🎓 LESSON 04
Team Formation

Start

Video Link

LESSON GOALS

🎯 Goal Summary:

- Implement team assignment logic
- Store players in arrays: Red, Blue
- Display team info in HUD Debug
- Create identification balls (5 Red, 5 Blue)
- Assign a colored ball above each player
- Teleport the players to their initial match positions
- Ensure Start Game only works if teams are balanced

Next

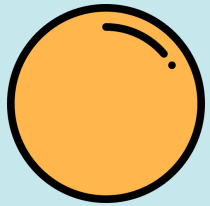
Learning Objectives + Deliverables

Learning Objectives

Create team arrays and assign arriving players

Determine team and index of each player

Debug team assignments using GameHUD



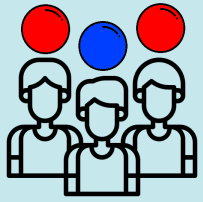
Create identification ball prefabs for each team

Enable the Start Game button only when teams are balanced

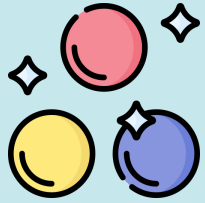
Deliverables

✓ Balanced Red & Blue teams

✓ Team assignments visible in the HUD



✓ Identification balls floating above players



✓ Correct ball ownership & despawn logic

✓ Start button becomes available only when teams are even

GENERAL DESCRIPTION OF THE LESSON

High-Level Description

🎯 **Goal:** Assign the incoming players to the 2 different teams (RED, BLUE) and use a colored ball over their avatar's head to identify them.

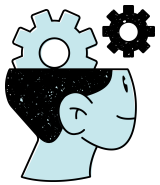
🏐 LEVEL 1: Advanced Challenge

In this lesson we will:

- Two arrays (**teamRed[], teamBlue[]**) that store player IDs
- Logic in **GameController** to assign players alternately by order of arrival
- Methods to get a player's team and team index
- Use GameHUD Debug output to show information about the team assignation
- Create ten "identification ball" prefabs (5 red, 5 blue)
- A **TeamAssignation** script attach to the previous "identification ball" that follows the owner player.
- Identification ball assignment is done during ORGANIZATION state
- Create a reset function to clear the ball assignment.
- Button logic that performs the change to GAME state only if both teams have the same number of players.

🧠AI Lesson Prompt: Team Formation

Next



Exercise 1: Create 2 Udon networked int arrays in **GameController** to keep the playerId of the players assigned to each team. The assignation is being done by order of arrival, so the players that are odd enter to team Red and the players who are even enter to team Blue.

- Actions (Part 1/2):

1) Define a constant (**MaxPlayersPerTeam = 5**). Define 2 network integer arrays for each team (Red, Blue) where we will keep the player's id, then allocate memory and initialize them to -1 at the (**Start()**) method.

2) With the previously done player counter done in the last lesson, create a function (**bool AssignPlayerToTeam(VRCPlayerApi player)**) which decides where to store the playerId of each player who has joined the game. (Blue → Red → Blue → Red →...). It should return true(Red), false(Blue).

3) Find out the right place to call the method (**AssignPlayerToTeam**)

4) Define this enum type named Team:

5) Create a function (**Team GetPlayerTeam(int targetId)**) to retrieve the team which a int playerId belongs to.

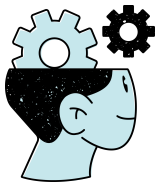
```
public class GameController : UdonSharpBehaviour
{
    public const int MaxPlayersPerTeam = 5;

    [UdonSynced]
    private int[] _teamBluePlayers = null;
    [UdonSynced]
    private int[] _teamRedPlayers = null;

    void Start()
    {
        if (_teamBluePlayers == null) _teamBluePlayers = new int[MaxPlayersPerTeam];
        if (_teamRedPlayers == null) _teamRedPlayers = new int[MaxPlayersPerTeam];
        ResetPlayerIndexes();
    }

    private void ResetPlayerIndexes()
    {
        for (int i = 0; i < MaxPlayersPerTeam; i++)
        {
            _teamBluePlayers[i] = -1;
            _teamRedPlayers[i] = -1;
        }
    }
```

```
public Team GetPlayerTeam(int targetId)
{
   if ((_teamBluePlayers == null) || (_teamRedPlayers == null)) return Team.TEAM_NONE;
   for (int i = 0; i < MaxPlayersPerTeam; i++)
   {
      if (_teamBluePlayers[i] == targetId)
      {
         return Team.TEAM_BLUE;
      }
   }
   return Team.TEAM_RED;
}
```

```
public enum Team
{
   TEAM_NONE = 0,
   TEAM_BLUE = 1,
   TEAM_RED = 2
}
```

```
private bool AssignPlayerToTeam(VRCPlayerApi player)
{
    int playerId = player.playerId;
    bool assignToRed = (_totalPlayersInGame % 2 == 0);
    int[] team = assignToRed ? _teamRedPlayers : _teamBluePlayers;
    for (int i = 0; i < MaxPlayersPerTeam; i++)
    {
       if (team[i] == -1)
       {
          team[i] = playerId;
          return assignToRed;
       }
    }
    return assignToRed;
}}
```

```
public override void OnPlayerJoined(VRCPlayerApi player)
{
   Debug.Log("On Player Joined");
   if (Networking.LocalPlayer == player)
   {
       player.SetWalkSpeed(6f);
       player.SetRunSpeed(16f);
   }
   _currentState = GameState.ORGANIZATION;
   StateChanged();
   if (Networking.IsMaster)
   {
      AssignPlayerToTeam(player);
      _totalPlayersInGame++;
   }
}
```



Exercise 1: Create 2 Udon networked int arrays in **GameController** to keep the playerId of the players assigned to each team. The assignation is being done by order of arrival, so the players that are odd enter to team Red and the players who are even enter to team Blue.

