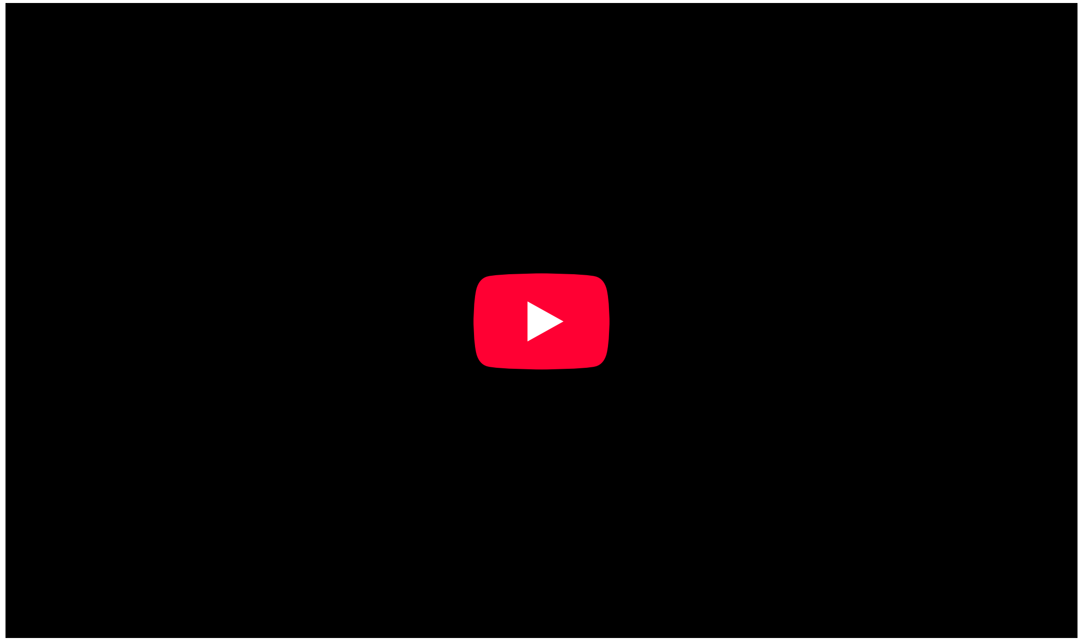
🎓 LESSON 03 State Machine + GameHUD

Start



Video Link

LESSON GOALS

🎯 Goal Summary:

- Build the GameHUD
- Create UI elements for Information, State, and Debug text
- Attach HUD to player camera (desktop & VR compatible)
- Implement the State Machine pattern in GameController
- Synchronize state changes across clients
- Display states on HUD

Next

Learning Objectives + Deliverables

Learning Objectives

Create a worldspace HUD that follows the player

Understand tracking data for VR vs Desktop

Create a clean GameHUD UdonSharp script

Implement the State Machine pattern

Synchronize game state with UdonSynced

Deliverables

✓ A working GameHUD with 3 textfields

✓ A functional state machine

✓ HUD updating when state changes

✓ Network-synchronized states

GENERAL DESCRIPTION OF THE LESSON

High-Level Description

🎯 **Goal:** We are going to create a **GameHUD** where we will display during runtime information that will help us to **debug the game** and later on. We will also implemented the **skeleton of a state machine** that will define the different states that the game is going to be.

🟥 LEVEL 1: Advanced Challenge

In this lesson we will:

◆ Build a **GameHUD** UI that stays in front of the player's view with 3 textfiels (Information, State, Debug)

◈ Implement the state machine that defines core game flow **(NONE, ORGANIZATION, GAME, GAME_OVER, RELOAD)**

◈ Use UdonSynced to synchronize state across all connected players.

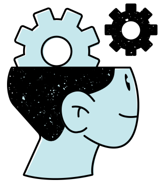
◈ Define a method **StateChanged()** that is run when there is a change in the game state.

◈ Use the GameHUD State textfield to inform about the current state to verify it has changed.

