Informe de Desarrollo del Proyecto Unity Netcode Multiplayer

~ Manuel González Portela ~

---

1. Uso adecuado de variables, métodos, orden, eficiencia y calidad del código

He seguido prácticas estandar que hemos ido aprendiendo en clase. El código está ordenado para que se presentan las variables al principio, las diferentes fases de ejecución de unity en orden, desde el Awake() hasta el OnGui(). A continuación se disponen los métodos.

Todo el cógido tiene un naming de variables acorde a la función que van a desempeñar, al menos dentro de unos estándares entendibles.

---

2. Creación desde cero de juego 3D tipo "Getting Started" con cápsulas que se mueven con NetworkTransform

Aquí usamos la práctica realizada en clase. Con el cambio de que el jugador se puede desplazar pulsado las teclas AWSD. Realizamos la comprobacion de if(!IsOwner) para que solo el jugador

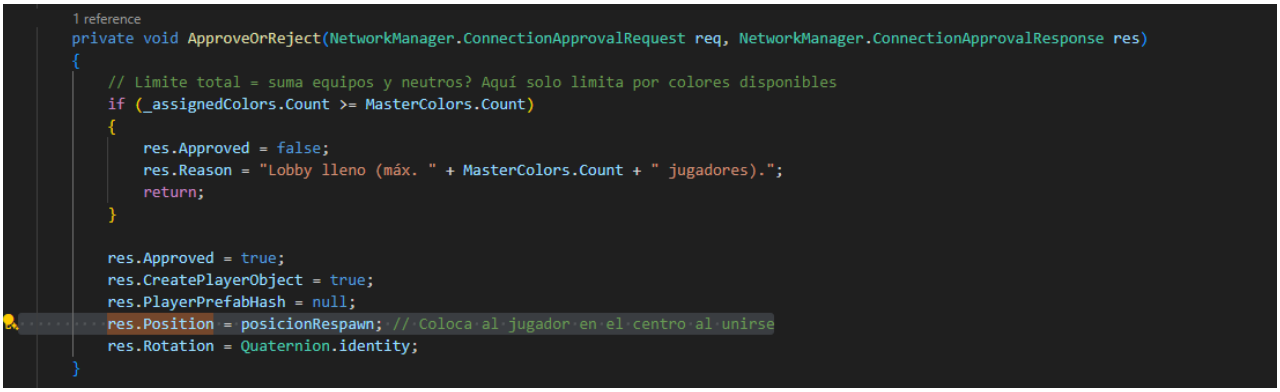
```
private void FixedUpdate()
{
    if (!IsOwner) return; // Solo el jugador local puede controlar su movimiento

    Vector3 direction = new Vector3(
        Input.GetKey(KeyCode.D) ? 1 : Input.GetKey(KeyCode.A) ? -1 : 0,
        0,
        Input.GetKey(KeyCode.W) ? 1 : Input.GetKey(KeyCode.S) ? -1 : 0
    );
```

tenga acceso a su movimiento

---

3. Spawn aleatorio en zona central, botón y tecla "m" para mover jugador al centro, con acción global si lo hace el servidor

Cuando aprobamos la conexión:



```
1 reference
private void ApproveOrReject(NetworkManager.ConnectionApprovalRequest req, NetworkManager.ConnectionApprovalResponse res)
{
    // Límite total = suma equipos y neutros? Aquí solo limita por colores disponibles
    if (_assignedColors.Count >= MasterColors.Count)
    {
        res.Approved = false;
        res.Reason = "Lobby lleno (máx. " + MasterColors.Count + " jugadores).";
        return;
    }

    res.Approved = true;
    res.CreatePlayerObject = true;
    res.PlayerPrefabHash = null;
    res.Position = posicionRespawn; // Coloca al jugador en el centro al unirse
    res.Rotation = Quaternion.identity;
}
```

