A2526_3384 - Prog. Moteur de Jeu (C)

Par Fergal MECHIN, modifié le 23 octobre 2025

Année scolaire 2025-2026

Pour la classe Lyon | GPROG MAST1

Netcode

Sync var & RPCs

NetworkObject

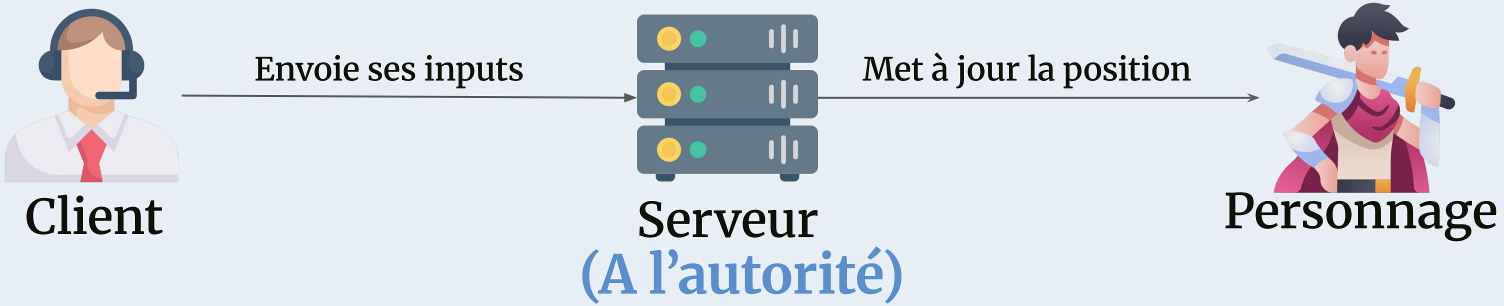
=> Tous les objets sont locaux par défaut

=> On indique quels objets synchroniser en lui ajoutant un NetworkObject

Autorité (Authority)

=> Définit qui a le dernier mot sur l’état d’un objet :

Dans notre cas, il s’agira toujours du serveur.



Client

Envoie ses inputs

Serveur

(A l’autorité)

Met à jour la position

Personnage

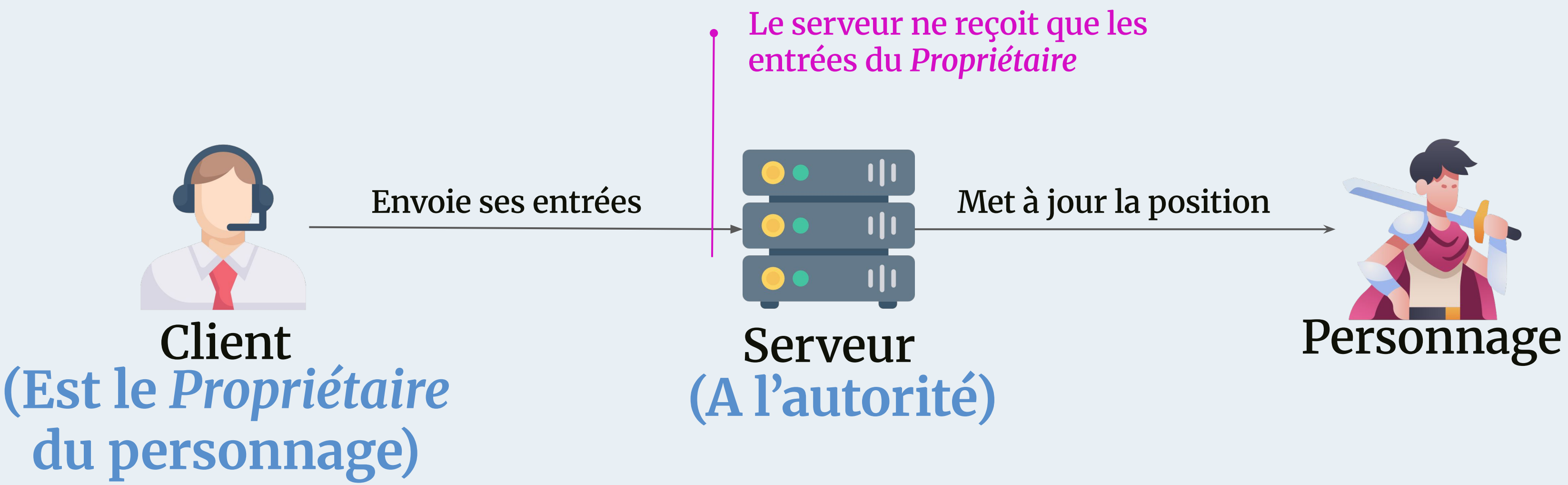
Possession (Ownership)

