso) | 0 referencias]
5  public class TankController2D : NetworkBehaviour
6  {
7      [Header("Movimiento del tanque")]
8      public float moveSpeed = 5f; // Velocidad de avance / retroceso
9      public float rotationSpeed = 180f; // Velocidad de giro del tanque (cuerpo) en grados por segundo
10
11     [Header("Torreta")]
12     public Transform turret; // Referencia al punto de pivote de la torreta
13     public float turretRotationSpeed = 200f; // Velocidad de giro de la torreta
14
15     private Rigidbody2D _rb; // Referencia al Rigidbody2D del tanque
16     private float _moveInput; // Input de movimiento (W/S)
17     private float _rotationInput; // Input de rotación del cuerpo (Q/E)
18     private float _turretRotationInput; // Input de rotación de la torreta (I/P)
19
20     [Mensaje de Unity | 0 referencias]
21     void Awake()
22     {
23         // Obtenemos el componente Rigidbody2D
24         _rb = GetComponent<Rigidbody2D>();
25     }
26
27     [Mensaje de Unity | 0 referencias]
28     void Update()
29     {
30         // MUY IMPORTANTE: sólo el dueño procesa el input
31         if (!IsOwner) return;
32         // --- MOVIMIENTO ADELANTE / ATRÁS (CUERPO) ---
33         _moveInput = Input.GetAxisRaw("Vertical");
34
35         // --- ROTACIÓN DEL CUERPO (Q / E) ---
36         _rotationInput = 0f;
37
38         // Q = girar cuerpo a la izquierda (antihorario)
39         if (Input.GetKey(KeyCode.Q))
40             _rotationInput = 1f;
41
42         // E = girar cuerpo a la derecha (horario)
43         if (Input.GetKey(KeyCode.E))
44             _rotationInput = -1f;
45
46         // --- ROTACIÓN DE LA TORRETA (I / P) ---