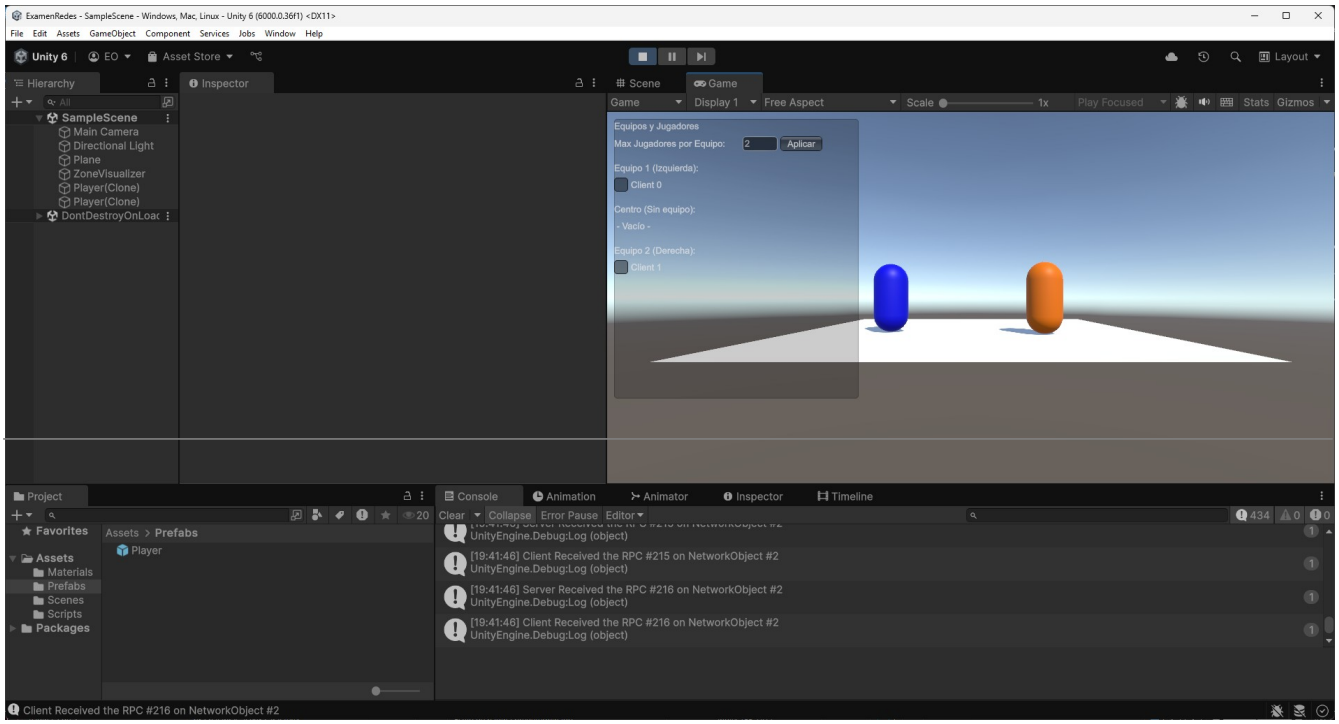
Script Manager. Línea de spawn resaltada. Colocamos al jugador en la zona central.

Para realizar el teletransporte:

```
[ServerRpc]
1 reference
private void TeleportToCenterServerRpc(ServerRpcParams rpcParams = default)
{
    RemoveFromTeam(OwnerClientId);
    Position.Value = centerPosition;
    PlayerColor.Value = neutralColor;
}

[ServerRpc(RequireOwnership = false)]

1 reference
private void TeleportAllToCenterServerRpc()
{
    foreach (var clientId in NetworkManager.Singleton.ConnectedClientsIds)
    {
        var player = NetworkManager.Singleton.SpawnManager
            .GetPlayerNetworkObject(clientId)
            .GetComponent<HelloWorldPlayer>();

        player.RemoveFromTeam(clientId);
        player.Position.Value = centerPosition;
        player.PlayerColor.Value = neutralColor;
    }
}
```