- Actions (Part 2/2):

6) Create a function (**int GetPlayerIndex(int targetId)**) that will get the position in the team's array of an int playerId.

7) Create a method (**void ShowDebugInfo()**) that will show through the gameHUD.SetDebug() the next information:

- Player Team
- Player Index
- The 2 arrays of the team assignation

8) Show that information every time a new player connects.

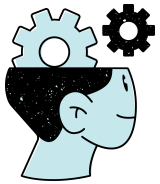
9) Test multiple VRChat instance to verify the debug information is displayed

`Code Checkpoint: Team Assignation`

```
private void ShowDebugInfo()
{
  string teamA = "";
  string teamB = "";
  for (int i = 0; i < MaxPlayersPerTeam; i++)
  {
    teamA += _teamBluePlayers[i] + ",";
    teamB += _teamRedPlayers[i] + ",";
  }
  int indexPlayer = GetPlayerIndex(Networking.LocalPlayer.playerId);
  Team teamPlayer = GetPlayerTeam(Networking.LocalPlayer.playerId);
  string teamName = "BLUE";
  if (teamPlayer == Team.TEAM_RED)
  {
    teamName = "RED";
  }
  gameHUD.SetDebug("P["+ indexPlayer +"]("+ teamName +")::A-Blue("+ teamA +")::A-Red("+ teamB +")");
}
```

```
public int GetPlayerIndex(int targetId)
{
    if ((_teamBluePlayers == null) || (_teamRedPlayers == null)) return -1;
    for (int i = 0; i < MaxPlayersPerTeam; i++)
    {
        if (_teamBluePlayers[i] == targetId)
        {
            return i;
        }
    }
    for (int i = 0; i < MaxPlayersPerTeam; i++)
    {
       if (_teamRedPlayers[i] == targetId)
       {
          return i;
       }
    }
    return -1;
}
```

```
public override void OnDeserialization()
{
    // OTHER CODE ...

    ShowDebugInfo();
}

private void StateChanged()
{
    ShowDebugInfo();

    // OTHER CODE ...
}

public override void OnPlayerJoined( ... )
{
  // OTHER CODE ...

  AssignPlayerToTeam(player);
  RequestSerialization();

  // OTHER CODE ...
}
```



Exercise 2: In order to identify to which team the players belong to we are going to place a colored ball over their heads (Blue and Red). We will handle the ball with the script **TeamAssignation**.

- Actions (Part 1/3):

1) Create a prefab for the ball Blue & Red. Then create a container in the scene with 5 balls for each color. Place the balls far away the game field so they aren't visible.

2) Create the Udon script (**TeamAssignation**) in the folder **/Game/Scrips/View** with the following variable members:

3) Next you need to create the next methods:

public bool IsTeamRed { get; }: // A getter for the private value

public void AssignOwner(int playerId) // Assignation of ball to a playerId, propagate

public void ClearOwner() // Clearing the ownership, propagate

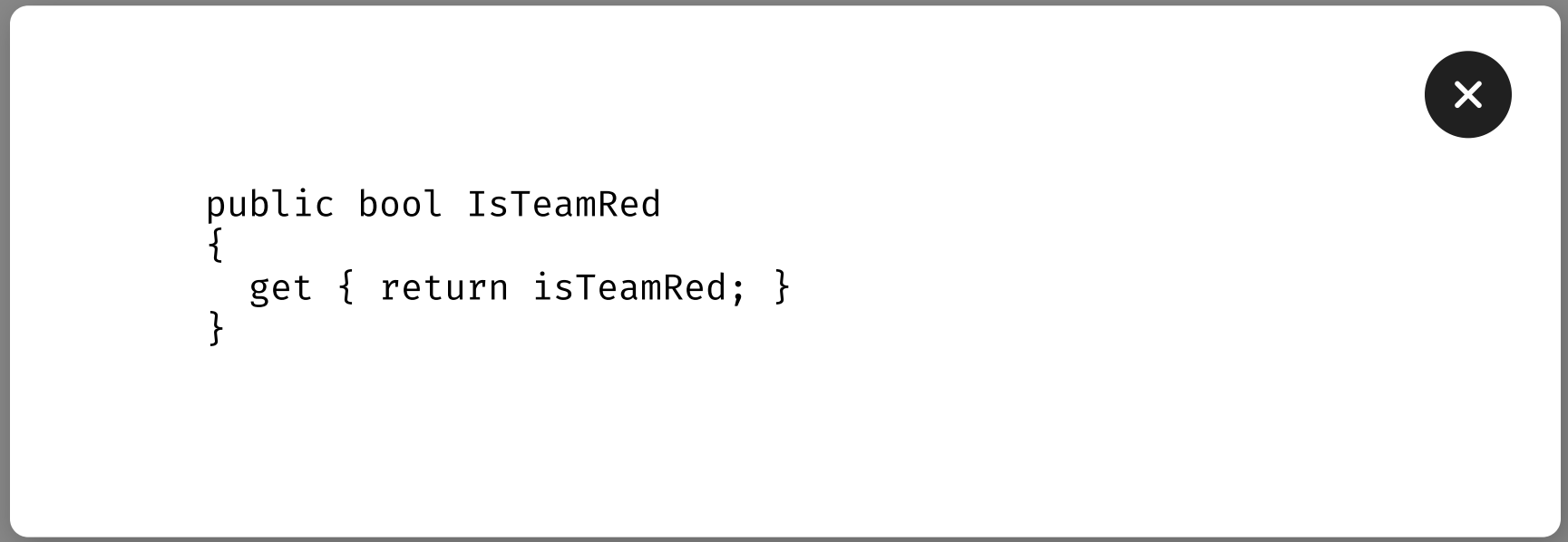
OnDeserialization() // Process RequestSerialitzation() call: Initialize _targetPlayer