47         _turretRotationInput = 0f;
48
49         // I = girar torreta a la izquierda
50         if (Input.GetKey(KeyCode.I))
51             _turretRotationInput = 1f;
52
53         // P = girar torreta a la derecha
54         if (Input.GetKey(KeyCode.P))
55             _turretRotationInput = -1f;
56     }
57

```

```

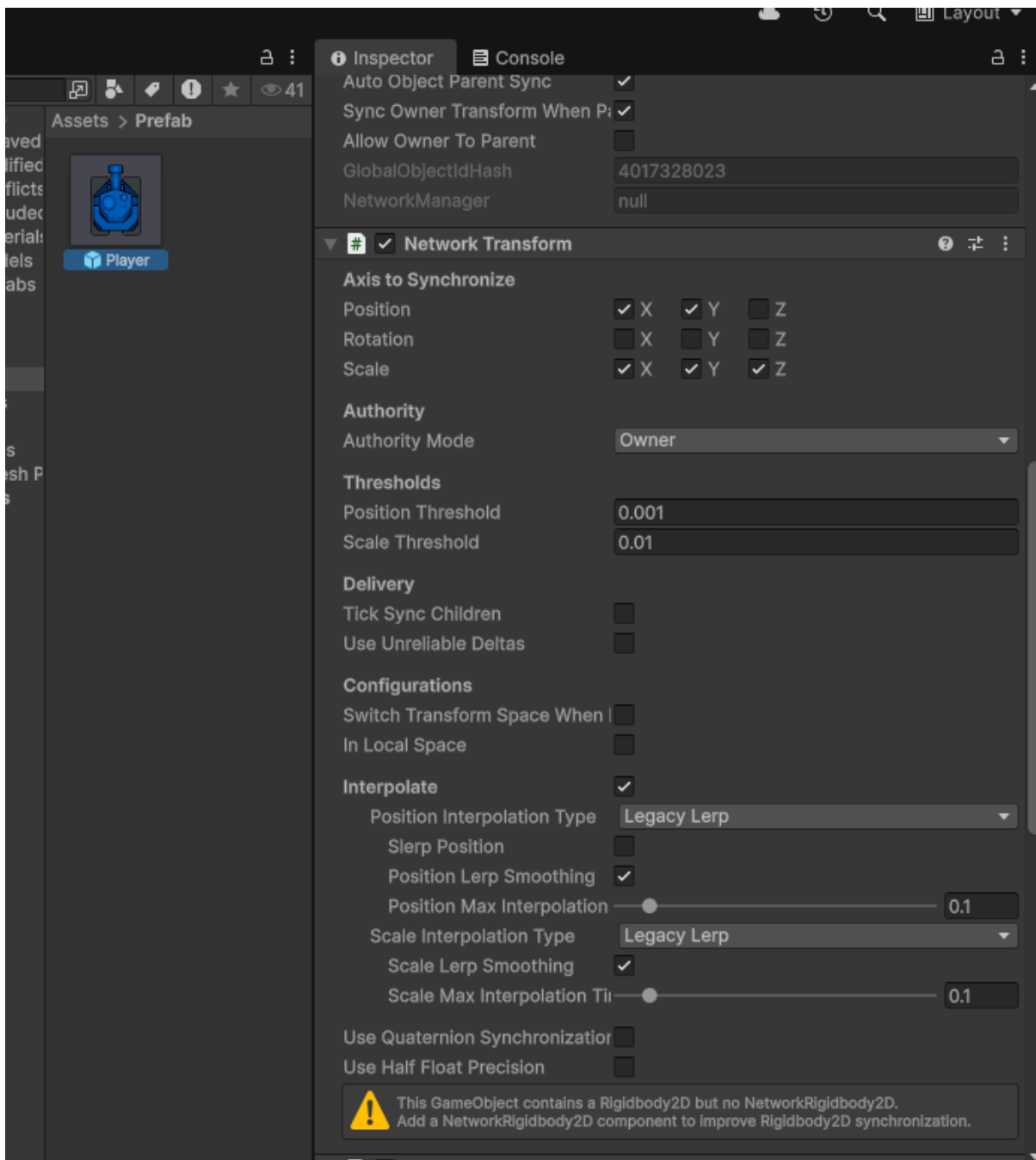
55 void FixedUpdate()
56 {
57     // Sólo el dueño mueve físicamente el tanque
58     if (!IsOwner) return;
59     // --- MOVIMIENTO DEL CUERPO SEGÚN DONDE MIRA ---
60     // La "punta" del tanque (el Sprite) en este caso mira hacia arriba
61     // Cambiaríamos a transform.right si el sprite mira a la derecha
62     Vector2 forward = transform.up;
63
64     Vector2 newPosition = _rb.position + forward * _moveInput * moveSpeed * Time.fixedDeltaTime;
65     _rb.MovePosition(newPosition);
66
67     // --- ROTACIÓN DEL CUERPO ---
68     float newRotation = _rb.rotation + _rotationInput * rotationSpeed * Time.fixedDeltaTime;
69     _rb.MoveRotation(newRotation);
70
71     // --- ROTACIÓN DE LA TORRETA ---
72     if (turret != null)
73     {
74         // Obtenemos la rotación actual en Z
75         float currentZ = turret.eulerAngles.z;
76
77         // Calculamos la nueva rotación
78         float newTurretZ = currentZ + _turretRotationInput * turretRotationSpeed * Time.fixedDeltaTime;
79
80         // Aplicamos la rotación solo en Z
81         turret.rotation = Quaternion.Euler(0f, 0f, newTurretZ);
82     }
83 }
84
85
86

```

Lo que hemos necesitado cambiar aquí es añadir la nueva librería de Netcode, para que la clase ahora sea de NetworkBehaviour y así poder usar IsOwner que nos permite conocer si el objeto que ejecuta el script es el que está en el cliente o host concreto que tengamos seleccionado en ese momento y en ejecución.