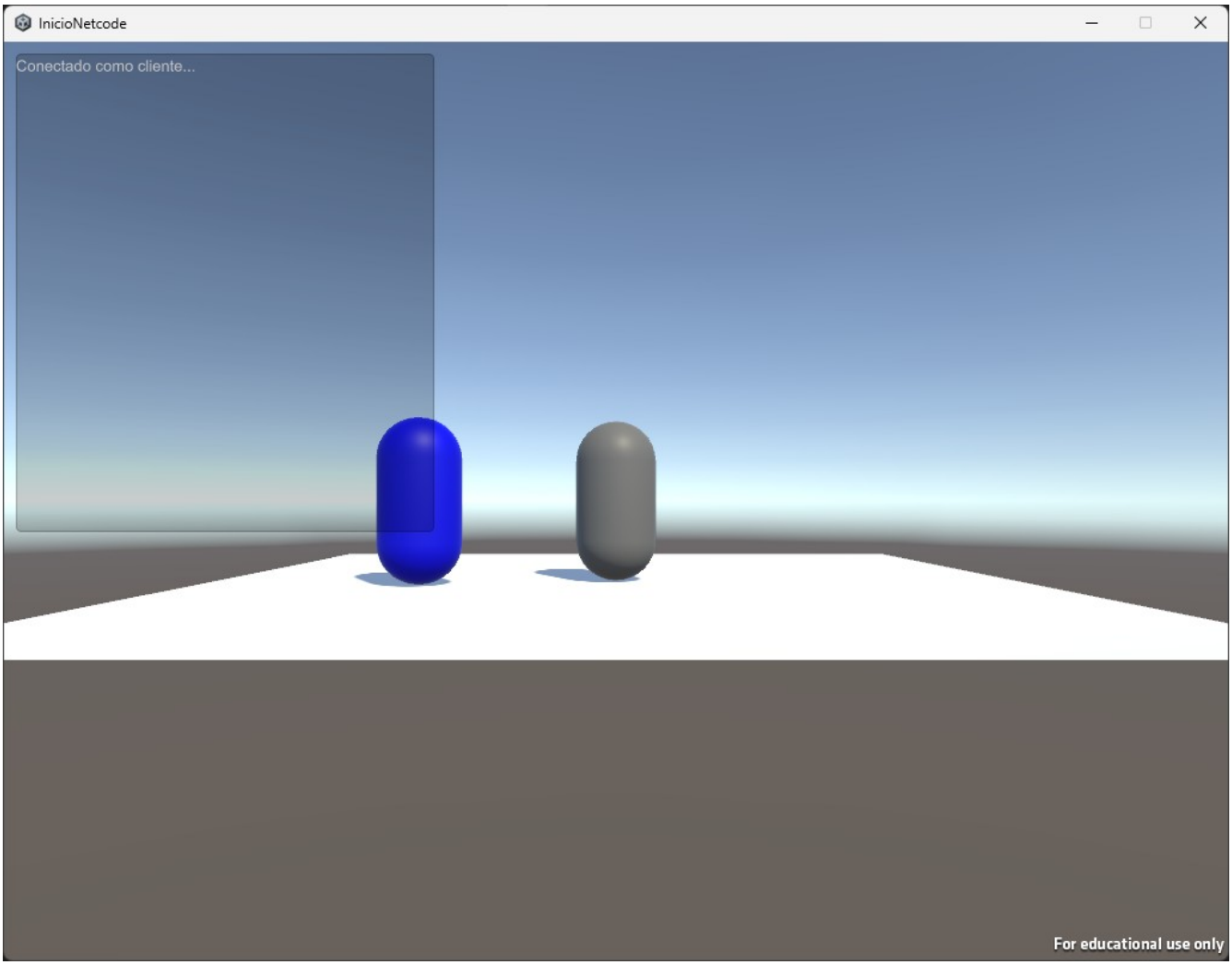
Script Player. Tenemos 2 peticiones diferenciando si se es Cliente o Servidor. En el segundo el ServerRpc puede configurarse con RequireOwnership = false para permitir que clientes envíen solicitudes incluso cuando no son dueños del objeto.