void Update() // If there is a player assigned to the ball update the position, if not place it far away

```
public class TeamAssignation : UdonSharpBehaviour {

    private const float scale = 0.5f;
    private const float offset = 2.5f; // Offset y over player's head

    [SerializeField] private bool isTeamRed = false; // Team Red or Blue

    [UdonSynced] public int ownerPlayerId; // playerId of the owner

    private VRCPlayerApi _targetPlayer; // player owner reference
```

```
void Update()
{
    if (ownerPlayerId ⩽ 0)
    {
        transform.position = new Vector3(1000, 1000, 1000);
        return;
    }
    if (_targetPlayer == null)
        _targetPlayer = VRCPlayerApi.GetPlayerById(ownerPlayerId);
    if (_targetPlayer == null) return;

    Vector3 playerPos = _targetPlayer.GetPosition();
    Vector3 pos = playerPos + new Vector3(0, offset, 0);
    transform.position = pos;
}
```

```
public void AssignOwner(int playerId)
{
    ownerPlayerId = playerId;
    _targetPlayer = null;
    RequestSerialization();
}
```



```
public bool IsTeamRed
{
  get { return isTeamRed; }
}
```

```
public override void OnDeserialization()
{
    if (ownerPlayerId > 0)
        _targetPlayer = VRCPlayerApi.GetPlayerById(ownerPlayerId);
    else
        _targetPlayer = null;
}
```

```
public void ClearOwner()
{
    ownerPlayerId = 0;
    _targetPlayer = null;
    transform.position = new Vector3(1000, 1000, 1000);
    RequestSerialization();
}
```

Exercise 2: In order to identify to which team the players belong to we are going to place a colored ball over their heads (Blue and Red). We will handle the ball with the script **TeamAssignation**.

- Actions (Part 2/3):

4) Add the script to the 2 prefabs of the previously created balls and initialize the right value (bool isTeamBlue) to each prefab

5) In (**GameController**) script, create the variable member and initialize it with the team identification balls of the scene:

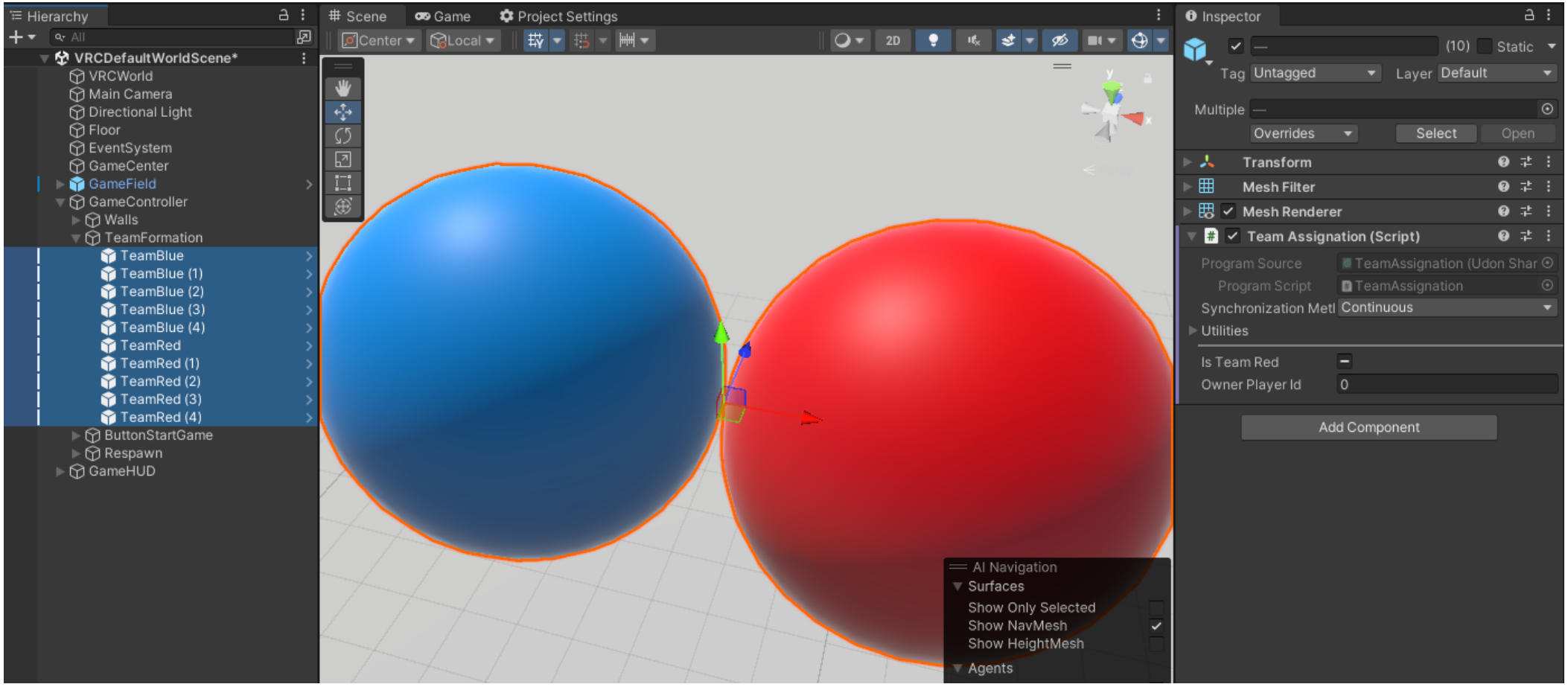
6) Still in (**GameController**) create the methods:

SpawnIdentificationBall(VRCPlayerApi player, bool isTeamRed): // This method will performed by the master and it will assign a ball of the array to the joined player. Find out where to call it.