=> Permet d’associer un NetworkObject à un joueur.

2 cas d’utilisation :

1. Simplement faire un lien avec un joueur
   - *C’est son personnage, son arme, son piège, etc…*
2. Permettre d’avoir **des accès privilégiés en Ecriture ou Lecture**
   - *Seul le propriétaire peut lancer une compétence sur ce personnage*

Possession (Ownership)



Le serveur ne reçoit que les entrées du *Propriétaire*

Envoie ses entrées

Met à jour la position

Client

(Est le *Propriétaire* du personnage)

Serveur

(A l'autorité)

Personnage

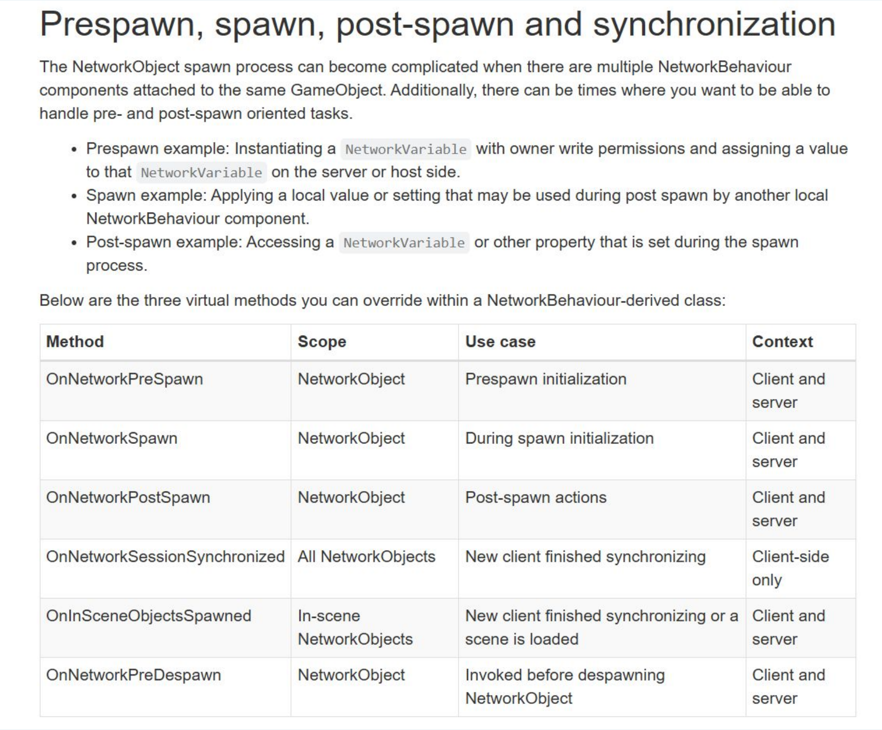
NetworkBehaviour

=> Permet d’**ajouter des comportements** aux NetworkObjects

Sur vos NetworkObjects, pour ajouter des comportements synchronisés sur le réseau, héritez de ***NetworkBehaviour*** plutôt que de ***MonoBehaviour***.

