Documentation technique

Partie multi/réseau

Marine Mangan

Nicolas Jatob

Alexis Clement

Année : 2021-2022

Sommaire

Présentation Netcode

Netcode for Gameobjects est une librairie pour Unity. Elle permet de faire des communications réseaux facilement et a l'avantage d'être maintenue par Unity directement. Il s'agit d'une bibliothèque obligatoire pour tout projet multijoueur sur Unity. Au moment de notre projet, nous avons commencé à l'utiliser dans sa version 0.1 (appelé alors MLAPI) puis suite à une mise à jour vers la 1.0 (Netcode) nous avons alors mis à jour le projet en 1.0. En revanche, Unity continue de développer la solution et de futures mises à jour permettront surement à ce projet de s'améliorer davantage.

Lien vers la documentation officielle : https://docs-multiplayer.unity3d.com

Version supportée

Version d'Unity supportée : 2020.3 – 2021.1 – 2021.2

Supporte également le Mono et le IL2CPP.

Fonctionne sous Windows, linux, MacOs, IOS et Android.

Si combiné avec de la XR, fonctionne sous Windows, Android et IOS.

Fonctionne normalement sur console également.

Ne supporte pas WebGL.

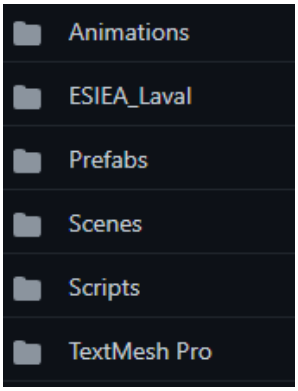
Installation

Actuellement, Netcode n'apparait pas encore dans la liste des packages Unity.

Pour l'installer il va donc falloir procéder de cette façon : ouvrir le package manager dans un projet Unity, cliquer sur ajouter un nouveau package depuis une URL Git et insérer l'URL suivante `com.unity.netcode.gameobjects`

Architecture du projet

Dans le dossier Assets, nous retrouvons tous les dossiers et fichiers importants du projet :



Le dossier animation contient les différents fichiers nécessaires aux animations dans l'école. Cela inclut les vidéos et musiques par exemple sous forme de mp4 et mp3.

ESIEA_Laval est le dossier contenant la maquette 3D de l'école. Il s'agit ici du campus de Laval, celui de Paris n'ayant pas encore été modélisé.

Prefabs contient les différents prefab du projet.

Scenes contient les différentes scènes du projet. La scène finale qui contient l'entièreté du projet est la scène : Multiplayer. Les autres scènes sont des tests des différentes features du projet.

Scripts contient tous nos scripts pour ce projet.

TextMesh Pro contient les dossiers nécessaires pour les composants d'affichage du chat.

Détails du code

Serveur

Pour la gestion du serveur, on retrouve cela dans le fichier defaultPlayer.cs.

La première chose importante pour gérer une classe de type serveur, est de définir son comportement comme un networkbehaviour.

```
public class DefaultPlayer : NetworkBehaviour
{
```

Dans ce genre de classe, il est alors possible de créer des variables qui seront automatiquement partagées entre les différents clients du serveur. Ces variables s'appellent des **NetworkVariable**. Leurs types se composent de NetworkVariable puis du type de variable que l'on veut. Par exemple si l'on souhaite partagé un vecteur3 alors la variable est `NetworkVariable<Vector3> Position`

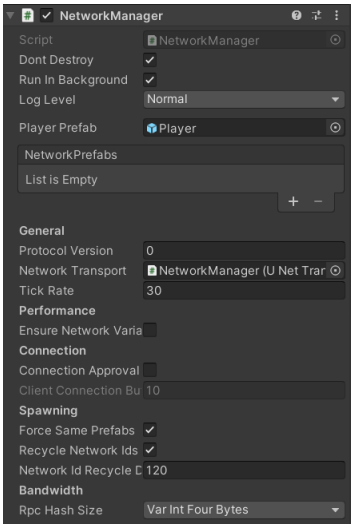
Il existent la majorité des types de base (int, float, char…) ainsi que certains types plus complexes comme les vecteur3 mais tous en sont pas encore implémentés en 1.0, par exemple, le type String n'existe pas encore.

