|  |  |
| --- | --- |
| 6C5- MultiJoueur - V1  Tutoriel et Formatif no 4 | |
| **Nom, prénom et DA** | * Nom, Prénom (1234567) |
| **Cours** | **6C5 – Applications de jeux et simulations II** |
| **Enseignant** | Julien Brunet |

Table des matières

[NetCode For GameObjects 1](#_Toc196465933)

[Formatif : Construire un premier Jeu Multiplayer 1](#_Toc196465934)

[Préalable : 1](#_Toc196465935)

[Mise en place de NetCode 1](#_Toc196465936)

[Création des boutons host, client et la console 2](#_Toc196465937)

[Ne contrôler que son joueur 3](#_Toc196465938)

[Utiliser les NetworkVariable< > 4](#_Toc196465939)

[Autres formes de synchronisations : les RPC (Remote Procedure Calls) 5](#_Toc196465940)

[Références : 7](#_Toc196465941)

# NetCode For GameObjects

Disponible depuis 2021.3.10 on utilise le package Netcode for GameObjects. Il en existe d’autres (Mirror en particulier), celui-ci semble être la solution préconisée par Unity!

# Formatif : Construire un premier Jeu Multiplayer

Préalable :

1. Cloner le projet 2.01-Multiplayer
2. Tester le déplacement et le shoot du joueur.
3. S’approprier le code et son fonctionnement

Mise en place de NetCode

1. Installer **NetCode for GameObjects** et **Multiplayer tools** depuis le package manager.
2. Créer un Empty GameObject nommé **NetworkManager** et y ajouter le composant : [NetworkManager](https://docs-multiplayer.unity3d.com/netcode/current/components/networkmanager/index.html). Il doit être unique dans la scene et est fondamental pour un jeu multijoueur.
3. Sectionner le **UnityTransport** layer dans l’inspecteur.
4. Créer un prefab avec le joueur et supprimer la capsule de la Scène.
5. Ajouter un composant [NetworkObject](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/index.html) sur le **Player** **Prefab** et faire hériter le Player Controller de NetworkBehaviour.
6. Ajouter le Player Prefab dans l’inspecteur du **NetworkManager**.

Une image contenant capture d’écran, texte, Logiciel multimédia, logiciel

Description générée automatiquement

1. Exécuter et tester la configuration **Host**

**Remarque** : Il est important de comprendre l’utilité à ce stade du [NetworkManager](https://docs-multiplayer.unity3d.com/netcode/current/components/networkmanager/index.html) et du [NetworkObject](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkobject/index.html)  (attention à bien faire hériter les scripts du NetworkObject de NetworkBehaviour)

Création des boutons host, client et la console

1. Ajouter un **canvas** dans la scene, le nommer NetworkUI
2. Créer un bouton nommé **HostButton** et ancrez-le en haut à gauche (s’assurer qu’il est bien ancré pour tous les aspect ratios). Mettre le texte Host sur le bouton.
3. Dupliquer pour créer un **ClientButton.**
4. Créer le script NetworkManagerUI (en annexe au besoin) de telle sorte que les boutons lancent les actions du NetworkManager.
5. Tester le Host Button.
6. Dans le NetworkManager, régler le log level sur Developper
7. Installer la **InGameDebugConsole** de l’Asset store puis l’ajouter depuis package manager.
8. Ajouter son prefab à la scene.
9. Tester, en mettant un log quelque part!
10. Vérifier les builds settings (la scene en particulier) et le mode windowed dans ProjectSettings/Player/Resolution and Presentation), puis faire un build. Tester le host et le client. Il semble n’y avoir qu’un seul joueur… En fait ce n’est pas tout à fait ça… En fait **on contrôle les deux joueurs en même temps.**

Ne contrôler que son joueur

1. Pour régler le problème, modifier **PlayerController** pour ne faire l’update que si **IsOwner** est vrai.
2. Tester. On ne voit pas le mouvement sur l’autre instance, pas de synchronisation!
3. Ajouter au **player** un [NetworkTransform](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/)  (ne synchroniser que ce qui doit l’être c’est couteux!).
4. Tester. C’est mieux mais le client ne bouge pas! Par défaut le système est configuré en [**Autorité au Serveur**](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/#server-authoritative-mode).
5. Ajouter un nouveau script ClientNetworkTransform.cs qui hérite de NetworkTransform et surcharger la méthode :

protected bool OnIsServerAuthoritative() pour la faire retourner false

1. Remplacer le composant NetworkTransform du player par celui-ci. (Ajouter le NetworkRigidbody comme demandé).
2. Tester.

