

Proposta de corrección

E2.0 Exame: equipos en rede (Miña corrección)

Sobre este informe

A idea e retroceder nos commits, comentar brevemente os pasos feitos e a corrección aplicada a posteriori

Punto 2

2.1.-

Pártese da base do proxecto creado en clase a partires do manual de Unity [Get started with NGO](#) que teño intacto na rama main do repositorio das prácticas deste tema en clase.

Clonando esa rama créase un novo repositorio para o proxecto do exame.

2.2.-

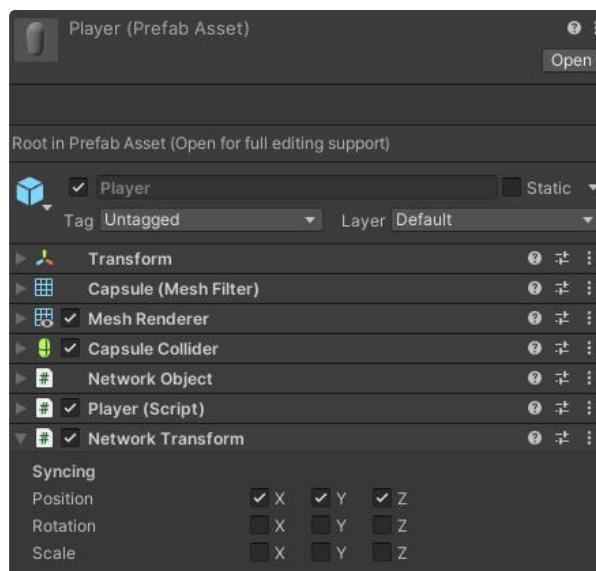
Cámbianselle os nomes aos scripts HelloWorldManager e HelloWorldPlayer, por GameManager e Player respectivamente

Créase unha función ServerRpc para mudar o transform.position de tódalas intancias de Player a través do NetworkTransform capturando o Input dos NetPlayers no Update()

Player.cs

```
[ServerRpc]
void MoveServerRpc(Vector3 direction, ServerRpcParams rpcParams = default)
{
    // Posición enviada por Input de Player
    // Componente Network Transform sincroniza o transform das instancias
    transform.position += direction;
}
```

Sincronízanse soamente os valores do transform.position para aforrar recursos no Player prefab > NetworkTransform > Syncing



Punto 3

3.1.- Espaneo inicial na zona Neutral

Sobreescríbese o método `OnNetworkSpawn()`, cunha condición só para players propietarios, cunha chamada a un método `ServerRpc` que espaeña as capsulas entre -1.5 a 1.5 no eixo X e de -4.5 a 4.5 no eixo Z, mantendo constante o eixo Y en 1.

Player.cs

```
[ServerRpc]
void SetStartPositionServerRpc(ServerRpcParams rpcParams = default)
{
    // Posición aleatoria no taboleiro
    // No espazo central no que o xogador non ten equipo
    transform.position =
        new Vector3(Random.Range(-1.5f, 1.5f), 1f, Random.Range(-4.5f, 5f));
}
```

Nota: Aquí perdín moito tempo porque ocorréuseme a feliz idea de mover o Plano na vista de `Scene`, en vez de mover a cámara.

3.1.- Corrección

`Random.Range()` é un método que inclúe o primeiro parámetro pero exclúe o segundo.

Sería más axeitado agregarlle un factor de corrección, o que sexa, o caso é que se entenda ao ler o código que as liñas imaxinarias son as que son, a saber +/- 1.5 e +/- 4.5 neste caso.

Player.cs

```
[ServerRpc]
void SetStartPositionServerRpc(ServerRpcParams rpcParams = default)
{
    float f2 = 0.1f;
    // Posición aleatoria no taboleiro
    // No espazo central no que o xogador non ten equipo
    transform.position = new Vector3(
        Random.Range(-1.5f, 1.5f + f2),
        1f,
        Random.Range(-4.5f, 4.5f + f2)
    );
}
```

3.2.- Botón de mover á zona Neutral

Os NetPlayers xa se espaeñan de inicio na zona neutra, trátase de que existan un botón para volver dende calquier equipo á zona neutra.

