# Práctica 5 - ClientRcp

## Práctica 5.- Des/vantaxes para os players

*Requisitos*

1. *Partir da práctica 4*
2. *Usar ClientRpc*

*Obxectivo*

*Os players van recibir de forma aleatoria e por tempo limitado unha vantaxe ou desvantaxe.*

- *A vantaxe é que o movemento é máis rápido e pode verse porque o player se pon de cor verde,*
- *a desvantaxe ao contrario, o player ponse vermello ou laranxa e vai máis lento.*

*Por exemplo, pasados 20 seguntos o player 2 recibe unha vantaxe, ponse verde e vai máis rápido por 10 segundos.*

## GameManager.cs I

O GameManager fai as veces de *Anillo Único* e necesitamos

1. Un float para o tempo de arroutada na que se elixe un probe player, p.ex., 20 segundos.
2. Un float para o tempo de *Boon* ou *Bane, p.ex. 10 segundos*
3. Unha lista de players, esta sí, en tempo real multiplayer

```
public class GameManager : MonoBehaviour
{
    List<NetworkClient> players;

    float timeRandom, timePower;

    void Awake()
    {
        timeRandom = 20f;
        timePower = 10f;
    }
    ...
}
```

**Player.cs I**

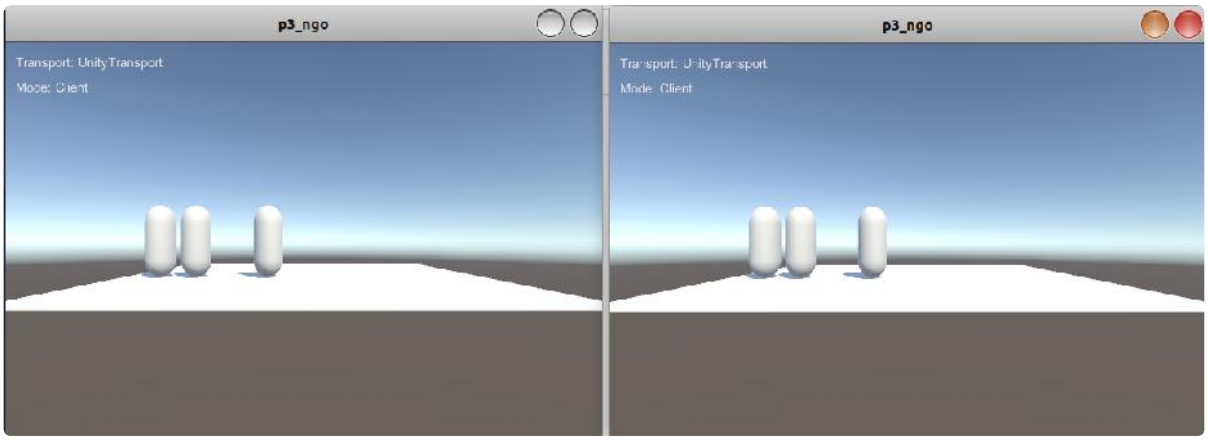
Primeiro cambiaremos un pouco como se inicia o xogo para ver millor a tódolos players. Coma no meu proxecto de base non teño velocidade, hai que agregar algún tipo de movimento para ver as ordes do Game Manager.

Vouno plantexar coma se de unha carreira de vallas se tratase. Cada Player ocupará un carril en position.x que estea libre, comezando en -4 ate +4 ~~e ós se poderán saltar~~.

En principio, para testerar, os players terán un movimento infinito en Z de -4 a +4 con unha velocidade inicial idéntica a tódalas instancias.