Netcode

Callbacks



Prespawn, spawn, post-spawn and synchronization

The NetworkObject spawn process can become complicated when there are multiple NetworkBehaviour components attached to the same GameObject. Additionally, there can be times where you want to be able to handle pre- and post-spawn oriented tasks.

- Prespawn example: Instantiating a `NetworkVariable` with owner write permissions and assigning a value to that `NetworkVariable` on the server or host side.
- Spawn example: Applying a local value or setting that may be used during post spawn by another local NetworkBehaviour component.
- Post-spawn example: Accessing a `NetworkVariable` or other property that is set during the spawn process.

Below are the three virtual methods you can override within a NetworkBehaviour-derived class:

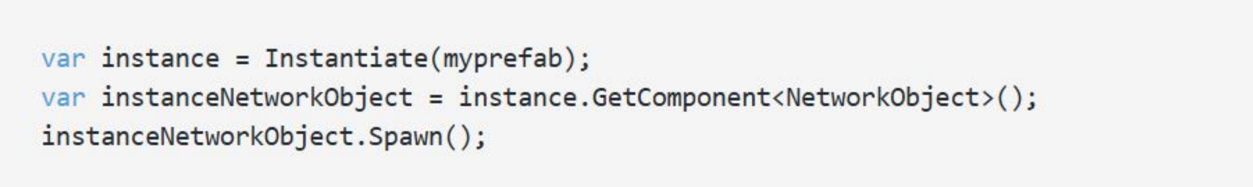
Method	Scope	Use case	Context
OnNetworkPreSpawn	NetworkObject	Prespawn initialization	Client and server
OnNetworkSpawn	NetworkObject	During spawn initialization	Client and server
OnNetworkPostSpawn	NetworkObject	Post-spawn actions	Client and server
OnNetworkSessionSynchronized	All NetworkObjects	New client finished synchronizing	Client-side only
OnInSceneObjectsSpawned	In-scene NetworkObjects	New client finished synchronizing or a scene is loaded	Client and server
OnNetworkPreDespawn	NetworkObject	Invoked before despawning NetworkObject	Client and server

Spawning

Pour qu’un ***NetworkObject*** soit correctement instancié sur le réseau, il faut :

1. L’ajouter au dictionnaire des objets instanciables
   a. Sur le NetworkManager, trouver le ***NetworkprefabsList***
   b. Y ajouter le prefab du *NetworkObject* à instancier

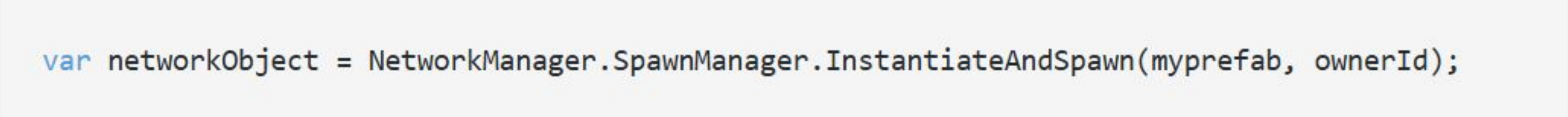
2. Appeler sa méthode Spawn



```
var instance = Instantiate(myprefab);
var instanceNetworkObject = instance.GetComponent<NetworkObject>();
instanceNetworkObject.Spawn();
```

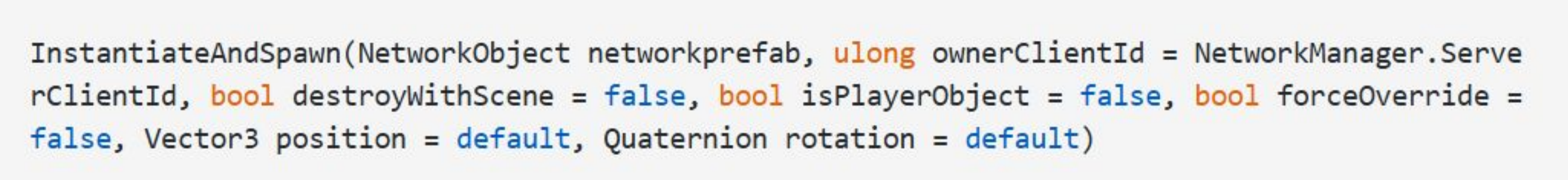
Spawning - Alternative

Alternativement…



```
var networkObject = NetworkManager.SpawnManager.InstantiateAndSpawn(myprefab, ownerId);
```