◈ Test state transition to GAME when there are up to 4 VRChat instances connected to the world.

🧠AI Lesson Prompt : State Machine

Next



Exercise 1: Create the GameHUD UI structure in the game scene.

- Actions:

1) Create an empty GameObject in the scene named GameHUD

2) Add an image inside the previous GameObject

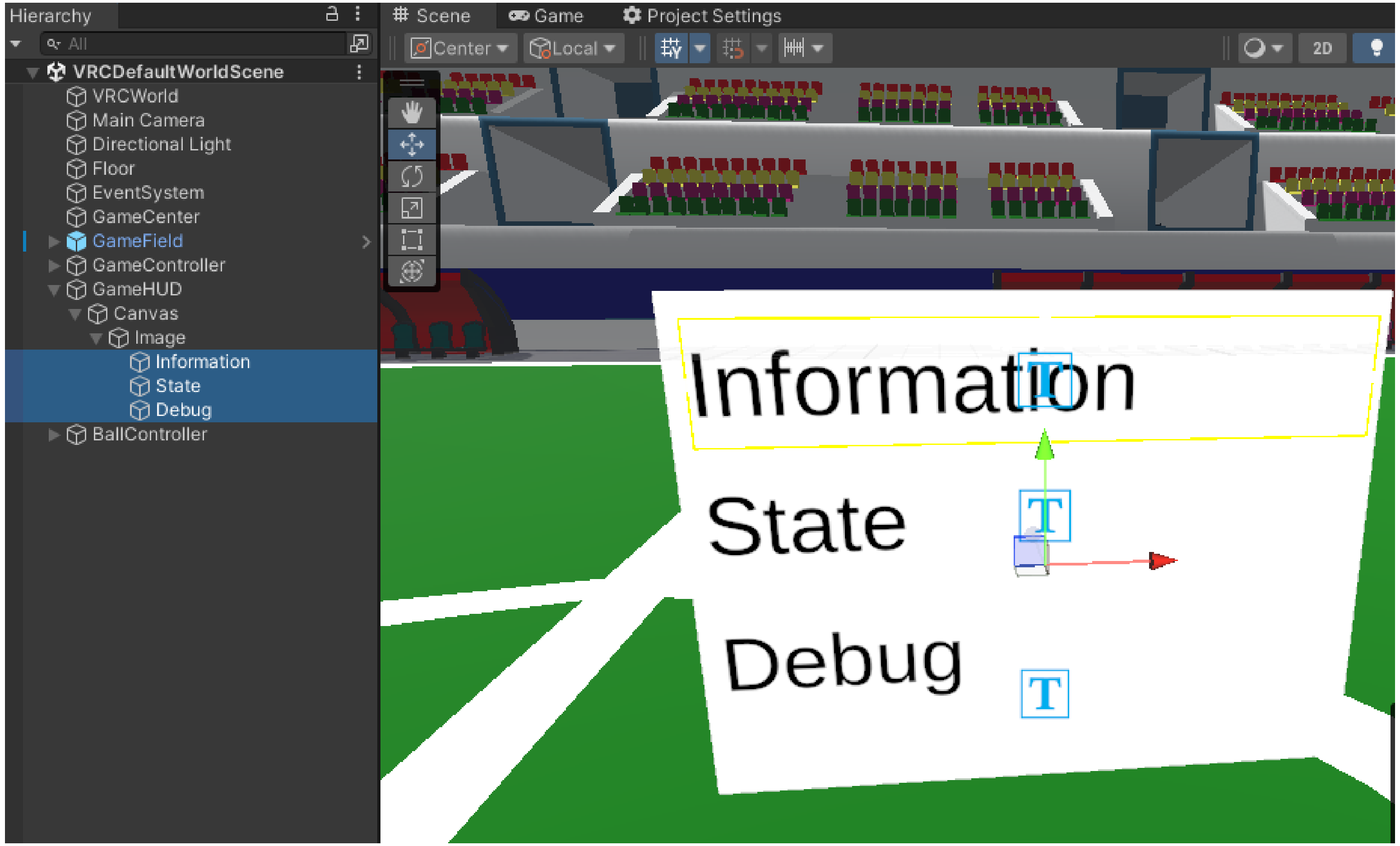
3) Change the RenderMode of the Canvas to WorldSpace