DespawnAllIdentificationBalls(): // Method that will despawn all the balls

DespawnIdentificationBall(VRCPlayerApi player): // Method that will despawn the ball for a particular player.

OnPlayerLeft(VRCPlayerApi player): // Despawn the ball for the player who left.


Hierarchy
All
VRCDefaultWorldScene*
VRCWorld
Main Camera
Directional Light
Floor
EventSystem
GameCenter
GameField
GameController
Walls
TeamFormation
TeamBlue
TeamBlue (1)
TeamBlue (2)
TeamBlue (3)
TeamBlue (4)
TeamRed
TeamRed (1)
TeamRed (2)
TeamRed (3)
TeamRed (4)
ButtonStartGame
Respawn
GameHUD
Scene
Game
Project Settings
Center
Local
2D
AI Navigation
Surfaces
Show Only Selected
Show NavMesh
Show HeightMesh
Agents
Inspector
(10)
Static
Tag
Untagged
Layer
Default
Multiple
Overrides
Select
Open
Transform
Mesh Filter
Mesh Renderer
Team Assignation (Script)
Program Source
TeamAssignation (Udon Shar
Program Script
TeamAssignation
Synchronization Metl
Continuous
Utilities
Is Team Red
Owner Player Id
0
Add Component

```
public class GameController: UdonSharpBehaviour {

    // OTHER CODE ...

    [SerializeField]
    private GameObject[] teamAssignation; // All the references to the balls
```

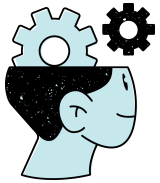
```
public override void OnPlayerLeft(VRCPlayerApi player)
{
   DespawnIdentificationBall(player);
}
```

```
private void DespawnAllIdentificationBalls()
{
  foreach (GameObject identification in teamAssignation)
  {
     TeamAssignation follower = identification.GetComponent<TeamAssignation>();
     follower.ClearOwner();
  }
}
```

```
public void AssignOwner(int playerId)
{
    ownerPlayerId = playerId;
    _targetPlayer = null;
    RequestSerialization();
}
```

```
private void DespawnIdentificationBall(VRCPlayerApi player)
{
   if (player ≠ null)
   {
      foreach (GameObject identification in teamAssignation)
      {
        TeamAssignation follower = identification.GetComponent<TeamAssignation>();
        if (follower.ownerPlayerId == player.playerId)
        {
           follower.ClearOwner();
           break;
        }
      }
    }
}
```

```
private void SpawnIdentificationBall(VRCPlayerApi player, bool isTeamRed)
{
    foreach (GameObject identification in teamAssignation)
    {
       TeamAssignation follower = identification.GetComponent<TeamAssignation>();
       if ((follower.ownerPlayerId == 0) && (follower.IsTeamRed == isTeamRed))
       {
          follower.AssignOwner(player.playerId);
          break;
       }
    }
}

public override void OnPlayerJoined(VRCPlayerApi player)
{
   // OTHER CODE ...
   if (Networking.IsMaster)
   {
      bool isTeamRed = AssignPlayerToTeam(player);
      SpawnIdentificationBall(player, isTeamRed);
      _totalPlayersInGame++;
      if (_totalPlayersInGame >= 4)
      {
         SetState(GameState.GAME);
      }
    }
}
```



Exercise 2: In order to identify to which team the players belong to we are going to place a colored ball over their heads (Blue and Red). We will handle the ball with the script **TeamAssignation**.

- Actions (Part 3/3):

7) Test in the Unity Editor and verify that over your head is the right colored ball

8) Test with multiple instances if the player team assignation is properly done and each player has the right colored ball.

`Code Checkpoint: Colored Team Balls`

```
public override void OnPlayerLeft(VRCPlayerApi player)
{
   DespawnIdentificationBall(player);
}
```

```
private void DespawnIdentificationBall(VRCPlayerApi player)
{
   if (player ≠ null)
   {
      foreach (GameObject identification in teamAssignation)
      {
        TeamAssignation follower = identification.GetComponent<TeamAssignation>();
        if (follower.ownerPlayerId == player.playerId)
        {
           follower.ClearOwner();
           break;
        }
      }
    }
}
```

```
public void AssignOwner(int playerId)
{
    ownerPlayerId = playerId;
    _targetPlayer = null;
    RequestSerialization();
}
```

```
private void SpawnIdentificationBall(VRCPlayerApi player, bool isTeamRed)
{
    foreach (GameObject identification in teamAssignation)
    {
       TeamAssignation follower = identification.GetComponent<TeamAssignation>();
       if ((follower.ownerPlayerId == 0) && (follower.IsTeamRed == isTeamRed))
       {
          follower.AssignOwner(player.playerId);
          break;
       }
    }
}

public override void OnPlayerJoined(VRCPlayerApi player)
{
   // OTHER CODE ...
   if (Networking.IsMaster)
   {
      bool isTeamRed = AssignPlayerToTeam(player);
      SpawnIdentificationBall(player, isTeamRed);
      _totalPlayersInGame++;
      if (_totalPlayersInGame >= 4)
      {
         SetState(GameState.GAME);
      }
    }
}
```

```
private void DespawnAllIdentificationBalls()
{
  foreach (GameObject identification in teamAssignation)
  {
     TeamAssignation follower = identification.GetComponent<TeamAssignation>();
     follower.ClearOwner();
  }
}
```

Exercise 3: Implement a logic to start the game when there are the same number of players for each team. **Create an interactable button** with a textfield that displays a text depending if the game is ready or not.