Ses paramètres :



```
InstantiateAndSpawn(NetworkObject networkprefab, ulong ownerClientId = NetworkManager.Serve
rClientId, bool destroyWithScene = false, bool isPlayerObject = false, bool forceOverride =
false, Vector3 position = default, Quaternion rotation = default)
```

Despawning

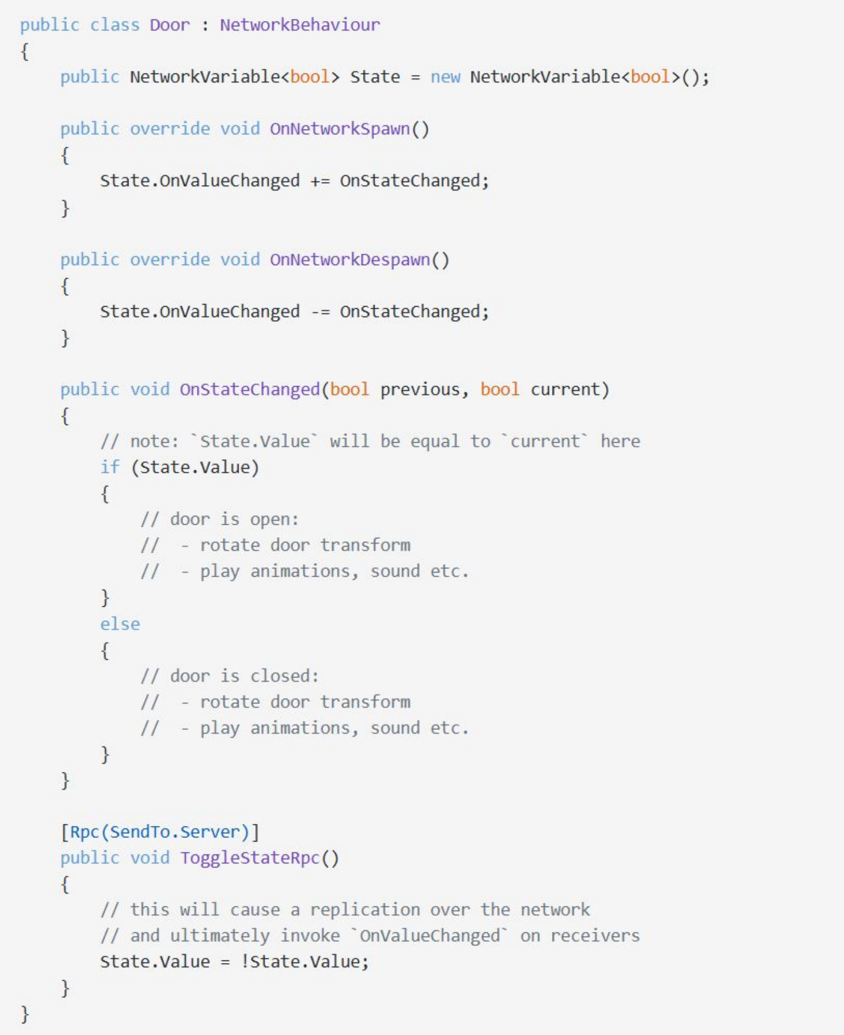
=> Un **simple Object.Destroy() depuis l'autorité** (Le serveur) **détruit proprement l'objet** chez tout le monde.

=> La méthode **NetworkObject.Despawn() retirera simplement l'objet des objets synchronisés** mais **restera dans la scène en local**.