Ejemplo:


ExamenRedes - SampleScene - Windows, Mac, Linux - Unity 6 (6000.0.36f1) <DX11>
File Edit Assets GameObject Component Services Jobs Window Help
Unity 6
Asset Store
Layout
Hierarchy
Inspector
Scene
Game
SampleScene
Main Camera
Directional Light
Plane
ZoneVisualizer
Player(Clone)
Player(Clone)
DontDestroyOnLoad
Display 1
Free Aspect
Scale
1x
Play Focused
Stats
Gizmos
Equipos y Jugadores
Max Jugadores por Equipo:
2
Aplicar
Equipo 1 (Izquierda):
Client 0
Centro (Sin equipo):
- Vacío -
Equipo 2 (Derecha):
Client 1
Project
Console
Animation
Animator
Inspector
Timeline
Favorites
Assets
Materials
Prefabs
Scenes
Scripts
Packages
Assets > Prefabs
Player
Clear
Collapse
Error Pause
Editor
[19:41:46] Client Received the RPC #215 on NetworkObject #2
UnityEngine.Debug:Log (object)
[19:41:46] Server Received the RPC #216 on NetworkObject #2
UnityEngine.Debug:Log (object)
[19:41:46] Client Received the RPC #216 on NetworkObject #2
UnityEngine.Debug:Log (object)
Client Received the RPC #216 on NetworkObject #2

Aquí tenemos un usuario azul (Host) y otro derecho (Client). Al pulsar M desde el Client.

La petición desde Server no funciona correctamente.



4. Cambio de color según la zona y equipo: blanco (sin equipo), rojo (Equipo 1) y azul (Equipo 2)

- Tenemos gestionados los equipos:

```
25 references
public enum Team
{
    8 references
    None,
    5 references
    Team1,
    5 references
    Team2
}
```

Y un Diccionario en el que se añaden o eliminan para luego actualizar el GUI:

```
public Dictionary<Team, List<ulong>> teamMembers = new Dictionary<Team, List<ulong>>()
{
    { Team.Team1, new List<ulong>() },
    { Team.None, new List<ulong>() },
    { Team.Team2, new List<ulong>() }
};
```

Cada vez que el jugador se mueve, realizamos una comprobación de su posición:

```
[ServerRpc]
1 reference
private void MoveRequestServerRpc(Vector3 direction)
{
    float speed = 3f;
    Vector3 newPos = transform.position + direction * speed * Time.fixedDeltaTime;

    newPos.x = Mathf.Clamp(newPos.x, -5f, 5f);
    newPos.z = Mathf.Clamp(newPos.z, -5f, 5f);

    UpdateTeamByPosition(newPos, OwnerClientId);

    Position.Value = newPos;
}
```

Si se traspasan los umbrales, se intenta colocarlos en su respectivo equipo:

```
1 reference
private void UpdateTeamByPosition(Vector3 position, ulong clientId)
{
    if (position.x < -2)
    {
        TryJoinTeam(Team.Team1, team1Color, clientId, ref position);
    }
    else if (position.x > 2)
    {
        TryJoinTeam(Team.Team2, team2Color, clientId, ref position);
    }
    else
    {
        RemoveFromTeam(clientId);
        PlayerColor.Value = neutralColor;
        playerTeam.Value = Team.None;
    }
}
```

```
private void TryJoinTeam(Team targetTeam, Color teamColor, ulong clientId, ref Vector3 position)
{
    if (playerTeam.Value == targetTeam)
        return;

    var manager = HelloWorldManager.Instance;

    // Pide al manager que intente meter al jugador
    manager.AddPlayerToTeam(clientId, targetTeam);

    // El manager ya ajustará el color y actualizará playerTeam
    playerTeam.Value = targetTeam;
}
```