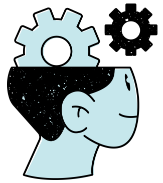
4) Adjust the canvas and image scale and position so it's visible close to the center of the game field.

5) Add 3 Textfields inside the Image and name them (Information, State, Debug)

6) Adjust their size, position and font size until all of them are visible

GameHUD ✓


Hierarchy
All
VRCDefaultWorldScene
VRCWorld
Main Camera
Directional Light
Floor
EventSystem
GameCenter
GameField
GameController
GameHUD
Canvas
Image
Information
State
Debug
BallController
Scene
Game
Project Settings
Center
Local
2D
Information
State
Debug



Exercise 2: Write the **GameHUD** UdonSharp script so it's linked to the camera and positioned depending platform (Desktop, VR).

- Actions:

1) Create a UdonSharp script named **GameHUD** inside the folder **/Game/Scripts/View**

2) Create serialized variable members for the 3 textfields:

3) Add the script as a component of the GameObject in the scene GameHUD and initialize its variable members.

4) Implement the methods that will set the values for the textfields:

5) In the GameHUD script define 2 private members and initialize them in **Start()** method:

6) In the **GameHUD** script, for the **Update**() method implement so the position and orientation of the GameHUD is facing the player. Search online or Ask AI for help.

7) Adjust the position of the GameHUD until you feel ok with it.

Code

```
 void Start() {
        _localPlayer = Networking.LocalPlayer;
        _isVR = _localPlayer.IsUserInVR();
    }
void Update() {
   if (_isVR) {
      VRCPlayerApi.TrackingData head = Networking.LocalPlayer.GetTrackingData(VRCPlayerApi.TrackingDataType.Head);

      Vector3 headPos = head.position;
      Quaternion headRot = head.rotation;
      transform.position = headPos + headRot * new Vector3(0, 0, 2f);
      transform.rotation = headRot;
    }
     else {
      transform.position = _localPlayer.GetPosition() + _localPlayer.GetRotation() * new Vector3(0, 1, 2f);
      transform.rotation = _localPlayer.GetRotation();
   }
}
```

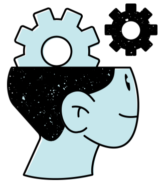
- `public void SetInformation(string text)`
- `public void SetState(string text)`
- `public void SetDebug(string text)`

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

    [SerializeField] private TMP_Text information;
    [SerializeField] private TMP_Text state;
    [SerializeField] private TMP_Text debug;
```

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

    private VRCPlayerApi _localPlayer;
    private bool _isVR;
```



Exercise 3: Create the state machine in **GameController**. and implement that **OnPlayerJoined** there is a transition to the state ORGANIZATION

- Actions in **GameController** script:

1) Create enum GameState with values(NONE, ORGANIZATION, GAME, GAME_OVER, RELOAD)

2) Create the variables that will keep the state:

3) Create a method called **StateChanged()** that will evaluate the exact moment that we change the state. Implement a switch that will evaluate each state.

4) Create a serialized variable member to the GameHUD and initialize it:

6) In the method **StateChanged()** use the **gameHUD.SetState(string text)** method to inform that we have changed of state.

7) On the **OnPlayerJoined()** method**,** for the **local player**, set the local state to ORGANIZATION and call **StateChanged()**. When you play the scene the textfield of the GameHUD should be displaying that information.