Légendes

Netcode

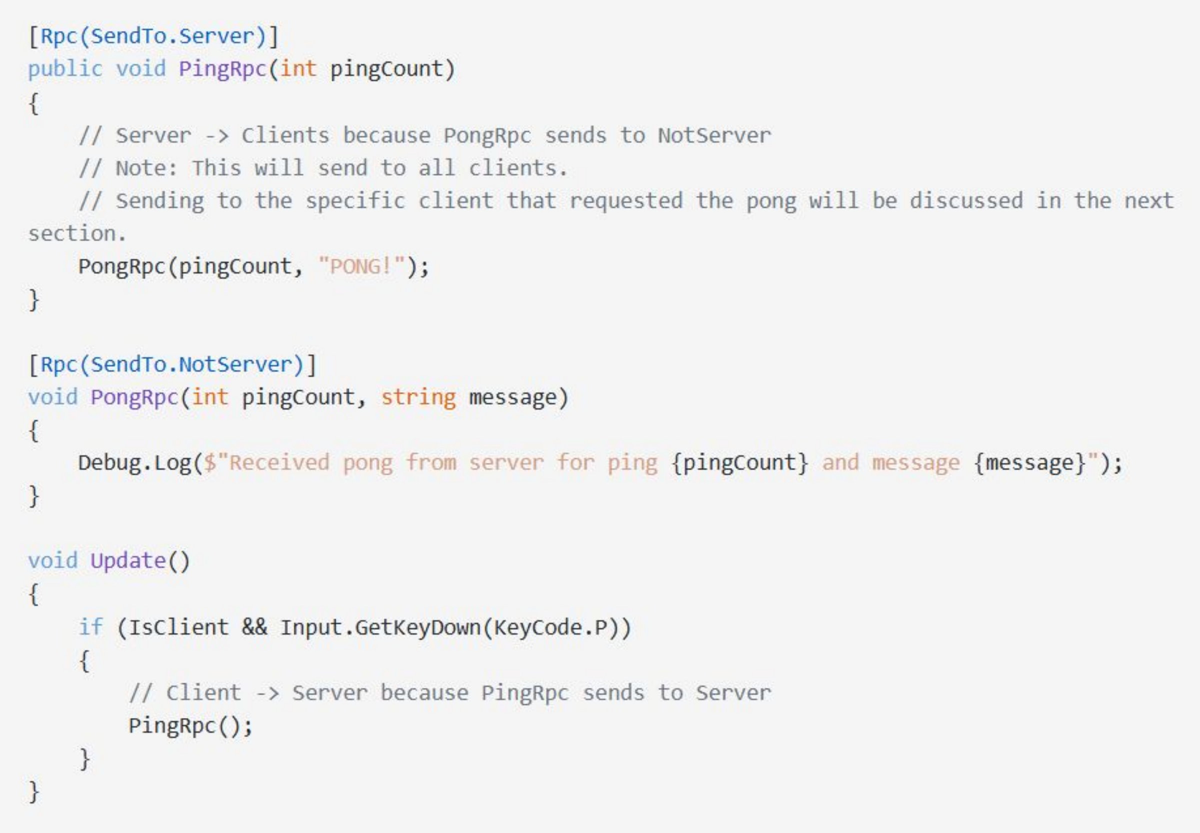
Network Variable



```csharp
public class Door : NetworkBehaviour
{
    public NetworkVariable<bool> State = new NetworkVariable<bool>();

    public override void OnNetworkSpawn()
    {
        State.OnValueChanged += OnStateChanged;
    }

    public override void OnNetworkDespawn()
    {
        State.OnValueChanged -= OnStateChanged;
    }

    public void OnStateChanged(bool previous, bool current)
    {
        // note: `State.Value` will be equal to `current` here
        if (State.Value)
        {
            // door is open:
            //  - rotate door transform
            //  - play animations, sound etc.
        }
        else
        {
            // door is closed:
            //  - rotate door transform
            //  - play animations, sound etc.
        }
    }

    [Rpc(SendTo.Server)]
    public void ToggleStateRpc()
    {
        // this will cause a replication over the network
        // and ultimately invoke `OnValueChanged` on receivers
        State.Value = !State.Value;
    }
}
```