- Actions (Part 1/3):

1) In the scene create a GameObject named ButtonStartGame that will contain a canvas with a text inside.

2) Create the Udon script (**ButtonStartGame**) that will contain the logic of the button. That will have the next variable members:

3) Add the script to the created GameObject in the scene and initialize its members

4) In (**ButtonStartGame**) implement the following methods:

void EnableReady() // Enable the button to be able to start the game

void DisableReady() // Disables the button forbidding the game start

public void NetworkEnable(bool isReady) // Method that will call SendCustomNetworkEvent to enable or disable the button for all the connected players.

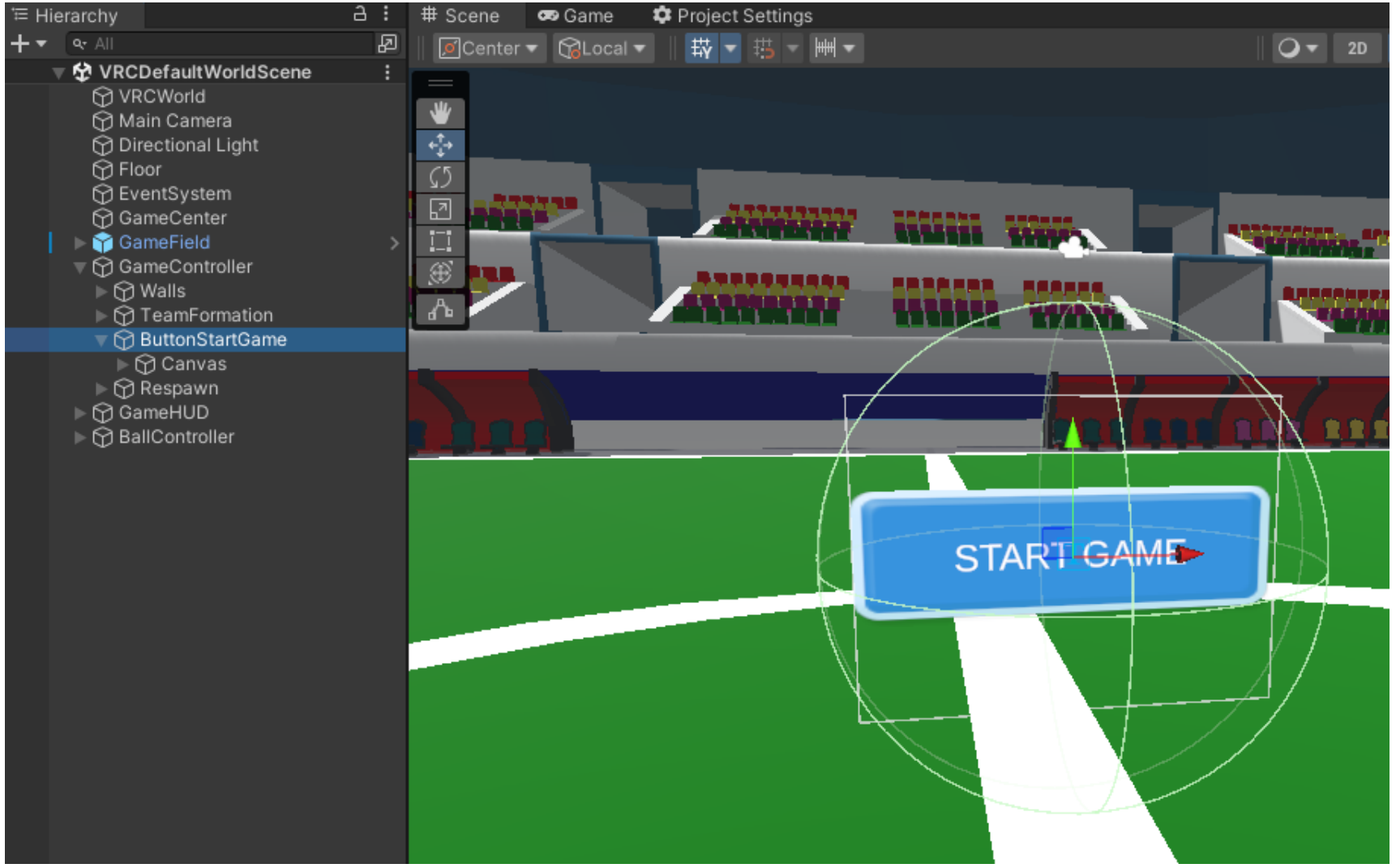
public override void Interact() // Inherited method that enables the interaction. Override it and just display a Debug.Log("PRESSED START") when enabled.

```
public void DisableReady()
{
    _isEnabled = false;
    textMessage.text = "NEED 1 MORE PLAYER...";
 }
```

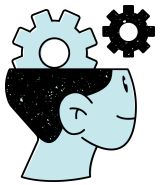
```
public void NetworkEnable(bool isReady)
{
    if (isReady)
    {
     SendCustomNetworkEvent(VRC.Udon.Common.Interfaces.NetworkEventTarget.All, nameof(EnableReady));
    }
    else
    {
     SendCustomNetworkEvent(VRC.Udon.Common.Interfaces.NetworkEventTarget.All, nameof(DisableReady));
    }
}
```

```
public override void Interact()
{
    if (gameController ≠ null)
    {
        if (_isEnabled)
        {
           Debug.Log("PRESSED START");
        }
     }
}
```

```
public class ButtonStartGame : UdonSharpBehaviour
{
    [SerializeField]
    private GameController gameController; // Reference to the GameController

    [SerializeField]
    private TextMeshProUGUI textMessage; // Reference to the textfield

    private bool _isEnabled = false; // Enable/Disable interaction
```


Hierarchy
All
VRCDefaultWorldScene
VRCWorld
Main Camera
Directional Light
Floor
EventSystem
GameCenter
GameField
GameController
Walls
TeamFormation
ButtonStartGame
Canvas
Respawn
GameHUD
BallController
Scene
Game
Project Settings
Center
Local
2D
START GAME

```
public void EnableReady()
{
    _isEnabled = true;
    textMessage.text = "START GAME";
}
```



Exercise 3: Implement a logic to start the game when there are the same number of players for each team. **Create an interactable button** with a textfield that displays a text depending if the game is ready or not.

