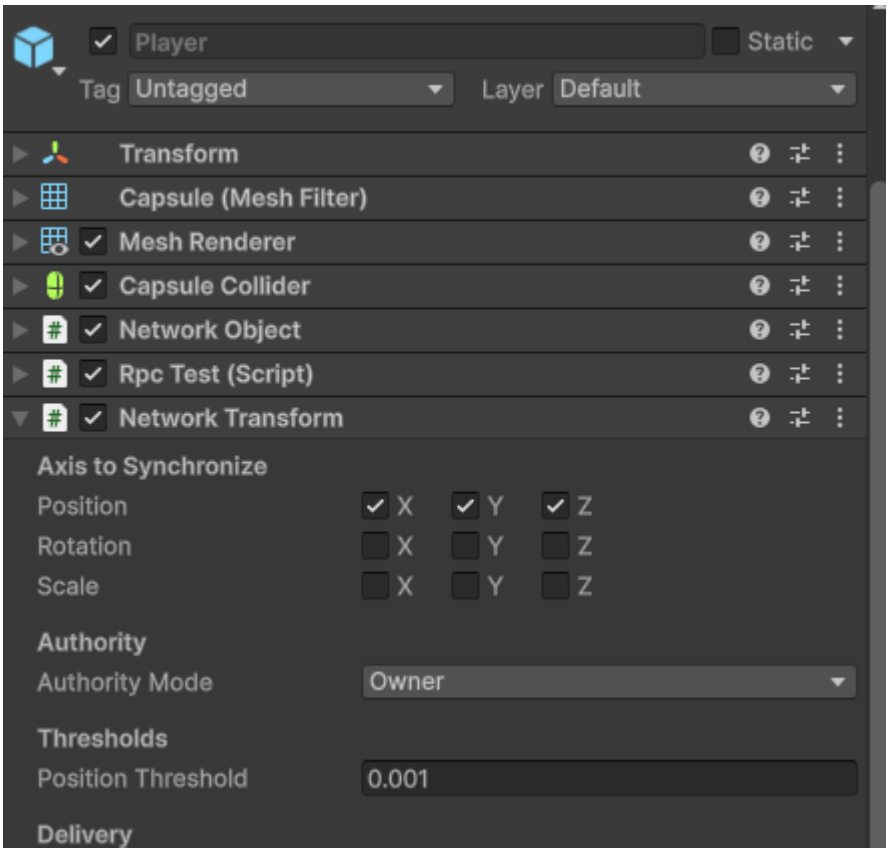
- Movimiento de los jugadores:

```
1 reference
public float moveSpeed = 5f;

0 references
private void Update()
{
    if (!IsOwner) return;

    float moveX = Input.GetAxis("Horizontal");
    float moveZ = Input.GetAxis("Vertical");

    Vector3 move = new Vector3(moveX, 0f, moveZ) * moveSpeed * Time.deltaTime;
    transform.Translate(move); // Mueve localmente

    // La posición será sincronizada automáticamente por NetworkTransform
}
```

Cada jugador en el script de "HelloWorldPlayer" mueve solo su personaje según los inputs; "if (!IsOwner) return;" sirve para que si no eres el propietario del jugador que tiene ese script no podrás hacer nada.