Netcode

RPCs



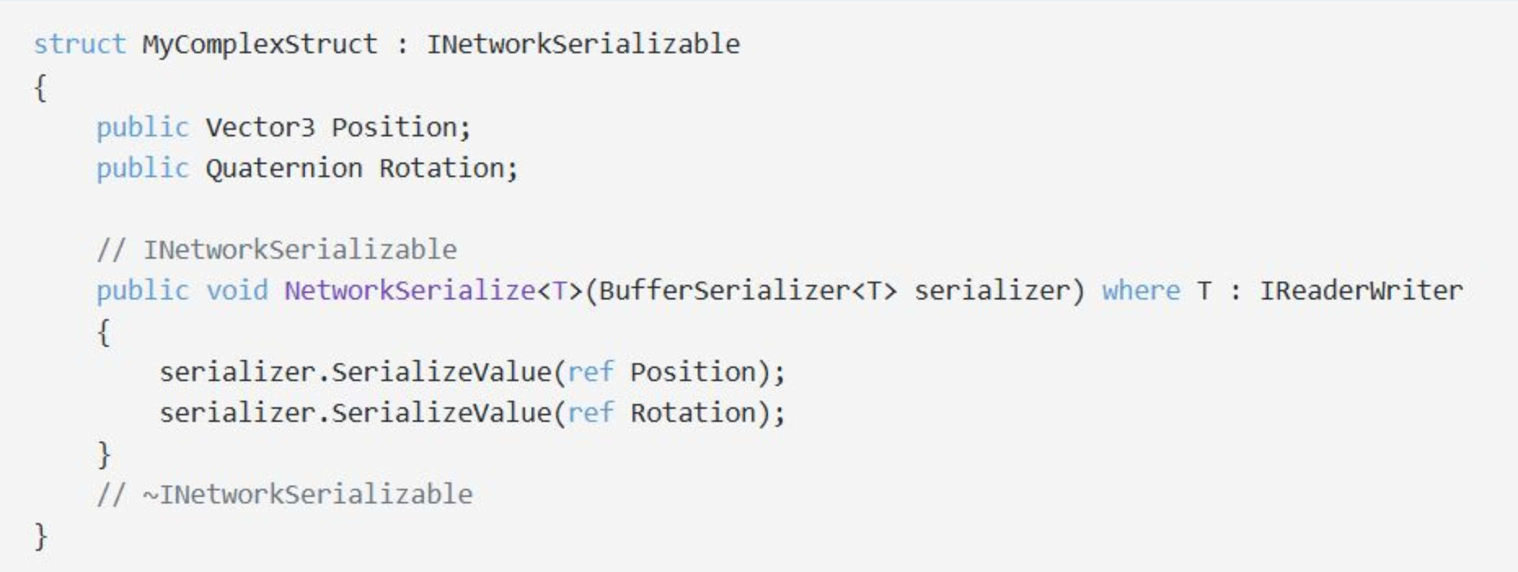
```
[Rpc(SendTo.Server)]
public void PingRpc(int pingCount)
{
    // Server -> Clients because PongRpc sends to NotServer
    // Note: This will send to all clients.
    // Sending to the specific client that requested the pong will be discussed in the next
section.
    PongRpc(pingCount, "PONG!");
}

[Rpc(SendTo.NotServer)]
void PongRpc(int pingCount, string message)
{
    Debug.Log($"Received pong from server for ping {pingCount} and message {message}");
}

void Update()
{
    if (IsClient && Input.GetKeyDown(KeyCode.P))
    {
        // Client -> Server because PingRpc sends to Server
        PingRpc();
    }
}
```

Sérialisations gérées par défaut

Les RPCs et les NetworkVariables prennent en paramètres tous les types nativement sérialisables :