- Actions (Part 2/3):

5) In the script (**GameController**), define a new member variable and initialize it:

6) Still in (**GameController**) implement the method (**public void StartGameRequest()**). This method will be called by the (**ButtonStartGame**) and it will perform a (**SendCustomNetworkEvent**) that will ask the master client to start the game.

7) Back in (**ButtonStartGame**) now call the (**StartGameRequest**) when running **(void Interact())** method.

8) Back to (**GameController**) we are going to remove the code that considered the total number of players to change to the state GAME and now we are going to use the new system. Remember, the button should show that is ready to start the game only when there are the same number of players for each team.

9) Test it with multiple VRChat instances.

10) In order to be able to run the game in the same Unity editor we are going to type this code in (**ButtonStartGame**)

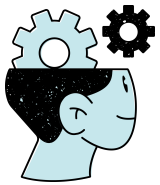
```
public void NetworkEnable(bool isReady)
{
#if UNITY_EDITOR
   SendCustomNetworkEvent(VRC.Udon.Common.Interfaces.NetworkEventTarget.All, nameof(EnableReady));
#else
   if (isReady)
   {
     SendCustomNetworkEvent(VRC.Udon.Common.Interfaces.NetworkEventTarget.All, nameof(EnableReady));
   }
   else
   {
     SendCustomNetworkEvent(VRC.Udon.Common.Interfaces.NetworkEventTarget.All, nameof(DisableReady));
   }
#endif
}
```

```
public class GameController : UdonSharpBehaviour
{
  // OTHER CODE ...

  public void StartGameRequest()
  {
    SendCustomNetworkEvent(VRC.Udon.Common.Interfaces.NetworkEventTarget.All, nameof(RequestMasterStartGame));
  }

  public void RequestMasterStartGame()
  {
     if (Networking.IsMaster)
     {
        SetState(GameState.GAME);
     }
  }
```

```
public override void OnPlayerJoined(VRCPlayerApi player)
{
    // OTHER CODE ...

    if (Networking.IsMaster)
    {
        bool isTeamRed = AssignPlayerToTeam(player);
        SpawnIdentificationBall(player, isTeamRed);
        _totalPlayersInGame++;
        buttonStartGame.NetworkEnable(_totalPlayersInGame % 2 == 0);
    }
}
```

```
public override void Interact()
{
     if (gameController ≠ null)
     {
          if (_isEnabled)
          {
               gameController.StartGameRequest();
          }
     }
 }
```

```
public class GameController : UdonSharpBehaviour
{
    // OTHER CODE ...

    [SerializeField] private ButtonStartGame buttonStartGame;
```



Exercise 3: Implement a logic to start the game when there are the same number of players for each team. **Create an interactable button** with a textfield that displays a text depending if the game is ready or not.

- Actions (Part 3/3):

10) In the script (**GameController**), define a new member variable and initialize it:

11) Back in the script (**ButtonStartGame**) we are going to implement these methods:

void ShowStart() **// Will make visible the buttonStartGame**

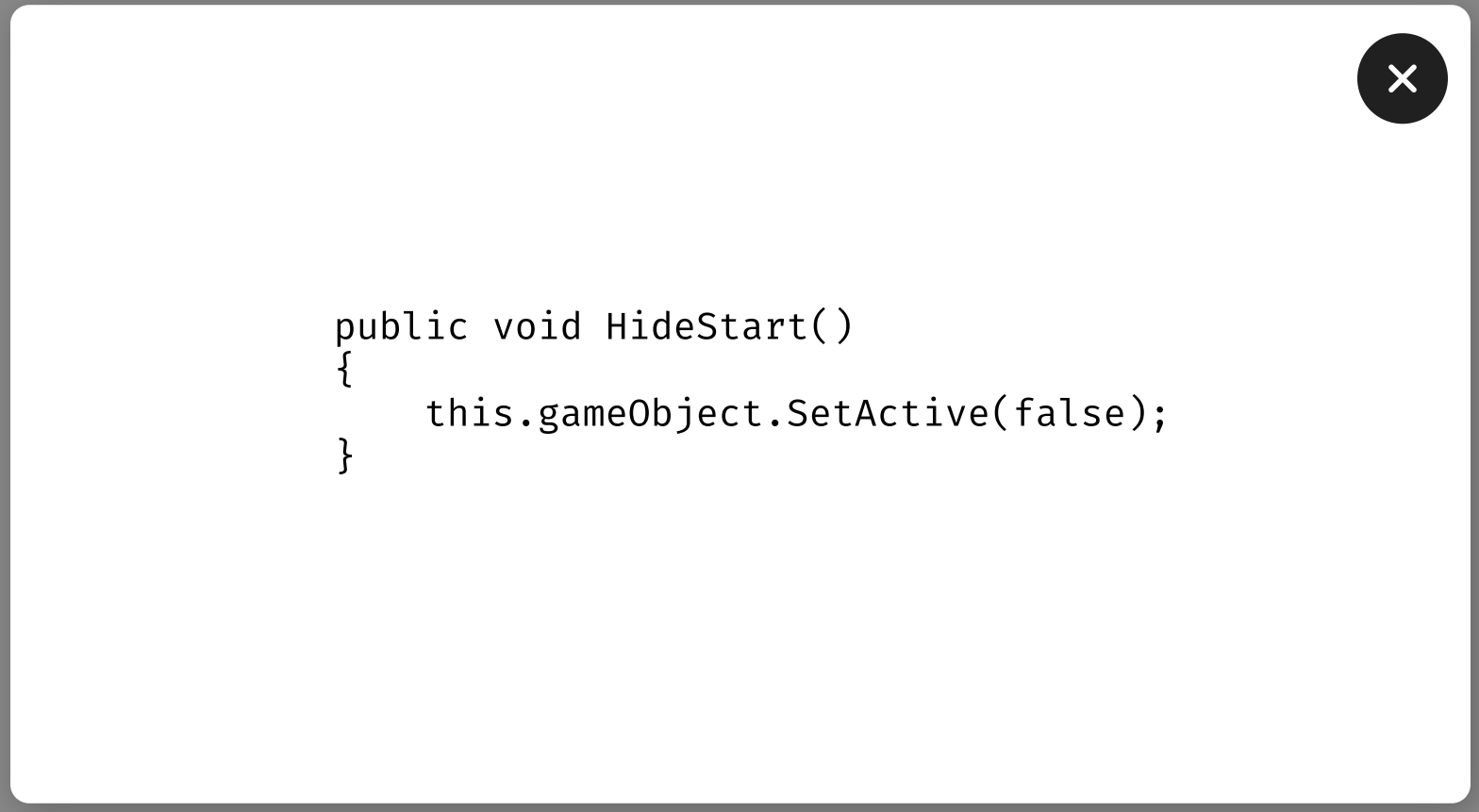
void HideStart() **// Will hide the buttonStartGame**