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

    [SerializeField]
    private GameHUD gameHUD;
```

```
public override void OnPlayerJoined(VRCPlayerApi player)
{
        Debug.Log("On Player Joined");
        if (Networking.LocalPlayer == player)
        {
            player.SetWalkSpeed(8f);
            player.SetRunSpeed(16f);
            _currentState = GameState.ORGANIZATION;
            StateChanged();
        }
}
```

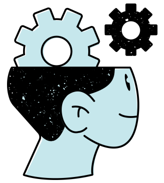
```
private void StateChanged()
{
 switch (_currentState) {
    case GameState.ORGANIZATION:
        break;
    case GameState.GAME:
        break;
    case GameState.GAME_OVER:
        break;
    case GameState.RELOAD:
        break;
    }
}
```

```
public enum GameState
{
    NONE = 0,
    ORGANIZATION = 1,
    GAME = 2,
    GAME_OVER = 3,
    RELOAD = 4
}
```

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

    [UdonSynced, NonSerialized]
    public int syncedState; // Keep the networked synchronized state

    private GameState _currentState; // Keep the local state
private void StateChanged()
{
 switch (_currentState) {
   case GameState.ORGANIZATION:
       gameHUD.SetState("ORGANIZATION");
       break;
   case GameState.GAME:
       gameHUD.SetState("GAME");
       break;
   case GameState.GAME_OVER:
       gameHUD.SetState("GAME_OVER");
       break;
   case GameState.RELOAD:
       gameHUD.SetState("RELOAD");
       break;
   }
}
```



Exercise 4: Enter GAME state when 4 players have entered the world.

- Actions in GameController script:

1) In GameController create a private member that will keep the count of the joined players

2) Increase that counter on method (OnPlayerJoined).

3) Create a method **SetState(GameState newState)** that will allow only the master client to change the state. Remember you need to do a **RequestSerialization()** after you change any network variable to transmit the changes to the rest of connected clients

4) In (**OnPlayerJoined**) method, do an action only for the master client so when the counter of players is over 2, change the state to GAME.

5) On the inherited method (**OnDeserialization**) override in order to detect when the network state has changed in regards the local state to update the state and run (**StateChanged();**) to run the changes linked to the state.

6) Test the system by setting the minimum number of players to transition to GAME state to 4.

```
public void SetState(GameState newState)
{
   if (Networking.IsMaster)
   {
      syncedState = (int)newState;
      _currentState = (GameState)syncedState;
      RequestSerialization();
      StateChanged();
   }
}
```

```
public override void OnDeserialization()
{
   bool isThereChangeState = (_currentState ≠ (GameState)syncedState);
   _currentState = (GameState)syncedState;
   if (isThereChangeState) StateChanged();
}
```

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

    if (Networking.IsMaster)
    {
        if (_currentState == GameState.ORGANIZATION)
        {
            _totalPlayersInGame++;
            if (_totalPlayersInGame >= 2)
            {
                SetState(GameState.GAME);
            }
        }
    }
}
```

LESSON 03 COMPLETED

You now have:

- A fully functional GameHUD
- A synchronized state machine
- A scalable foundation for all future lessons

Code Checkpoint: State Machine

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

What is the main purpose of using a state machine in the game?

To control player movement and animations	To organize and control the game flow through well-defined states	To replace networking logic

Why is the GameHUD implemented as a world-space UI that follows the player?

- Because screen-space UI does not work in Unity
- To improve game performance
- To ensure the UI works consistently in both Desktop and VR mode

What is the role of an UdonSynced variable in the state machine?

To store values permanently on the local machine	To make variables editable in the Unity Inspector	To synchronize important values, such as the game state, across all players

When should the StateChanged() method be called?

Whenever the game state changes

Only when the game starts

Every frame inside Update()

Help us to improve

Did you understand how the GameHUD follows the player?

Write your answer here.

Send

Do you feel confident with the concept of a state machine?

Write your answer here.

Send

Was the explanation of UdonSynced variables clear?

Write your answer here.

Send

What extra examples could make state machines clearer?

Write your answer here.

Send