Dado que os eixos Z e Y son correctos do espaneo inicial, os NetPlayers que desexen abandonar un equipo, só deben mudar á súa posición en X

```

public void MoveNeutral()
{
    Vector3 tmpPosition = transform.position;
    tmpPosition.x = Random.Range(-1.5f, 1.5f);
    transform.position = tmpPosition;
}

[ClientRpc]
public void MoveNeutralClientRpc(ClientRpcParams rpcParams = default)
{
    MoveNeutral();
}

[ServerRpc]
public void MoveNeutralServerRpc(ServerRpcParams rpcParams = default)
{
    MoveNeutral();
}

```

Nota: Utilicei aquí un ClientRpc porque me daba que non ía chegar ao punto 5

E desde o GameManager faise a chamada distinguindo a pedida individual da de Servidor

GameManager.cs

```

static void SubmitNeutralZone()
{
    if (GUILayout.Button(NetworkManager.Singleton.IsServer
        ? "Todos a Inicio"
        : "Mover a inicio"))
    {
        if (NetworkManager.Singleton.IsServer)
        {
            foreach (ulong uid in NetworkManager.Singleton.ConnectedClientsIds)
            {
                NetworkManager.Singleton.SpawnManager.GetPlayerNetworkObject(uid)
                    .GetComponent<Player>()
                    .MoveNeutralClientRpc();
            }
        }
        else
        {
            NetworkManager.Singleton.SpawnManager.GetLocalPlayerObject()
                .GetComponent<Player>()
                .MoveNeutralServerRpc();
        }
    }
}

```

3.2.- Corrección

Amén do despropósito que supón que o Host (que é servidor pero tamén player) teña a vantaxe de poder mover a tódolos participantes á zona neutra, resulta que, así implementado, o player do equipo Host ten a desvantaxe de que non ten botón para se mover individualmente á zona neutra.

1.- Cámbiase a condición

```
if (NetworkManager.Singleton.IsServer)
```

por

```
if (NetworkManager.Singleton.IsServer && !NetworkManager.Singleton.IsClient)
```

para que soamente un equipo servidor puro, que non participe no xogo, teña ese poder.

2.- Elimínase o MoveNeutralClientRpc()

Este servidor puro percorre tódalas instancias e chama ao método local MoveNeutral()

3.3.- fai que a tecla "m" teña a mesma acción.

Agrégase a condición no Update de cada player propietario

```
if (Input.GetKeyDown(KeyCode.M)) { MoveNeutralServerRpc(); }
```

De querer que a letra "m" execute a acción no servidor, habería que agregar a condición no Update() do GameManager.

Punto 4

Créase unha lista de materiais no script de Player



Créase unha NetworkVariable para os colores de equipo

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

No Update monitoreamos cando as cápsulas mudan de zona e pídeselle ao server que asigne valor á NetworkVariable segundo a posición das cápsulas no taboleiro.

```
[ServerRpc]
public void SetTeamColorServerRpc(int zone = 0,
    ServerRpcParams clientRpcParams = default)
{
    PlayerColor.Value = zone;
}
```

O meu fallo foi actualizar a cor das cápsulas dentro da condición (IsOwner).

Corrección:

```

void Update()
{
    if (IsOwner)
    {
        ...

        if (transform.position.x < -1.5f)
        {
            SetTeamColorServerRpc(1);
        }

        else if (transform.position.x > 1.5f)
        {
            SetTeamColorServerRpc(2);
        }

        else
        {
            SetTeamColorServerRpc(0);
        }
    }
    mr.material = playerColors[PlayerColor.Value];
}

```

Punto 5

Solución 1 - Usando o Delegate

En caso de que un equipo esté cheo, só se poderán mover os players dese equipo e os outros so poderán moverse de novo cando o equipo non esté cheo

Entendo que cando un dos dous equipos estea cheo, o resto queda inmovilizado, así se consigue que ningún equipo teña máis xogadores do máximo establecido. So recuperarán todos o permiso de movimiento cando algún xogador do equipo cheo abandona o seu equipo, por tanto volve cambiar de equipo.

A miñá idea é crear unha lista de contadores de xogadores en cada equipo, na que o index 0 sexa a zona neutra e o resto de índices o nº de equipo ([1] equipo 1, [2] equipo 2, etc.)