**Remarque** :

* Il est important de comprendre l’utilité à ce stade du [NetworkTransform](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/) et du [NetworkBehaviour](https://docs.unity3d.com/Manual/class-NetworkBehaviour.html).
* Il faut également bien comprendre le principe de [ServerAuthority](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/#authority-modes) et les raisons du [ClientNetworkTransform](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/#owner-authoritative-mode)
* La solution qui consiste à donner l’autorité au client n’est pas la seule solution possible, on verra avec les appels RPC comment conserver une architecture [**Autorité au Serveur**](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/#server-authoritative-mode) à la place d’une [**Autorité distribuée**](https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/#owner-authoritative-mode)

|  |
| --- |
| **Exercice 1** : Écrire un message de log qui affiche au Awake() du joueur ses attributs : IsOwner, IsServer et OwnerClientId  Interpréter le résultat. Pourquoi IsOwner est false?  Déplacer maintenant le log dans la méthode Start().  Interpréter le résultat, puis lancer le client aussi.  Combien de messages sont affichés? Interpréter!  Faire de même avec OnNetworkSpawn(). |
|  |

A ce stade, vous êtes en mesure de réaliser la première partie du TP2!

Utiliser les NetworkVariable< >

[NetworkVariable](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/) est une classe qui permet d’avoir une valeur synchronisée entre toutes les instances!

On veut créer une variable qui compte les appuis sur un touche (par exemple T) et voir cette valeur partagée entre clients et host par un log chez chaque joueur.

1. Ajouter un script TestCounter.cs sur un empty GameObject.
2. Ajouter un attribut private NetworkVariable<int> counter = new NetworkVariable<int>(0);
3. Ajouter le code permettant d’incrémenter la variable à chaque appui sur la touche T.
4. Ajouter un log affichant la variable.
5. Tester dans le host. Constat? Ok mais on a un warning qui nous fait prendre conscience que rien n’est synchronisé!
6. Corriger! (ajouter un NetworkObject sur le GameObject et faire hériter le script de NetworkBehaviour!)
7. Tester (avec un client ) en appuyant sur T (dans le host). Constat? Le client ne log pas de message. Normal…
8. Corriger en implémentant un mécanisme d’écoute de la modification de la variable (voir TestCounter.cs)
9. Tester : Constat? Le log est effectué dans chaque client!! Mais le client n’est pas autorisé à écrire dans la variable. **On va corriger cela plus tard!!!!**

**Remarque** :

* Notons qu’une [NetworkVariable](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/) doit toujours être utilisée dans un [NetworkBehaviour](https://docs.unity3d.com/Manual/class-NetworkBehaviour.html) et être initialisée
* Notons également que par défaut **elles ne peuvent pas** être lues par un client. Les paramètres [NetworkVariablePermissions](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable#permissions)  du constructeur de [NetworkVariable](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/) permettent d’y remédier en cas de besoin.
* Notons qu’une [NetworkVariable](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/) ne supporte comme type que [les types primitifs de C#, ainsi que quelques autres](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/#supported-types), mais pas n’importe quelle classe.
* Optionnel : On peut utiliser les [NetworkVariable](https://docs-multiplayer.unity3d.com/netcode/current/basics/networkvariable/) avec des types de données Custom qui implémentent INetworkSerializable Les informations sont [ici](https://www.youtube.com/watch?v=3yuBOB3VrCk) (minutage 29 :43)

|  |
| --- |
| **Exercice 2** : implémenter un compteur de joueurs comme sur l’image suivante! |
| Une image contenant texte, capture d’écran, Police, nombre  Description générée automatiquement |

Autres formes de synchronisations : [les RPC](https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/messaging-system/index.html) (Remote Procedure Calls)

On veut utiliser les RPC (Remote Procedure Calls) pour assurer qu’une méthode s’exécute sur une instance particulière (cela peut-être le server (SendTo.Server) , le owner (SendTo.Owner) ou d’autres). Voir la [documentation](https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/rpc/).