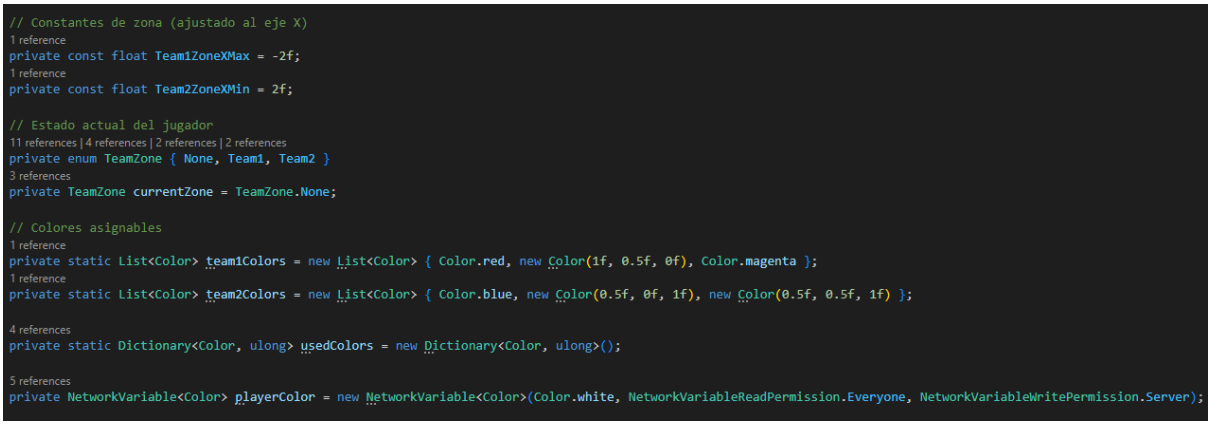



Añadimos también un "NetworkTransform" para sincronizar la posición con el servidor o host y solo pasamos la posición (aunque al no saltar

no haría falta pasar la Y) ya que es lo único que se puede alterar en este proyecto. También ponemos el Authority Mode en Owner para que sea el jugador el que se mueve primero, y no haya tanto lag -> Esto lo cambié más adelante porque al no tener autoridad el servidor no funcionaba bien el límite de equipos porque no tenía tanto control sobre la posición de los jugadores.

- **Cambio de color en los equipos:**

Primero delimité las zonas de cada equipo e hice una lista de colores para cada uno:



```
// Constantes de zona (ajustado al eje X)
1 reference
private const float Team1ZoneXMax = -2f;
1 reference
private const float Team2ZoneXMin = 2f;

// Estado actual del jugador
11 references | 4 references | 2 references | 2 references
private enum TeamZone { None, Team1, Team2 }
3 references
private TeamZone currentZone = TeamZone.None;

// Colores asignables
1 reference
private static List<Color> team1Colors = new List<Color> { Color.red, new Color(1f, 0.5f, 0f), Color.magenta };
1 reference
private static List<Color> team2Colors = new List<Color> { Color.blue, new Color(0.5f, 0f, 1f), new Color(0.5f, 0.5f, 1f) };

4 references
private static Dictionary<Color, ulong> usedColors = new Dictionary<Color, ulong>();

5 references
private NetworkVariable<Color> playerColor = new NetworkVariable<Color>(Color.white, NetworkVariableReadPermission.Everyone, NetworkVariableWritePermission.Server);
```

Después con el diccionario “usedColors” se guardan los colores que ya estén en uso desde el servidor. Ya que el cambio de colores lo lleva el servidor, es el que decide qué colores se pueden usar, para que sea un sistema justo, ya que si dos jugadores entran a la vez y escogen el mismo color podría generar un bug.

```
1 reference
private void OnColorChanged(Color previous, Color current)
{
    SetColor(current);
}

2 references
private void SetColor(Color color)
{
    if (_renderer == null)
        _renderer = GetComponent<Renderer>();

    _renderer.material.color = color;
}
```

OnColorChanged avisa a todos los clientes del estado de los colores cuando este cambia. Y SetColor cambia el color local del jugador.

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

    float moveX = Input.GetAxis("Horizontal");
    float moveZ = Input.GetAxis("Vertical");
    Vector3 move = new Vector3(moveX, 0f, moveZ) * moveSpeed * Time.deltaTime;
    transform.Translate(move);

    if (Input.GetKeyDown(KeyCode.M))
    {
        if (IsServer) MoveToStart();
        else RequestMoveToStartServerRpc();
    }

    RequestZoneCheckServerRpc(transform.position);
}
```

Esto hace que en cada frame los clientes envían su posición al servidor y el es el encargado de decidir si cambiarlos de color o no.
El servidor en “CheckZoneChange” decide si hay un cambio de zona.

```
1 reference
private void CheckZoneChange(Vector3 pos)
{
    float x = pos.x;
    TeamZone newZone = TeamZone.None;

    if (x < Team1ZoneXMax)
        newZone = TeamZone.Team1;
    else if (x > Team2ZoneXMin)
        newZone = TeamZone.Team2;

    if (newZone != currentZone)
    {
        HandleZoneChange(newZone);
    }
}
```

Y posteriormente asigna un color disponible según el equipo que corresponda por la posición en