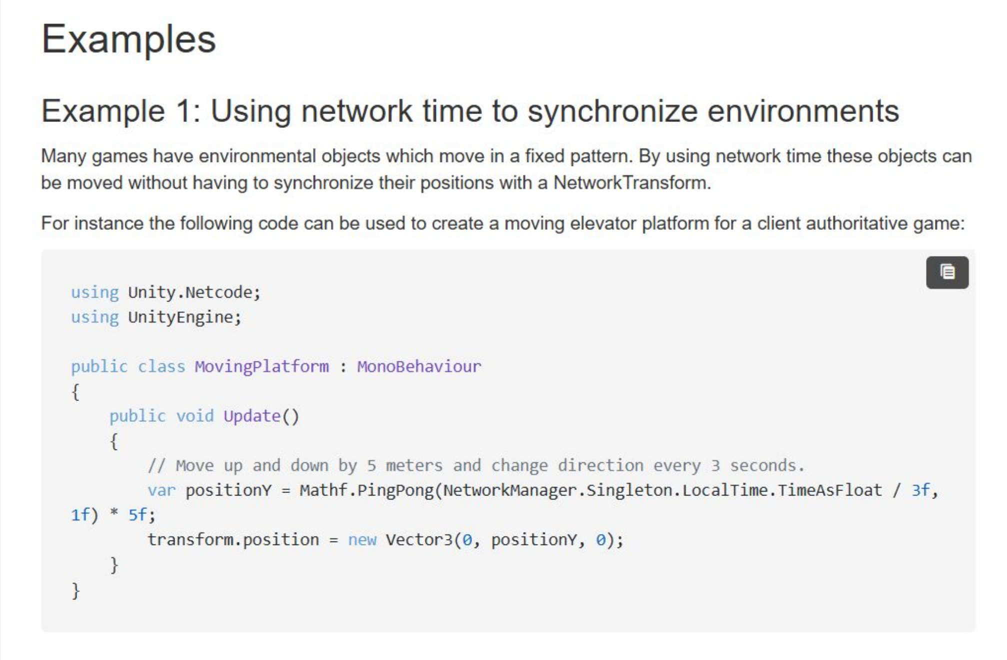
- Primitives C#
  - float, int, string, etc…
- Primitives Unity
  - Color, Vector3, etc…
- Les énumérations
- Les Arrays et les listes avec ***NativeArray*** et ***NativeList***

Sérialisations personnalisées



```
struct MyComplexStruct : INetworkSerializable
{
    public Vector3 Position;
    public Quaternion Rotation;

    // INetworkSerializable
    public void NetworkSerialize<T>(BufferSerializer<T> serializer) where T : IReaderWriter
    {
        serializer.SerializeValue(ref Position);
        serializer.SerializeValue(ref Rotation);
    }
    // ~INetworkSerializable
}
```

Gestion du temps



Examples

Example 1: Using network time to synchronize environments

Many games have environmental objects which move in a fixed pattern. By using network time these objects can be moved without having to synchronize their positions with a NetworkTransform.

For instance the following code can be used to create a moving elevator platform for a client authoritative game:

```
using Unity.Netcode;
using UnityEngine;

public class MovingPlatform : MonoBehaviour
{
    public void Update()
    {
        // Move up and down by 5 meters and change direction every 3 seconds.
        var positionY = Mathf.PingPong(NetworkManager.Singleton.LocalTime.TimeAsFloat / 3f,
1f) * 5f;
        transform.position = new Vector3(0, positionY, 0);
    }
}
```

https://docs.unity3d.com/Packages/com.unity.netcode.gameobjects@2.7/manual/advanced-topics/networktime-ticks.html

Crédits

Images prises sur Flaticon :
Icônes gratuites conçues par kerismaker | Flaticon
Icônes gratuites conçues par DinosoftLabs | Flaticon
Icônes gratuites conçues par max.icons | Flaticon