Y se envía esa informacion al Manager, que comprueba si tiene un equipo ya asignado y en ese caso lo saca (aunque no debería de ser un caso real). Tambien si el equipo está lleno o no. Si todo está correcto, se le pone el color del equipo correspondiente:

```
3 references
public void AddPlayerToTeam(ulong clientId, Team team)
{
    RemovePlayerFromAllTeams(clientId);

    if (!teamMembers.ContainsKey(team))
        teamMembers[team] = new List<ulong>();

    if (team == Team.Team1 || team == Team.Team2)
    {
        if (teamMembers[team].Count >= maxPlayersPerTeam)
        {
            Debug.Log($"Equipo {team} lleno, no se añade el jugador {clientId}");
            AddPlayerToTeam(clientId, Team.None); // Si está lleno, vuelve al centro
            return;
        }
    }

    teamMembers[team].Add(clientId);

    // Cambia color en el jugador según equipo
    var player = m_NetworkManager.SpawnManager
        .GetPlayerNetworkObject(clientId)
        .GetComponent<HelloWorldPlayer>();

    switch (team)
    {
        case Team.Team1:
            player.PlayerColor.Value = Color.blue;
            break;
        case Team.Team2:
            player.PlayerColor.Value = new Color(1f, 0.5f, 0f); // naranja
            break;
        case Team.None:
        default:
            player.PlayerColor.Value = Color.gray;
            break;
    }
}
```

5. Restricción de máximo 2 jugadores por equipo.

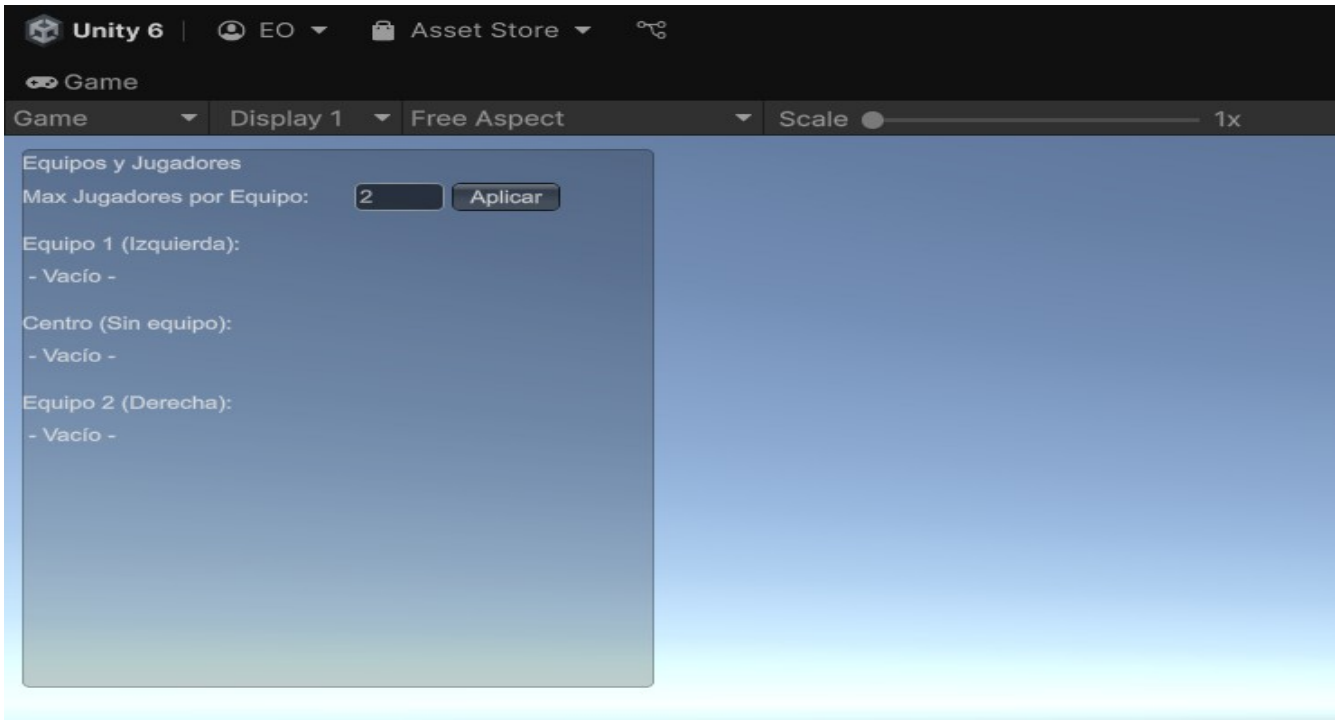
Usamos una variable en el manager que por defecto inicializamos a 2:

```
4 references
public int maxPlayersPerTeam = 2;
```