12) Back to (**GameController**) script , we are going to hide the button when we enter the GAME state.

13) Now you should be able to change to GAME state in Unity Editor and the buttonStartGame should disappear.

14) Do tests with multiple VRChat Instances

`Code Checkpoint: Match Players`

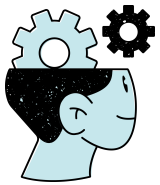


```
public void HideStart()
{
    this.gameObject.SetActive(false);
}
```

```
public class GameController : UdonSharpBehaviour
{
    // OTHER CODE ...

    [SerializeField] private ButtonStartGame buttonStartGame;
```

```
public void ShowStart()
{
    this.gameObject.SetActive(true);
}
```

```
private void StateChanged()
{
    switch (_currentState)
    {
        // OTHER CODE ...
        case GameState.GAME:
            gameHUD.SetState("GAME");
            buttonStartGame.HideStart();
            break;

     // OTHER CODE ...
```



Exercise 4: When the game changes to the state GAME we should reposition the players in a predefined positions.

- Actions:

1) Create 5 gameObjects for each team in both sides of the field where the players will respawn. Disable the MeshRenderer component to make them invisible.

2) Create in (**GameController**) the members:

3) Create an non-visible gameObject in the center of the game field because we will need to reorient the players towards it when they respawn.

4) Still in (**GameController**), create the member and initialize it with the previous GameObject:

5) When changing the state to GAME we are going to teleport each local player to one of the spawning positions.

- Tip 1: Use (GetPlayerTeam) to get player's team
- Tip 2: Use (GetPlayerIndex) to get player's index in array
- Tip 3: Only the same local player can teleport himself Networking.LocalPlayer

6) Test in Unity Editor and test with multiple VRChat Windows instances.