Referencia: GitHUB commit 5e00767199

Os xogadores teñen todos o mesmo movemento Smooth Ping Pong infinito




TODO Os player non saltan

## GameManager.cs II

En principio o servidor chama a unha corrutina que chama aos métodos ClientRpc con sendas esperas

```
public override void OnNetworkSpawn()
{
    // Só o servidor pode otorgar premio ou castigo
    if (IsServer) { StartCoroutine(CoRandomBoonBane()); }
}

IEnumerator CoRandomBoonBane()
{
    while (true)
    {
        // pasados 20 seguntos escóllese un netPlayer aleatoriamente
        yield return new WaitForSeconds(timeRandom);

        int rdmIndex = Random.Range(0,
            NetworkManager.Singleton.ConnectedClientsList.Count);
        ulong rdmUID = NetworkManager.Singleton.ConnectedClientsIds[rdmIndex];

        var rdmPlayerObject = NetworkManager.Singleton.SpawnManager
            .GetPlayerNetworkObject(rdmUID);
        var rdmPlayer = rmdPlayerObject.GetComponent<Player>();

        // Método ClientRpc do player des/afortunado
        rdmPlayer.SetBoonBaneClientRpc();

        // Os cambios duran 10s
        yield return new WaitForSeconds(timePower);

        // O netPlayer recupera as propiedades de orixe
        rdmPlayer.ResetBoonBaneClientRpc();
    }

}
```

~~TODO agregar un booleano a SetBoonBaneClientRpc para dictar si é premio ou castigo~~

```
        // Método ClientRpc do player des/afortunado
        bool isBoon = Random.Range(0, 2) == 0 ? true : false;
        rdmPlayer.SetBoonBaneClientRpc(isBoon);
```

**Player.cs II**

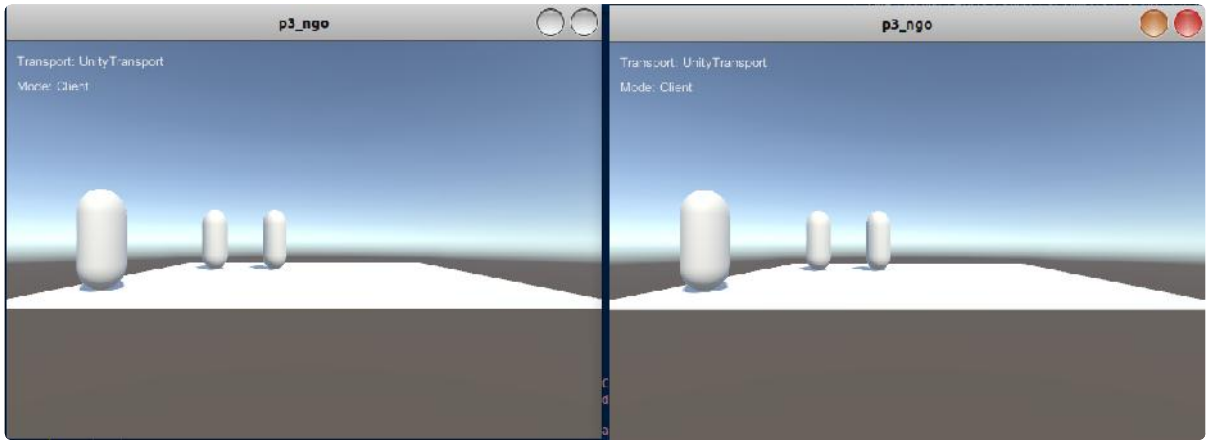
No player hai que implementar ámbolos métodos ClientRPC

Coma a miña práctica anterior non traballaba cas cores, en principio só lle cambiamos o periodo para o Mathf.PingPong()