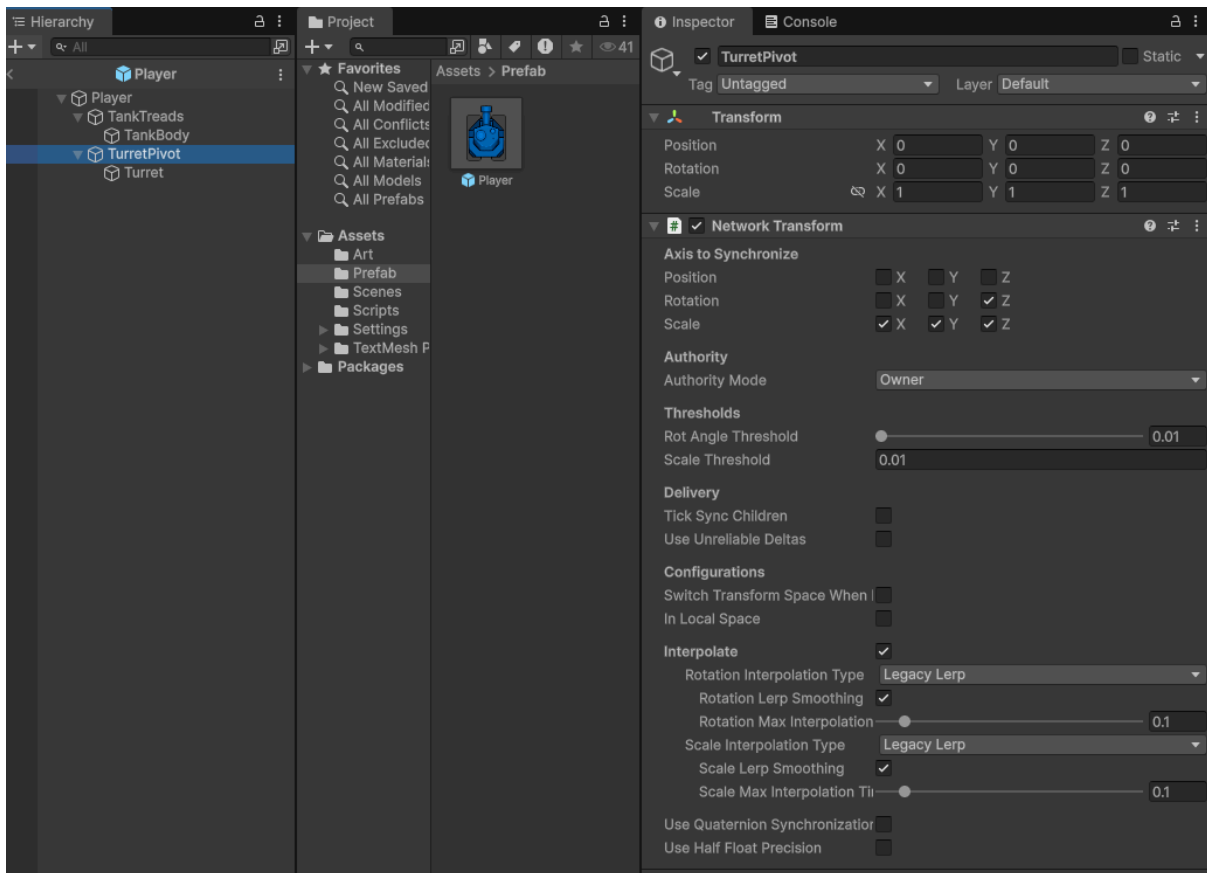
Esto acarrea el siguiente problema, que ahora efectivamente en el Host solo podemos mover el tanque que pertenece a este Host, pero en el Cliente, ya no podemos mover ni el tanque perteneciente al Cliente ni el del Host.

La solución en este caso pasa por modificar el prefab de Player y en su Network Transform/Authority Mode pasarlo de Server a Owner, para que el control no solo sea por parte del host sino de los clientes. O lo que es lo mismo, cada cliente será owner (dueño) de su tanque.



Por último queda un problema por resolver, y es que a pesar de que ahora ya podemos mover cada Tanque en su instancia del juego por separado, en el cliente los Sprites del tanque no rotan en ningún caso, como si lo hacen en el Host.

El problema es el mismo de antes: el prefab de Player tiene dentro hijos que hemos configurado antes con sus propios Network Transform en Authority Mode Server, y hay que cambiarlos a Owner, además si nos fijamos no es necesario que TankTreads tenga un Network Transform, ya que por como hemos configurado el script de movimiento del tanque, las ruedas se mueven y giran cuando lo hace el player o lo que es lo mismo el tanque, pero si mantenemos el Network Transform de la torreta ya que se mueve de manera separada al tanque.

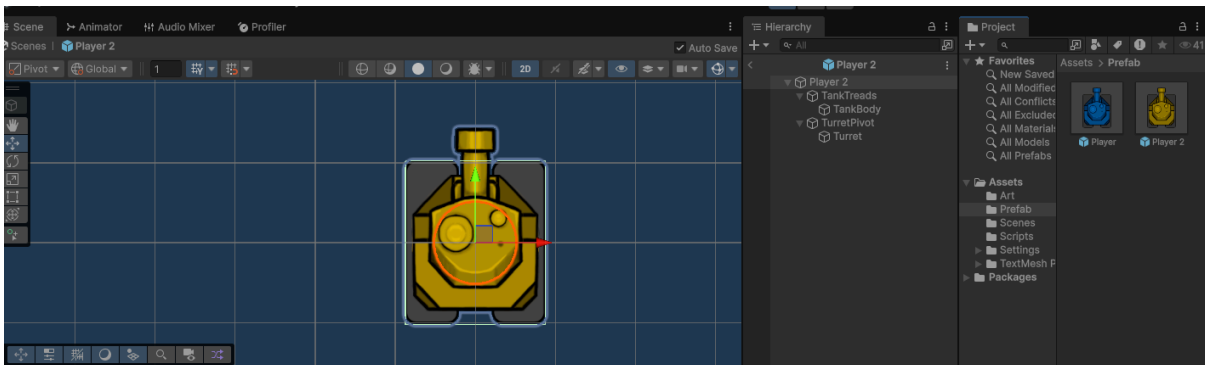


Con esto ya habremos solucionado los problemas y cada tanque se moverá por separado en su cliente, y también se respetarán las rotaciones y movimiento de los tanques respectivos.