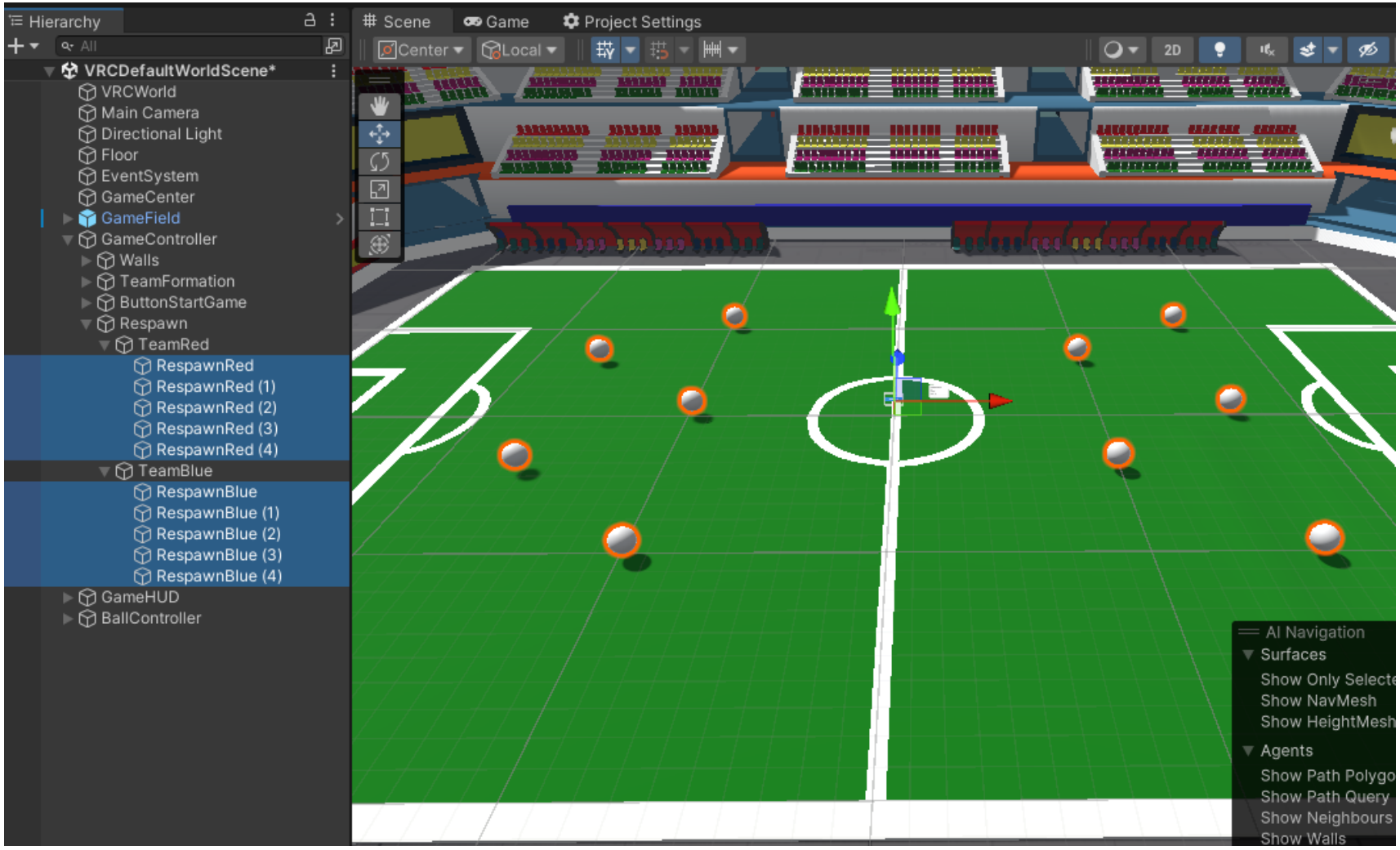
`Code Checkpoint: Respawn for match`

```
public class GameController : UdonSharpBehaviour
{
  // OTHER CODE  ...

  [SerializeField] private GameObject centerField;
```

```
public override void Interact()
{
    if (gameController ≠ null)
    {
        if (_isEnabled)
        {
           Debug.Log("PRESSED START");
        }
     }
}
```

```
public class GameController : UdonSharpBehaviour
{
    // OTHER CODE  ...

    [SerializeField]
    private Transform[] spawnPointsTeamRed;

    [SerializeField]
    private Transform[] spawnPointsTeamBlue;
```

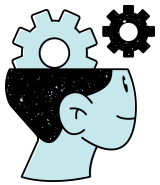
```
private void StateChanged()
{
   switch (_currentState)
   {
     // OTHER CODE  ...

      case GameState.GAME:
         gameHUD.SetState("GAME");
         buttonStartGame.HideStart();

         // RESPAWN PLAYERS IN POSITIONS
         Team teamPlayer = GetPlayerTeam(Networking.LocalPlayer.playerId);
         int indexRespawn = 0;
         switch (teamPlayer)
         {
           case Team.TEAM_BLUE:
              indexRespawn = GetPlayerIndex(Networking.LocalPlayer.playerId);
                Quaternion orientationA = Quaternion.LookRotation((centerField.transform.position -
spawnPointsTeamBlue[indexRespawn].position).normalized, Vector3.up);
                Networking.LocalPlayer.TeleportTo(spawnPointsTeamBlue[indexRespawn].position, orientationA);
                break;
           case Team.TEAM_RED:
                indexRespawn = GetPlayerIndex(Networking.LocalPlayer.playerId);
                Quaternion orientationB = Quaternion.LookRotation((centerField.transform.position -
spawnPointsTeamRed[indexRespawn].position).normalized, Vector3.up);
                 Networking.LocalPlayer.TeleportTo(spawnPointsTeamRed[indexRespawn].position, orientationB);
                 break;
          }
          break;
```


Hierarchy
All
VRCDefaultWorldScene*
VRCWorld
Main Camera
Directional Light
Floor
EventSystem
GameCenter
GameField
GameController
Walls
TeamFormation
ButtonStartGame
Respawn
TeamRed
RespawnRed
RespawnRed (1)
RespawnRed (2)
RespawnRed (3)
RespawnRed (4)
TeamBlue
RespawnBlue
RespawnBlue (1)
RespawnBlue (2)
RespawnBlue (3)
RespawnBlue (4)
GameHUD
BallController
Scene
Game
Project Settings
Center
Local
2D
AI Navigation
Surfaces
Show Only Selecte
Show NavMesh
Show HeightMesh
Agents
Show Path Polygo
Show Path Query
Show Neighbours
Show Walls

LESSON 04 COMPLETED

You now have:

- Working team assignment system
- HUD Debug showing team structure
- Floating identification balls over each player
- Balanced team detection
- Start button activation logic



Self-Evaluation

It’s time to put what we’ve learned into practice! Here are 4 questions to check by yourself what you have learnt in this lesson.

- Question 1
- Question 2
- Question 3
- Question 4

Why are players stored in team arrays (Red and Blue)?

- To visually separate players in the scene hierarchy
- To control player movement speed based on team
- To organize players logically and apply team-based game rules

What is the purpose of assigning an index to each player within a team?

- To uniquely identify a player's position inside the team array
- To decide which avatar the player can use
- To determine the player's walking and running speed

Why are identification balls attached above players?

To improve physics interactions between players

To visually indicate team membership to all players

To give players something to interact with during the game

Why should the Start Game button only be enabled when teams are balanced?

To prevent players from joining the world

To ensure fair gameplay between teams

To reduce network traffic

Help us to improve

Did you understand how players are assigned to teams?

Write your answer here.

Send

Did you understand how identification balls follow players?

Write your answer here.

Send

Which scripting part was the most challenging (team arrays, SetOwner, ball positioning)?

Write your answer here.

Send

How could this lesson be more intuitive or visual?

Write your answer here.

Send