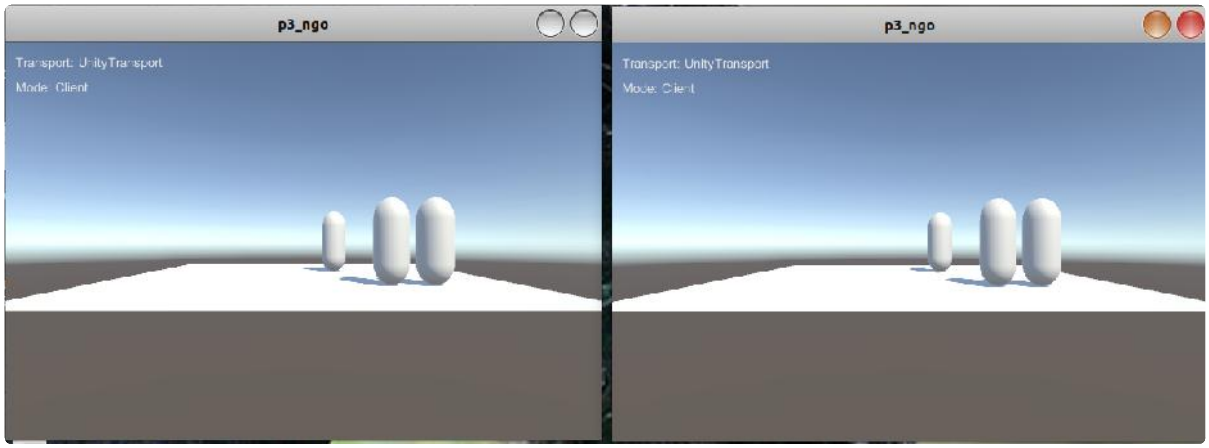
```
NetworkVariable<int> StartingLine = new NetworkVariable<int>();

float jumpForce, period;
float initPeriod, boonPeriod, banePeriod;

public override void OnNetworkSpawn()
{
    DevInfoNetworkObject();

    if (IsOwner)
    {
        Initialize();
    }

    else
    {
        // Valores das instancias de players no equipo cliente
        Debug.Log($"{gameObject.name}.OnNetworkSpawn() IsOwner {IsOwner}");
    }
}

public override void OnNetworkDespawn()
{
}

void Initialize()
{
    // Valores do GameObject Player no equipo cliente
    // Pide ao server que o sitúe no punto de saída
    // nun carril de carreira aleatorio entre os que non están ocupados

    SetStartingLinePositionServerRpc();

    jumpForce = 6f;
    period = 4.5f;
    initPeriod = period;
    boonPeriod = 2f;
    banePeriod = 8f;

    Debug.Log($"{gameObject.name}.OnNetworkSpawn() IsOwner {IsOwner}");
}

...

[ClientRpc]
public void SetBoonBaneClientRpc(bool isBoon,
    ClientRpcParams clientRpcParams = default)
{
    if (isBoon)
    {
        period = boonPeriod;
    }

    else
    {
        period = banePeriod;
    }
```

```
}

// Restablecimento dos valores normais de Play

[ClientRpc]
public void ResetBoonBaneClientRpc(ClientRpcParams clientRpcParams = default)
{
    period = initPeriod;
}
```

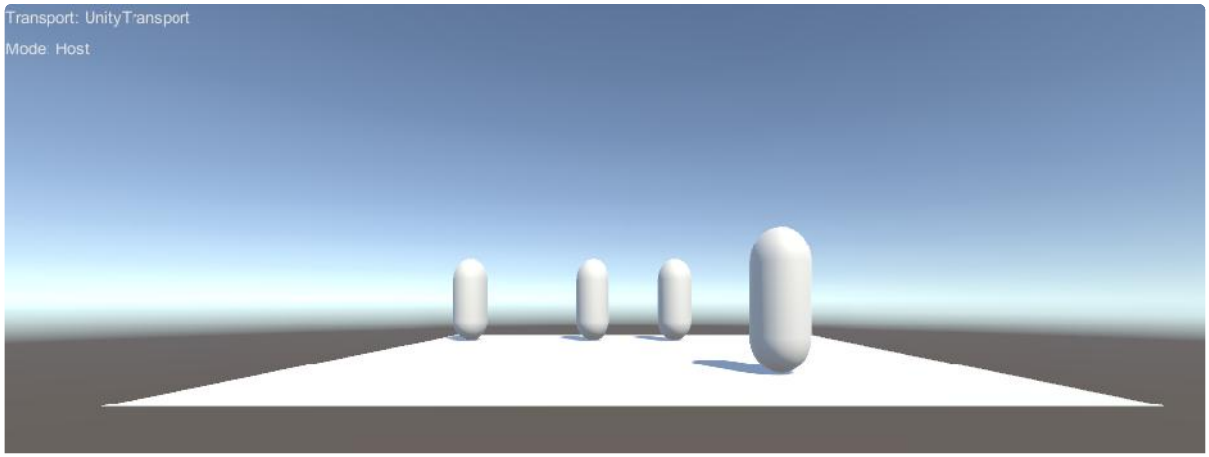
Vale, parece que funciona: ao player 1 tocóulle máis velocidade (menos período)




Probando noutra Build. Agora ao player 1 tocóulle menos velocidade (máis period)




E noutro Play agora tócalle ao player 4

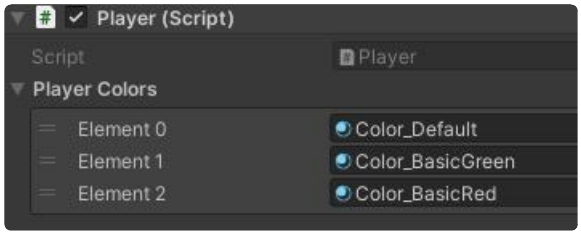


Referencia: Git commit d96fd9e975

## Dirfencias con cores

A miña rama de orixe non ten muda de cor algunha.