Prográmao de tal xeito que poidas mudar facilmente o número máximo de players nun equipo no futuro.

```
[SerializeField] const int MAX_TEAM_PLAYERS = 2;
```

Vale, para non crear a lista en cada instancia de Player hai que gardala no GameManager

```

public class GameManager : NetworkBehaviour
{
    public static GameManager instance;

    // Nº de equipos
    [SerializeField] const int MAX_TEAMS = 2;
    // Máximo de players por equipo
    [SerializeField] const int MAX_TEAM_PLAYERS = 2;

    // Lista de xogadores por equipo
    // o index 0 é a zona neutra, os seguintes son os nº de equipo
    List<int> playersTeam = new List<int>();

    public List<int> PlayersTeam
    {
        get { return playersTeam; }
        set { playersTeam = value; }
    }

    public int MaxTeamPlayers { get { return MAX_TEAM_PLAYERS; } }
    ...
}

```

Inicialízase no Awake(), ou no OnEnable(), a lista de equipos con 0 xogadores cada un

```

void OnEnable()
{
    // Simple Singleton
    if (instance != null && instance != this)
    {
        Destroy(this);
    }
    instance = this;

    // Inicialización de equipos, MAX_TEAMS + Neutral
    // e xogadores por equipo (ningún)
    for (int i = 0; i <= MAX_TEAMS; i++)
    {
        // Inicialización de players por equipo
        playersTeam.Add(0);
    }
}

```

E dende o script de players usamos un Delegate cada vez que un xogador muda de equipo.

Primeiro restamos o xogador ao equipo anterior e sumámolo no equipo novo, logo compárase se o novo equipo chegou ao máximo de xogadores e, de ser así, chámase a un método ServerRpc para que de os permisos de movimento segundo o enunciado:

```

public void OnTeamChanged(int previous, int current)
{
    // Descontamos o player do equipo anterior
    GameManager.instance.PlayersTeam[previous]--;

    // Sumamos o player ao novo equipo
    GameManager.instance.PlayersTeam[current]++;

    Debug.Log($"{gameObject.name}.OnTeamChanged({previous}, {current})");
    Debug.Log($"\\t GameManager.instance.PlayersTeam(
        {GameManager.instance.PlayersTeam[0]},
        {GameManager.instance.PlayersTeam[1]},
        {GameManager.instance.PlayersTeam[2]})");

    // Cando un equipo non neutral chegue ao límite de players
    if (GameManager.instance.PlayersTeam[current]
        == GameManager.instance.MaxTeamPlayers
        && IsOwner
    )
    {
        SetPlayerCanMoveServerRpc(current);
    }
}

[ServerRpc]
void SetPlayerCanMoveServerRpc
    (int currentTeam, ServerRpcParams rpcParams = default)
{
    foreach (ulong uid in NetworkManager.Singleton.ConnectedClientsIds)
    {
        var player = NetworkManager.Singleton.SpawnManager
            .GetPlayerNetworkObject(uid).GetComponent<Player>();

        // Cando un xogador se move de un equipo cheo a Neutro,
        // activase o movemento a todos
        if (currentTeam == 0)
        {
            player.PlayerCanMove.Value = true;

        }

        else
        {
            if (currentTeam == player.PlayerTeam.Value)
            {
                player.PlayerCanMove.Value = true;
            }

            else
            {
                player.PlayerCanMove.Value = false;
            }
        }

        Debug.Log($"{gameObject.name}.Player.Value = {PlayerTeam.Value}
CanMove = {PlayerCanMove.Value}");
    }
}

```

Vale, a idea funciona.