1. Pour résoudre le problème de l’écriture de la variable précédente par le client, on va simplement encapsuler la modification de la valeur dans une méthode

[Rpc(SendTo.Server)]

private void IncrementCounterRpc()

{

counter.Value++;

}

1. Tester!

**Remarque** :

* Il est fondamental de bien maitriser la mécanique de Rpc.
* Notons que les méthodes RPC doivent être précédée d’un attribut [Rpc(…)] et respecter la convention de nommage de **terminer le nom de la méthode par Rpc**

|  |
| --- |
| **Exercice 3.1** : Modifier le PlayerController pour configurer le mouvement en autorité au serveur. (Attention a s’assurer que la vitesse sur le client et sur le serveur soient cohérentes) |
|  |

|  |
| --- |
| **Exercice 3.2** : Corriger le shoot! Pour cela il faut ajouter le prefab de projectile à la liste NetworkPrefabsList, ajouter les NetworkObjects et les NetworkTransforms. On réalise ensuite que l’instantiation d’un objet ne suffit pas, il doit également être « Spawn(…) »! Et celui-ci ne peut être fait que par le serveur. C’est le moment d’utiliser nos Rpc…  Attention, la question du destroy n’est pas triviale, commencez par le supprimer, puis améliorez… |

Pour votre culture, on peut aussi utiliser les Client RPC (Remote Procedure calls running on all Clients (SendTo. **ClientsAndHost**)) pour assurer qu’une méthode s’exécute sur **chacun** des clients. Voir la [documentation](https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/clientrpc). Cela peut être utile pour gérer correctement l’effet de particule associé au hit du projectile…

**Remarque** :

* Notons que les méthodes Client RPC doivent être précédée d’un attribut [Rpc(SendTo. **ClientsAndHost)**] et respecter la convention de nommage de terminer le nom de la méthode par Rpc

|  |
| --- |
| **Exercice 4 :** Améliorer le spawn du joueur à des positions aléatoires et faites-le faire face à la caméra! |

|  |
| --- |
| **Exercice 5** : Corriger l’effet de particule pour les impacts de balle! |

|  |
| --- |
| **Exercice 6** : Ajouter une couleur spécifique à chaque joueur pour les distinguer. |

|  |
| --- |
| **Exercice 7** (facultatif): Implémenter un mécanisme de score. |

Scripts :

NetworkManagerUI.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using Unity.Netcode;  using UnityEngine;  using UnityEngine.UI;  public class NetworkManagerUI : MonoBehaviour  {  [SerializeField] private Button HostButton;  [SerializeField] private Button ClientButton;  private void Awake()  {  HostButton.onClick.AddListener(() =>  {  NetworkManager.Singleton.StartHost();  });  ClientButton.onClick.AddListener(() =>  {  NetworkManager.Singleton.StartClient();  });    }  } |

ClientNetworkTransform.cs

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using Unity.Netcode.Components;  using UnityEngine;  public class ClientNetworkTransform : NetworkTransform  {  protected override bool OnIsServerAuthoritative()  {  return false;  }  } |

# Références :

Unity Netcode for game objects référence doc :

[https://docs-multiplayer.unity3d.com/netcode/current/about/index.html](https://docs-multiplayer.unity3d.com/netcode/current/about/index.html%3c)

Des vidéos d’introduction au concept de Multiplayer :

<https://www.youtube.com/watch?v=HWPKlpeZUjM>

<https://www.youtube.com/watch?v=3yuBOB3VrCk>