Informe da proposta de solución

Práctica 3.4.- Jump multiplayer con Network Transform

Requisitos

1. Main project segundo o Manual de Unity: Get started with NGO
2. Utilizar NetworkTransform para todas as instancias de *Player* para esta práctica.
3. Que todo o código sexa eficiente e teña un propósito, eliminando as liñas redundantes.

Obxectivo

Crea un xogo multixogador no que os xogadores poidan moverse sobre o plano (arriba, abaixo, esquerda e dereita) e que poidan saltar (ou semellante). Faino usando o Network transform.

Fai que o código sexa o máis eficiente posible, sen partes que sobren, sen código redundante ou sen partes que non se usen nunca. Podes asumir que o código se vai executar sempre con Host e nunca con server.

Renomear scripts

Xa que imos optimizar o código e desfacernos dos métodos de propósito didáctico, comezaremos por darlles ás clases un nome axeitado

```
sonia@pria:~/p3_ngo$ git commit -m "Rename scripts & gameObject GameManager"
[test/InputJump c8bb2e7] Rename scripts & gameObject GameManager
 8 files changed, 235 insertions(+), 233 deletions(-)

 create mode 100644 Assets/Scripts/GameManager.cs
 rename Assets/Scripts/{HelloWorldManager.cs.meta => GameManager.cs.meta} (100%)
 delete mode 100644 Assets/Scripts/HelloWorldManager.cs
 delete mode 100644 Assets/Scripts/HelloWorldPlayer.cs
 create mode 100644 Assets/Scripts/Player.cs
 rename Assets/Scripts/{HelloWorldPlayer.cs.meta => Player.cs.meta} (100%)
```

NetworkTransform

Referencia de Unity: NetworkTransform

Necesito probar a Práctica 2 (Movemento multiplayer en rede) agora usando soamente o NetworkTransform en vez de a Network variable Position para me asegurar de que a rama funciona igual que a de orixe.

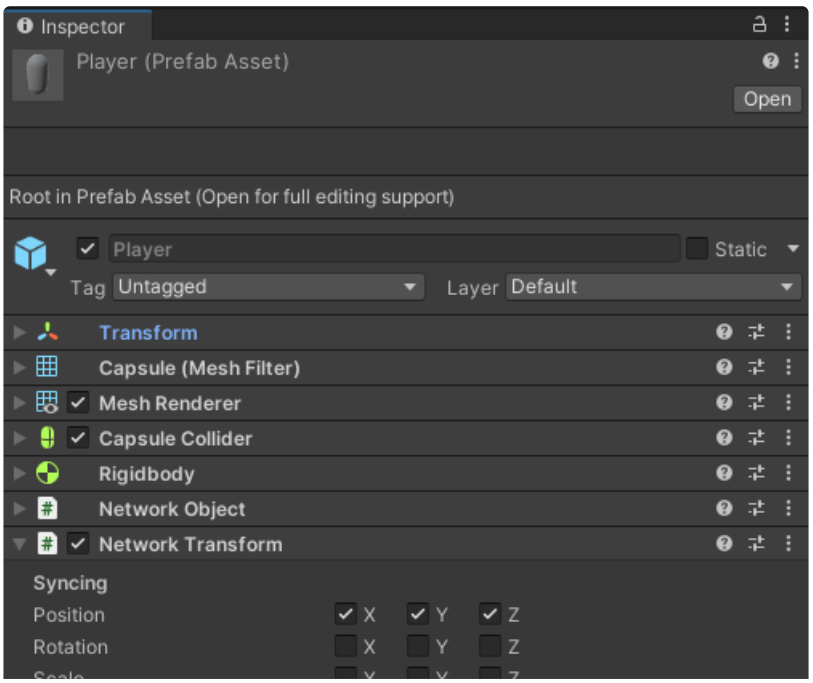
Aproveitamos e simplificamos o método Move() eliminando a condición IsServer, dado que, si un Player está nun equipo Server (o sea un Host), tamén executa os métodos ServerRpc

```
public void Move()
{
    // if (NetworkManager.Singleton.IsServer)
    // {
    //     Position.Value = GetRandomPositionOnPlane();
    // }
    // else
    // {
        SubmitPositionRequestServerRpc();
    // }
}
```

Synzing

Para aforrar CPU e ancho de banda, só sincronizamos en rede os valores para os transform.position que son os eixes que o Server necesita ter sincronizados, o X | Z para o movimento 2D e Y para o Jump()

Para mantemos activados no apartado **Syzing** as 3 variables do vector de transform.position (desactivamos o resto)



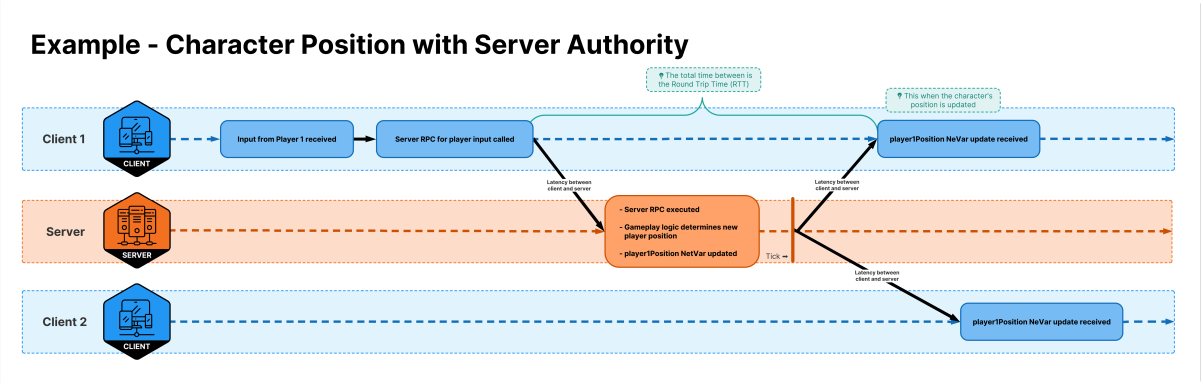
Player.cs I

> **Only the owner can invoke a ServerRpc**
>
> Only the owner can invoke a ServerRpc that requires ownership!
> UnityEngine.Debug:LogError (object)