```
1 reference
private void HandleZoneChange(TeamZone newZone)
{
    ReleaseColor();
    switch (newZone)
    {
        case TeamZone.Team1:
            SetTeamColor(team1Colors);
            break;
        case TeamZone.Team2:
            SetTeamColor(team2Colors);
            break;
        case TeamZone.None:
            playerColor.Value = Color.white;
            break;
    }

    currentZone = newZone;
}

2 references
private void SetTeamColor(List<Color> teamColors)
{
    foreach (var color in teamColors)
    {
        if (!usedColors.ContainsKey(color))
        {
            usedColors[color] = OwnerClientId;
            playerColor.Value = color;
            return;
        }
    }
}
```

También al cambiar de zona libera ese color para que se pueda volver a usar en "ReleaseColor":

```
2 references
private void ReleaseColor()
{
    List<Color> toRemove = new List<Color>();
    foreach (var kvp in usedColors)
    {
        if (kvp.Value == OwnerClientId)
        {
            toRemove.Add(kvp.Key);
        }
    }
    foreach (var color in toRemove)
    {
        usedColors.Remove(color);
    }
}
```

- Número máximo de jugadores por equipo:

Para esto usamos un “HashSet<ulong>” para cada equipo, que guarda la información de cuantos jugadores hay en cada equipo y si un jugador ya pertenece a ese quipo:

```
0 references
private static HashSet<ulong> team1Players = new HashSet<ulong>();
6 references
private static HashSet<ulong> team2Players = new HashSet<ulong>();
```

Después limitamos la entrada con “CanEnterZone” y sólo dejamos entrar a jugadores que ya pertenezcan a ese equipo (en este caso no haría falta ya que al salir de la zona dejas de ser del equipo) y a otros jugadores si aún no hemos llegado al límite:

```
1 reference
private bool CanEnterZone(TeamZone zone)
{
    switch (zone)
    {
        case TeamZone.Team1:
            return team1Players.Contains(OwnerClientId) || team1Players.Count < MaxPlayersPerTeam;
        case TeamZone.Team2:
            return team2Players.Contains(OwnerClientId) || team2Players.Count < MaxPlayersPerTeam;
        default:
            return true;
    }
}
```

El movimiento se pide desde el cliente:

```
if (move.magnitude > 0f)
{
    TryMoveServerRpc(move * moveSpeed * Time.deltaTime);
}
```

Y el servidor valida si se puede mover ahí con:

```
[Rpc(SendTo.Server)]
1 reference
private void TryMoveServerRpc(Vector3 delta)
{
    Vector3 targetPos = transform.position + delta;

    if (IsMoveAllowed(targetPos))
    {
        transform.position = targetPos;
        lastValidPosition = transform.position;
        CheckZoneChange(targetPos);
    }
    else
    {
        // Opcional: devolver al jugador a la última posición válida
        transform.position = lastValidPosition;
    }
}
```

```
1 reference
private bool IsMoveAllowed(Vector3 targetPos)
{
    float x = targetPos.x;

    if (x < Team1ZoneXMax)
    {
        return team1Players.Contains(OwnerClientId) || team1Players.Count < MaxPlayersPerTeam;
    }
    else if (x > Team2ZoneXMin)
    {
        return team2Players.Contains(OwnerClientId) || team2Players.Count < MaxPlayersPerTeam;
    }

    return true;
}
```

- Botón de Move to start o “M”:

En cada frame verificamos si se pulsó la tecla “M” , solo el dueño del jugador puede comprobarlo por el “if (!IsOwner) return;”:

```
0 references
private void Update()
{
    if (!IsOwner) return;

    float moveX = Input.GetAxis("Horizontal");
    float moveZ = Input.GetAxis("Vertical");
    Vector3 move = new Vector3(moveX, 0f, moveZ);

    if (move.magnitude > 0f)
    {
        TryMoveServerRpc(move * moveSpeed * Time.deltaTime);
    }

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

El metodo “RequestMoveToStartServerRpc” se ejecuta en el servidor cuando el cliente lo invoca y verifica si eres el dueño del jugador. Después con “MoveToStart” cambia la posición del jugador a una posición inicial aleatoria:

```
[ServerRpc]
1 reference
private void RequestMoveToStartServerRpc(ServerRpcParams rpcParams = default)
{
    if (rpcParams.Receive.SenderClientId != OwnerClientId) return;

    MoveToStart();
}

6 references
public void MoveToStart()
{
    Vector3 start = GetRandomCentralPosition();
    transform.position = start;
    lastValidPosition = start;

    playerColor.Value = Color.white;
    ReleaseColorAndTeam();
    currentZone = TeamZone.None;
}
```