Miramos na rama `test/ChangeColo` r, e agregamos ao script de cada Player un array de Materials, para este caso de uso só 3:

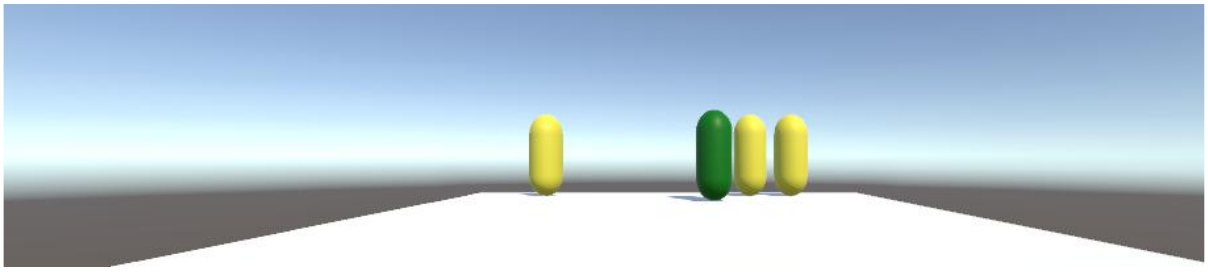



E agregamos o parámetro aos métodos ClientServerRpc

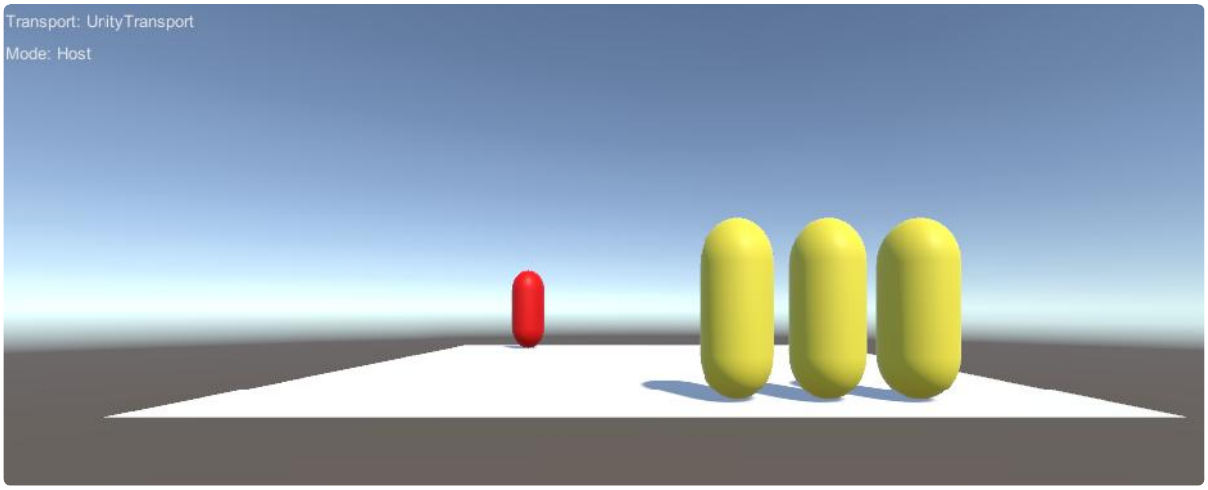
```
[ClientRpc]
public void SetBoonBaneClientRpc(bool isBoon,
    ClientRpcParams clientRpcParams = default)
{
    if (isBoon)
    {
        period = boonPeriod;
        mr.material = playerColors[1];
    }

    else
    {
        period = banePeriod;
        mr.material = playerColors[2];
    }
}

[ClientRpc]
public void ResetBoonBaneClientRpc(ClientRpcParams clientRpcParams = default)
{
    period = initPeriod;
    mr.material = playerColors[0];
}
```

Parece que funciona!