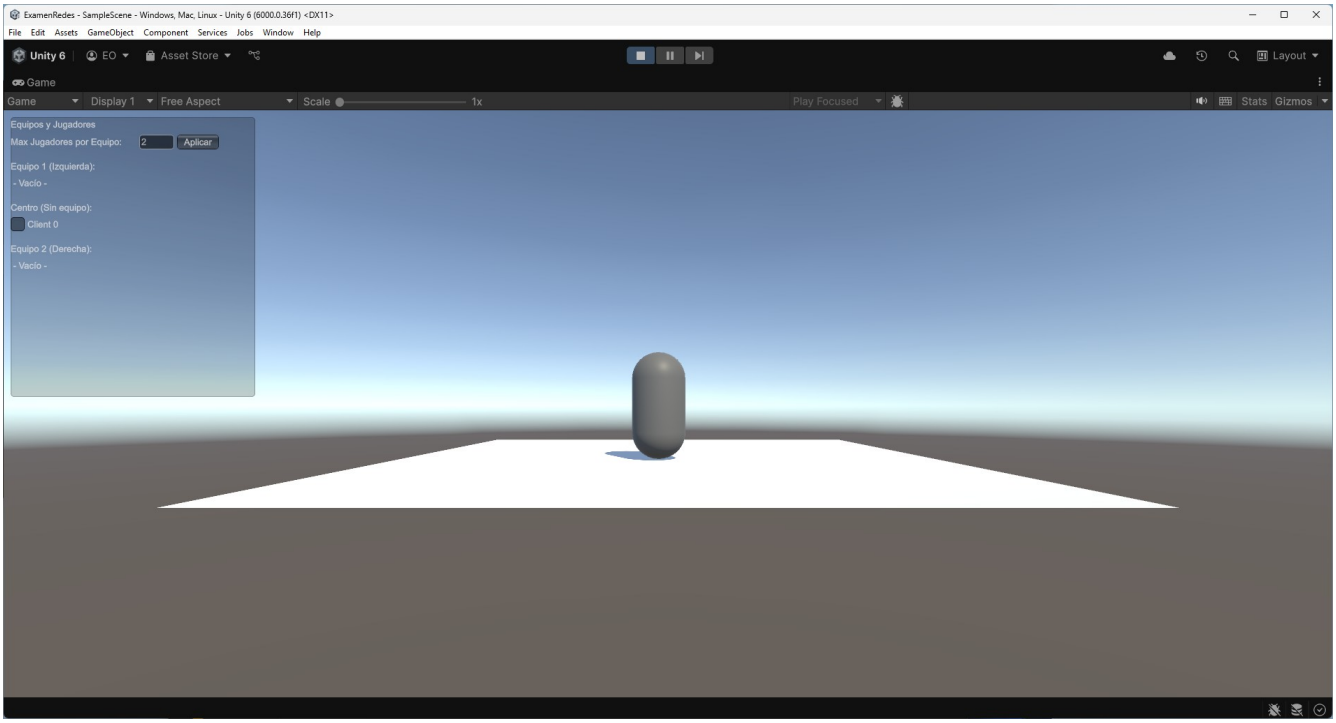
Y al intentar añadir el jugador al equipo, se hace esta comprobación:

```
// Añade jugador a equipo (y elimina de otros equipos)
3 references
public void AddPlayerToTeam(ulong clientId, Team team)
{
    RemovePlayerFromAllTeams(clientId);

    if (!teamMembers.ContainsKey(team))
        teamMembers[team] = new List<ulong>();

    if (team == Team.Team1 || team == Team.Team2)
    {
        if (teamMembers[team].Count >= maxPlayersPerTeam)
        {
            Debug.Log($"Equipo {team} lleno, no se añade el jugador {clientId}");
            AddPlayerToTeam(clientId, Team.None); // Si está lleno, vuelve al centro
            return;
        }
    }
```