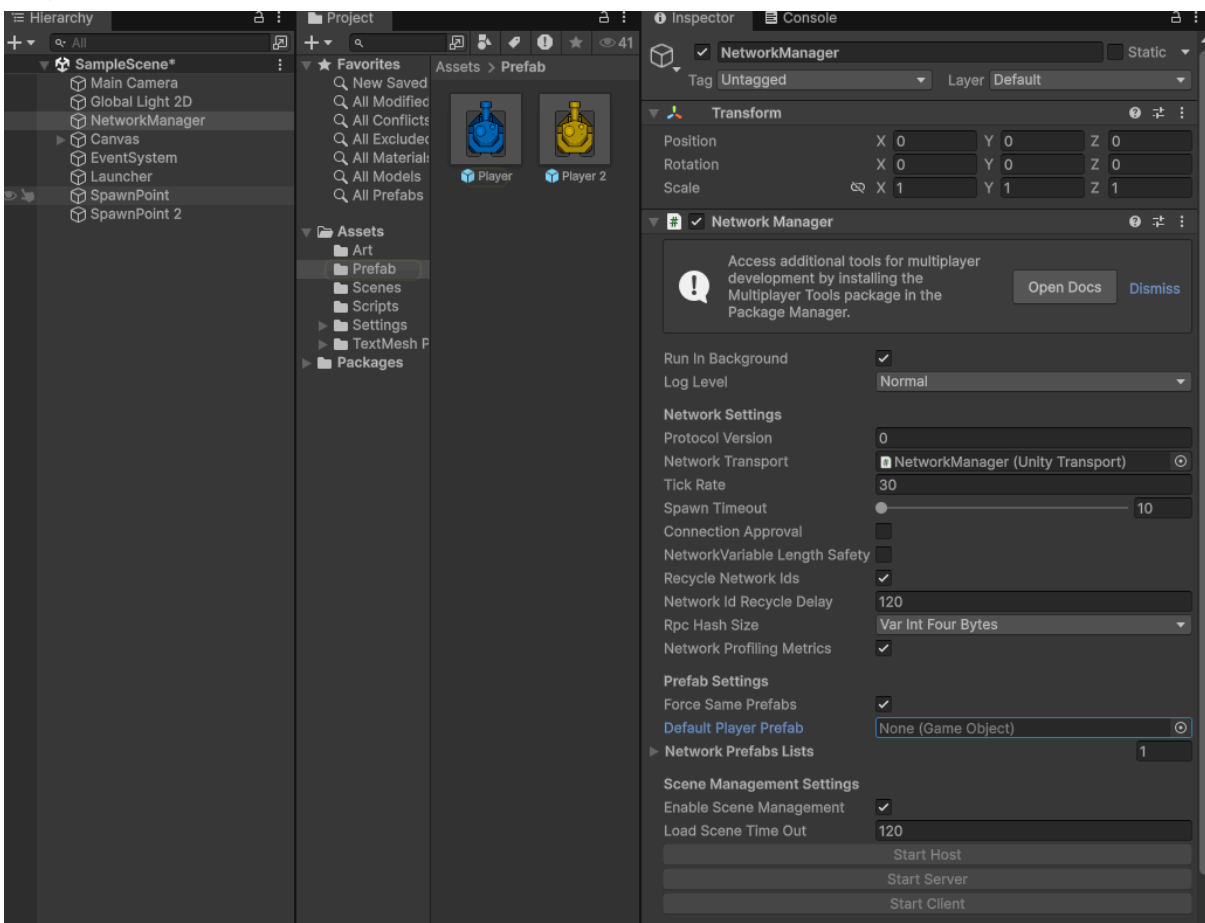
Paso 13: Desafío de Diseño (Opcional)

Personaliza tu Tanque o Crea un Nuevo Vehículo

En este caso he creado un segundo Prefab de Player, renombrado a Player2, con la idea que de que cada cliente (en este caso limitado a dos clientes), tenga cada uno su propio tanque distinto. En este caso solo he duplicado el prefab inicial del Player y le he cambiado el nombre y los colores.



Después y para que esto funcione he tenido que eliminar el prefab del Player del Network Manager.