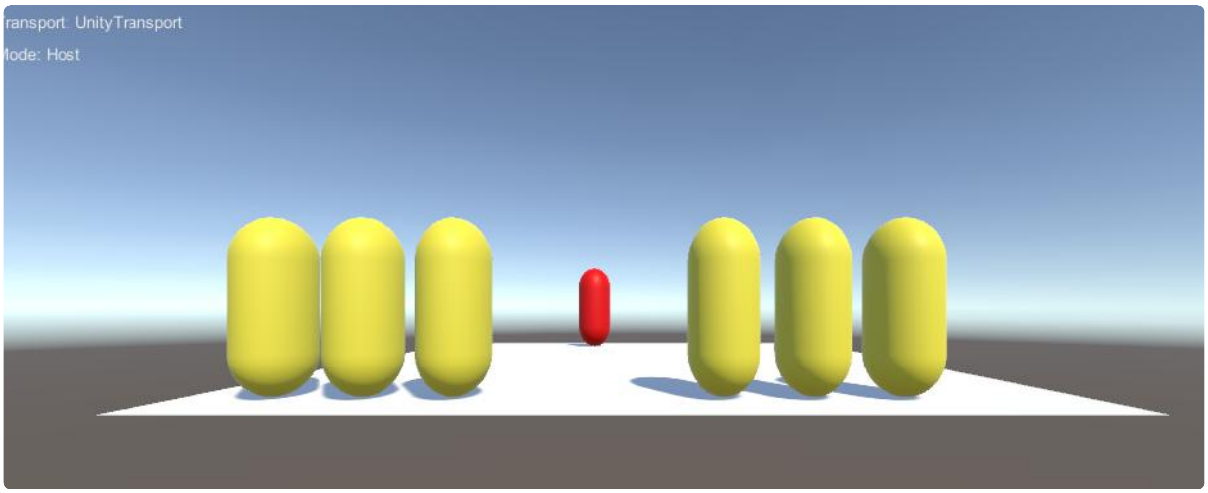
Tocóulle vantaxe ao netPlaer 2

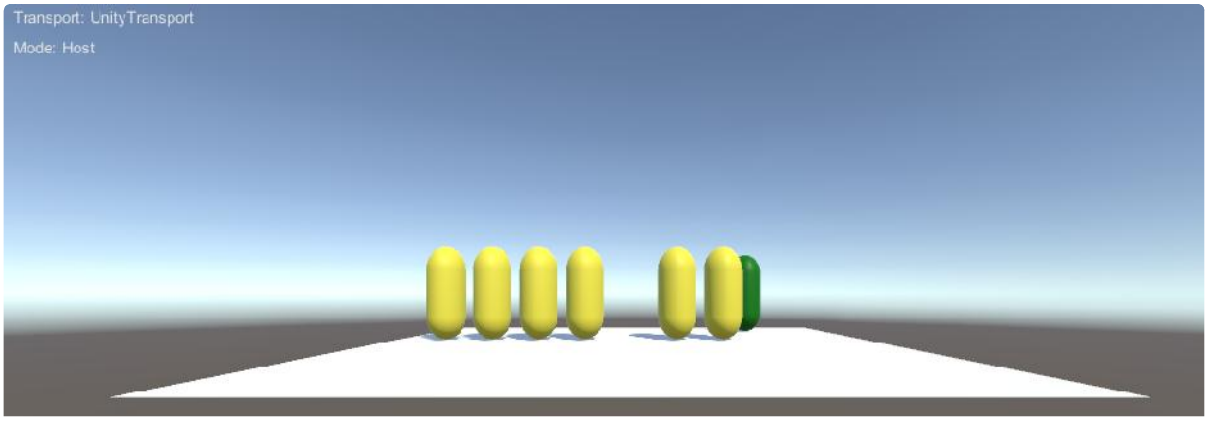


No mesmo Play, agora tócalle desvantaxe ao net Player 1




Capturas con 7 xogadores ou xogadoras:



Parece que ben. Todos van á mesma velocidade e cor, a excepción do Random netPlayer seleccionado polo server.

Cando está penalizado vése en vermello. Cando está premiado, en verde.

Listo