Teño un fallo de lóxica que é cando o equipo neutro ten máis de 2 xogadores despois da primeira inmovilización ou cando entran novos xogadores despois da primeira inmovilización, pero eso xa o amáño no seguinte commit

Solución 2

(utiliza ClientRPC para avisalos)

Player.cs

```
[ClientRpc]
public void SetPlayerCanMoveClientRpc(bool canMove
, ClientRpcParams clientRpcParams = default)
{
    PlayerCanMove.Value = canMove;
}
```

Si entendín ben a teoría, só un Owner pode chamar a ServerRpc e só un server pode chamar a ClientRpc, así que se simplifica o código facendo o cómputo dende o GameManager

Player.cs

```
public void OnTeamChanged(int previous, int current)
{
    GameManager.instance.CheckPlayersMovePermission(previous, current);
}
```

Vale, atopei o problema de que as NetworkVariables non son accesibles polo seu nivel de protección, así que fago á inversa dende Player a GameManger, creo una propiedad pública de tipo get

Player.cs

```
public int NwPlayerTeam { get { return PlayerTeam.Value; } }
```

E dende o GameManager chequéase e mándanse os permisos de movimento ao método ClientRpc

```

public void CheckPlayersMovePermission(int previousTeam, int currentTeam)
{
    // Descontamos 1 player do equipo anterior
    playersTeam[previousTeam]--;

    // Sumamos 1 player ao novo equipo
    playersTeam[currentTeam]++;

    // Debug.Log($"{gameObject.name}.SetPlayersMovePermission
    // ({previousTeam}, {currentTeam})");
    // Debug.Log($"{playersTeam[{playersTeam[0]}], {playersTeam[1]}, {playersTeam[2]}}");

    // Se algún equipo está cheo

    if (playersTeam[1] == MAX_TEAM_PLAYERS || playersTeam[2] == MAX_TEAM_PLAYERS)
    {
        foreach (ulong uid in NetworkManager.Singleton.ConnectedClientsIds)
        {
            var player = NetworkManager.Singleton.SpawnManager
                .GetPlayerNetworkObject(uid).GetComponent<Player>();

            int playerTeam = player.NwPlayerTeam;

            // Os xogadores do equipo cheo teñen permiso de movimiento
            if (playerTeam != 0 && playersTeam[playerTeam] == MAX_TEAM_PLAYERS)
            {
                player.SetPlayerCanMoveClientRpc(true);
            }

            // O resto non
            else
            {
                player.SetPlayerCanMoveClientRpc(false);
            }

            // Debug.Log($"GameManager Player {uid} Team {player.NwPlayerTeam}");
        }
    }

    // Se ningún equipo está cheo, todos teñen permiso de movimiento
    else
    {
        foreach (ulong uid in NetworkManager.Singleton.ConnectedClientsIds)
        {
            var player = NetworkManager.Singleton.SpawnManager
                .GetPlayerNetworkObject(uid).GetComponent<Player>();

            player.SetPlayerCanMoveClientRpc(true);
        }
    }
}

```

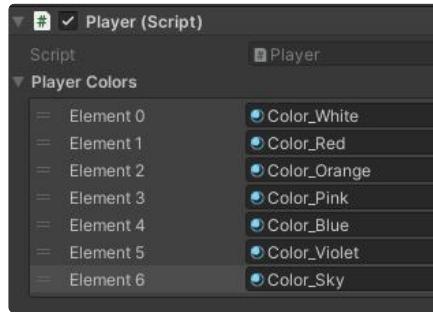
Listo!

Fago un commit porque funciona e logo escalo o código a múltiples equipos e doulle unha lectura más elegante

Punto 6

O idóneo sería, coma é o GameManager quen leva a conta de equipos, o propio sería que é tamén tivese as listas de corea permitidas a cada ún, máxime en implementación > de 2 equipos.

Pero non me vou complicar. Vou tilizar o código existente no Player e o exercicio feito en clase