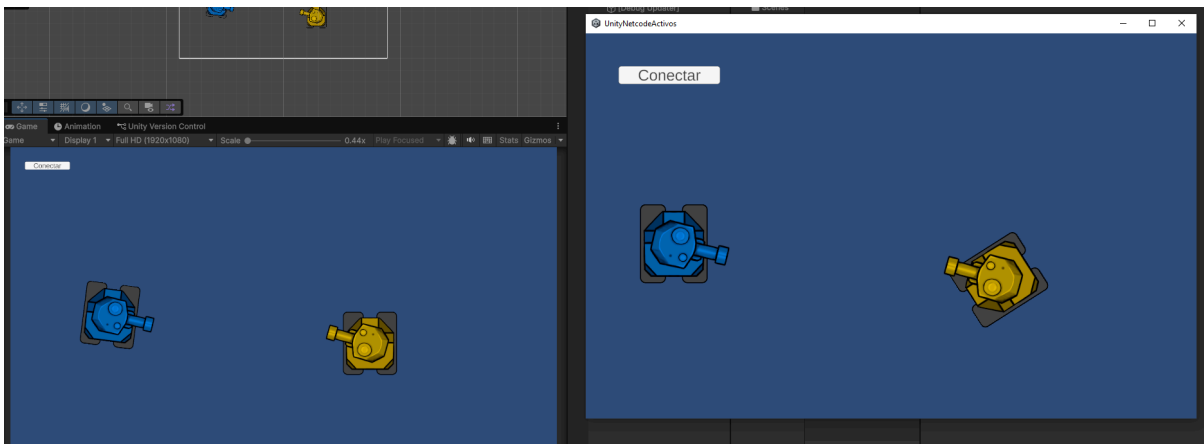
Y también crear un nuevo Script llamado NetCodeLauncher que me permite sacar otro player distinto si ya hay uno en alguno de los servidores, esto se podría escalar. Después he creado el objeto Launcher y le he asociado el script y sus referencias, entre las que he incluido dos SpawnPoint distintos dependiendo del tanque prefab que queramos hacer aparecer, así no aparecen uno sobre el otro.

El contenido del script y el resultado final se pueden ver a continuación:

```

NetCodeLauncher.cs | TankController2D.cs | JoinServer.cs
Assembly-CSharp | NetCodeLauncher
1  using System.Linq;
2  using UnityEngine;
3  using UnityEngine.Networking;
4
5  Script de Unity (1 referencia de recurso) | 0 referencias
6  public class NetCodeLauncher : MonoBehaviour
7  {
8      public NetworkObject playerOnePrefab;
9      public NetworkObject playerTwoPrefab;
10     public Transform spawnPointOne;
11     public Transform spawnPointTwo;
12
13     Mensaje de Unity | 0 referencias
14     void OnEnable()
15     {
16         NetworkManager.Singleton.OnClientConnectedCallback += OnClientConnected;
17     }
18
19     Mensaje de Unity | 0 referencias
20     void OnDisable()
21     {
22         if (NetworkManager.Singleton != null)
23             NetworkManager.Singleton.OnClientConnectedCallback -= OnClientConnected;
24     }
25
26     2 referencias
27     private void OnClientConnected(ulong clientId)
28     {
29         // Solo el servidor/host spawna jugadores
30         if (!NetworkManager.Singleton.IsServer) return;
31
32         // Número de clientes conectados en este instante
33         int numClients = NetworkManager.Singleton.ConnectedClientsIds.Count;
34
35         NetworkObject prefabToSpawn;
36         Transform spawnPoint;
37
38         // Primer cliente -> Player 1, segundo -> Player 2
39         if (numClients == 1)
40         {
41             prefabToSpawn = playerOnePrefab;
42             spawnPoint = spawnPointOne;
43         }
44         else
45         {
46             prefabToSpawn = playerTwoPrefab;
47             spawnPoint = spawnPointTwo;
48         }
49
50         // Instanciamos y lo marcamos como PlayerObject de ese clientId
51         NetworkObject playerInstance = Instantiate(prefabToSpawn, spawnPoint.position, spawnPoint.rotation);
52         playerInstance.SpawnAsPlayerObject(clientId);
53     }
54 }

```



Conclusión

En esta práctica, hemos logrado:

- Importar y organizar los activos proporcionados.
- Configurar el tanque con sus ruedas, cuerpo y torreta.
- Añadir y configurar los componentes Network Transform para sincronizar movimientos y rotaciones en red.
- Personalizar el aspecto visual del tanque.
- Comprender la función de cada componente añadido y cómo contribuye al funcionamiento del juego.