History Player.cs (commit 60f621f)

Comprobado. Funciona tan ben ou tan mal coma antes. Sen Network variables

Imos traballar co Network Transform baixo autoridade de Server



Jump no cliente

Imos convervir o método Move() en Jump(). Cando se pulse o botón nun equipo cliente esa instancia de player saltará `jumpForce` m. (presente!), cando o pulse o Servidor, tódalas instancias saltarán.

O xeito máis doado de implementar un salto en Unity é empregando as físicas de Rigidbody

TODO Referencia: Unity Multiplayer Networking Physics

No Update de Player.cs

Player.cs

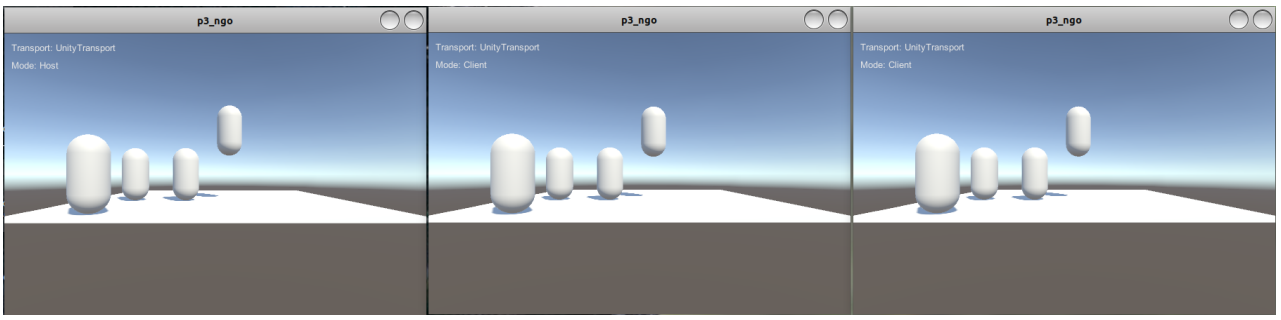
```
void Update()
{
    if (!IsOwner) { return; }

    ...

    if (Input.GetButtonDown("Jump"))
    {
        SubmitJumpRequestServerRpc();
    }
}
```

No ServerRpc

Player.cs

```
[ServerRpc]
void SubmitJumpRequestServerRpc(ServerRpcParams rpcParams = default)
{
    // Salto
    Rigidbody rb = GetComponent<Rigidbody>();
    rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
}
```

Parece que funciona.



Jump no servidor

Polo de agora, simplemente mudamos a chamada ó metodo Move, polo método Jump

Player.cs

```
public void Jump()
{
    SubmitJumpRequestServerRpc();
}
```

GameManager.cs

```
static void SubmitJump()
{
    if (GUILayout.Button(NetworkManager.Singleton.IsServer
    ? "Everyone jumps" : "I jump"))
    {
        if (NetworkManager.Singleton.IsServer
            && !NetworkManager.Singleton.IsClient)
        {
            foreach
            (ulong uid in NetworkManager.Singleton.ConnectedClientsIds)
            {
                NetworkManager.Singleton.SpawnManager
                    .GetPlayerNetworkObject(uid)
                    .GetComponent<Player>()
                    .Jump();
            }
        }
        else
        {
            var playerObject = NetworkManager.Singleton.SpawnManager
                .GetLocalPlayerObject();
            var player = playerObject.GetComponent<Player>();
            player.Jump();
        }
    }
}
```

Error do lado do Servidor

O GameManager non funciona cando se chama ao evento "Everyone jumps" dende o servidor porque se !(IsOwner) non ten Rigidbody

Para solucionar esto habería que crear un método JumpClientRpc que non ha lugar para este exercicio.

Lexibilidade

Coma xa non estamos en modo didáctico, cámbianse os nomes dos métodos por algo mái directo, do tipo Accion() e AccionServerRpc().

Eficiencia

1. O servidor non vai mover aos xogadores. Elimínase o método SubmitJump() do GameManager.cs
2. Coma temos o código gardado no Git non precisamos os Debug.Log() nen a función InitValues no Player.cs
3. Ninguén chama ao método Jump() que, ademáis chama a JumpServerRpc(), é redundante.
4. Comprobar que os valores sincronizados no Network Transform son os xustos e necesarios, xa feito máis arriba no apartado Synzing.
5. Elimínase o namespace HelloWorld

Lista de verificación de uso de RPC

Referencia: Sending Events with RPCs

- [ ] O atributo [ServerRpc] está en tódolos métodos
- [ ] O nome dos métodos remata en xxxServerRpc()
- [ ] Os métodos son declarados nunha clase que hereda de NetworkBehaviour
- [ ] Os métodos ServerRpc chámanse do lado cliente
- [ ] Os parámetros pasados son te tipo simple