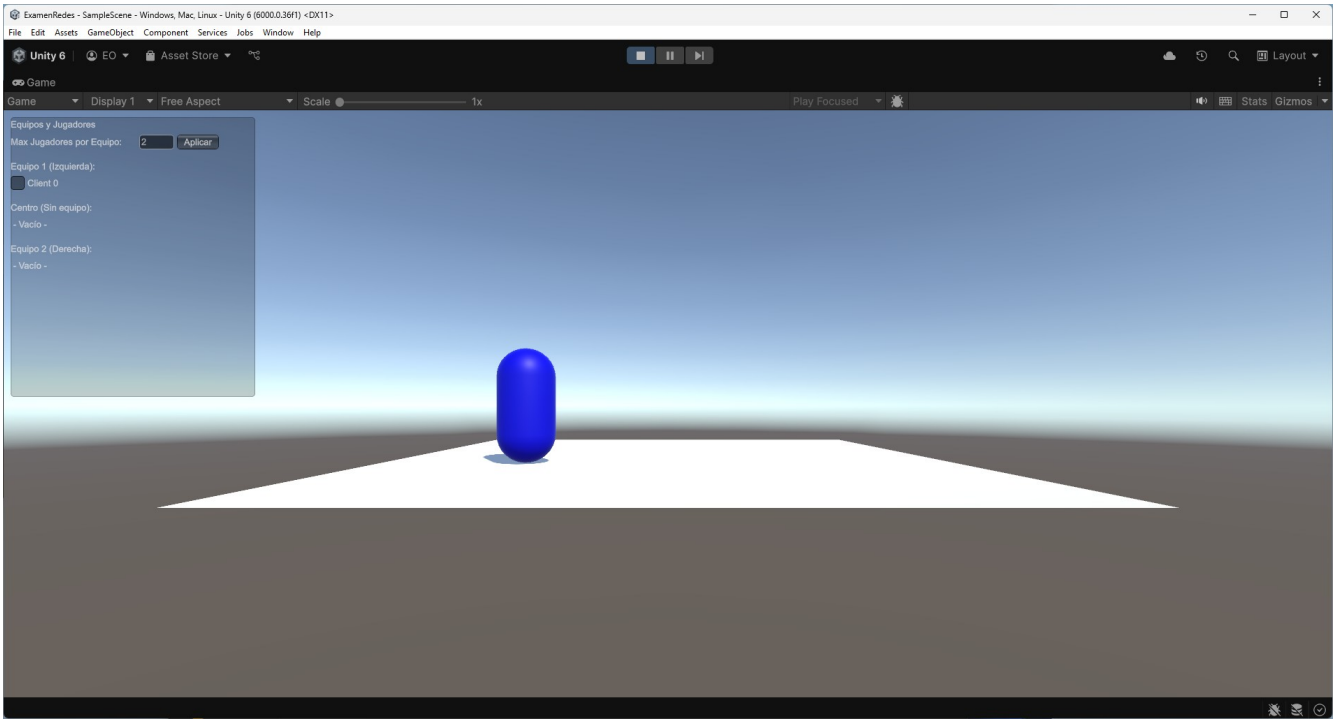
Si lo queremos cambiar en cualquier momento, al iniciar la demo lo tenemos en la interfaz gráfica:



El resto de la GUI va actualizando los equipos con los jugadores que estan en ese momento añadidos. Ejemplos:



Ejemplo1: Jugador sin equipo



Ejemplo 2: Jugador en equipo 1 (Azul)

6. Colores de equipo específicos y asignación aleatoria sin repetición

Esta parte del ejercicio no se pudo realizar.

7 . Interfaz en servidor para seleccionar el número máximo de jugadores por equipo

Explicada en el apartado 5.