Simplemente buscase un numero aleatorio que non estea asignado xa entre 1 e 3 para o equipo 1 e entre 4 e + para o equipo 2

```
int GetRandomTeamColor(int rdmMmin, int rdmMax)
{
    // Debug.Log($"{gameObject.name}.Player.SetRandomColor");

    int rdmColor;
    bool takenColor = false;

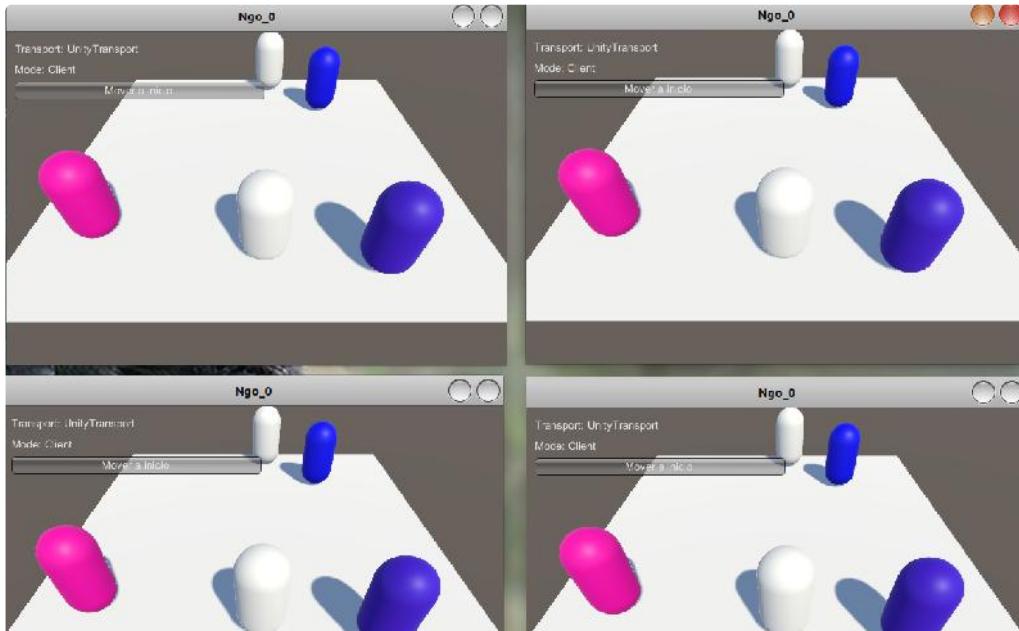
    do
    {
        rdmColor = Random.Range(rdmMmin, rdmMax + 1);
        foreach (ulong uid in NetworkManager.Singleton.ConnectedClientsIds)
        {
            var player = NetworkManager.Singleton.SpawnManager
                .GetPlayerNetworkObject(uid).GetComponent<Player>();

            takenColor = false;

            if (rdmColor == player.PlayerColor.Value)
            {
                takenColor = true;
                break;
            }
        }
    } while (takenColor);

    return rdmColor;
}
```

Poderíase millorar evitando o delegate de cambio de color asignando as cores cando se asigna o equipo.



Punto 7 - OnValueChanged

Punto 4

Utilízase OnValueChanged para o Punto 4 para monitorizar o cambio de valor da NetworkVariable PlayerColor.

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

Suscríbese ao delegate no OneNetworkSpawn()

```
public override void OnNetworkSpawn()
{
    PlayerColor.OnValueChanged += OnColorChanged;
    ...
}
```

e desinscíbese no OnNetworkDespawn()

```
public override void OnNetworkDespawn()
{
    PlayerColor.OnValueChanged -= OnColorChanged;
}
```

O método asociado muda a cor dos NetPlayers

```
public void OnColorChanged(int previous, int current)
{
    mr.material = playerColors[PlayerColor.Value];
}
```

Elimínase o control de zona no Update e aplícase a condición de PlayerColor cando os netPlayers se movan no taboleiro, reutilizando o método MoveServerRpc

```
[ServerRpc]
void MoveServerRpc(Vector3 direction, ServerRpcParams rpcParams = default)
{
    transform.position += direction;

    if (transform.position.x < -1.5f)
    {
        PlayerColor.Value = 1;
    }

    else if (transform.position.x > 1.5f)
    {
        PlayerColor.Value = 2;
    }

    else
    {
        PlayerColor.Value = 0;
    }
}
```

Por último, hai que aplicar a cor de equipo en tódalas instancias (condición ! IsOwner)

```

public override void OnNetworkSpawn()
{
    // Delegate que publica se una NetworkVariable muda de valor
    // Suscripción
    PlayerColor.OnValueChanged += OnColorChanged;

    mr = GetComponent<MeshRenderer>();

    if (IsOwner)
    {
        InitializeServerRpc();
    }
    else
    {
        mr.material = playerColors[PlayerColor.Value];
    }
}

```

Punto 5

```

PlayerTeam.OnValueChanged -= OnTeamChanged;

public void OnTeamChanged(int previous, int current)
{
    GameManager.instance.CheckPlayersMovePermission(previous, current);
}

```