Enfin, les méthodes qui seront appelées par les clients afin d'intéragir avec le serveur doivent avoir un **tag [ServerRpc]** avant la méthode ainsi que ce nom dans la méthode :

```
[ServerRpc]
public void setServerRpc(string message, ServerRpcParams rpcParams = default)
```

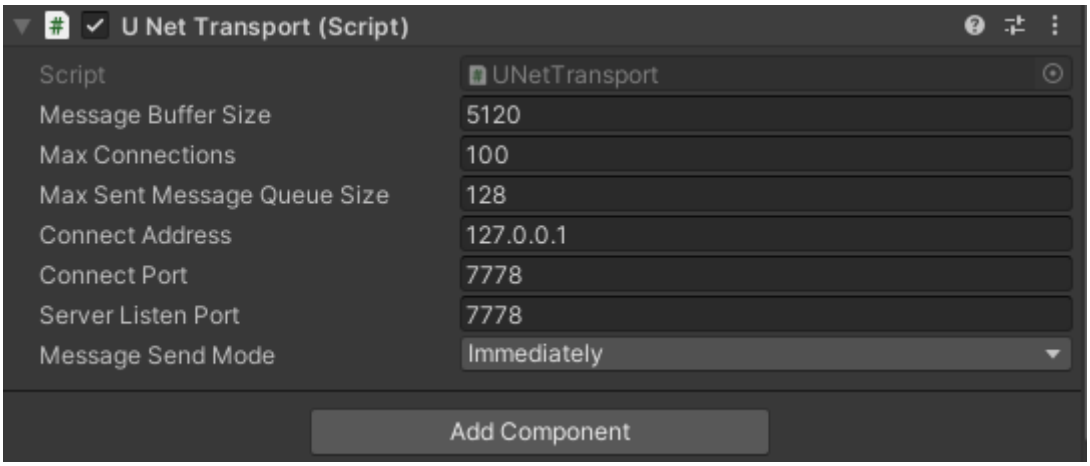
Il est évidemment possible d'appeler d'autres méthodes dans celles-ci, et ces méthodes n'auront pas besoin de respecter cette règle si elles sont privées et non accessible aux clients.

Du côté d'Unity, il faut avoir un **composant NetworkManager** (un script de l'API) attaché à un objet afin de faire fonctionner le serveur. C'est sur ce composant que de nombreux paramètres du serveur pourront être ajuster.



Une fois le script complété et cet objet dans la scène Unity, il est alors possible d'appeler ces méthodes afin de faire interagir des clients et le serveur.

On configure les paramètres du serveur (adresse IP, nombre de connexions…) directement dans le script dédié depuis l'éditeur d'Unity.



Il est également possible de lancer le serveur en ligne de commande avec la commande :

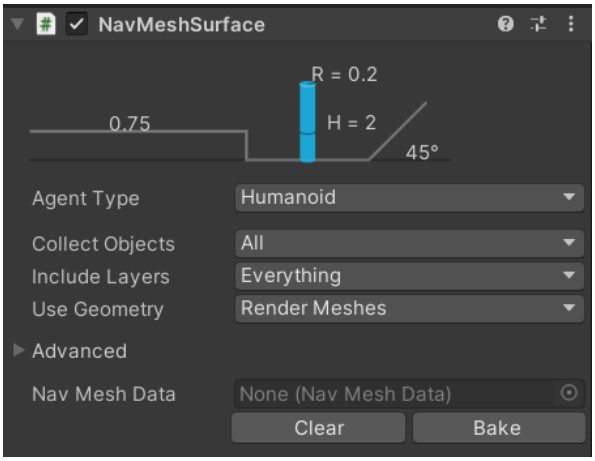
.\Virtual-Esiea-Multiplayer.exe -mlapi server -batchmode -nographics -logfile ./output.log

En revanche, il s'agit d'un point que nous n'avons pas eu le temps de peaufiner cette partie, il se peut donc que des bugs soient découvert plus tard.

Déplacements

Pour pouvoir se déplacer dans l'école, on utilise principalement un **Navmesh** ainsi qu'un NavmeshAgent.

On crée tout d'abord un navmeshsurface à la racine du modèle de l'école (Il faut que le composant soit bien à la racine afin de pouvoir détecter tous les obstacles et chemins possibles). Puis nous pouvons alors le configurer pour obtenir les chemins que l'on souhaite directement depuis l'interface unity.



Il est également important de placer certains composants sur le préfabriqué du joueur. Il faut impérativement d'abord mettre un **networktransform et un networkrigidbody**.

La suite se passe dans les scripts campusmanager et playercontroller.

Campusmanager sert à récupérer le composant navmeshsurface puis à le construire depuis le script.

```
NavMeshSurface s = parent.GetComponent<NavMeshSurface>();
s.BuildNavMesh();
```

On crée par la suite notre navmeshagent (qui est alors attaché à un gameobject jouant le rôle de joueur).

```
myNavMeshAgent = player.AddComponent<NavMeshAgent>();
```

Une fois la phase de paramétrage terminée, on peut alors récupérer les entrées clavier dans playercontroller afin de faire bouger notre navmeshagent.

```
// Update is called once per frame
0 references | Nicolas, 12 hours ago | 2 authors, 2 changes
void Update()
{
    if (Input.GetKey(KeyCode.UpArrow))
    {

        if (gameObject.tag == "NetworkPlayer")
        {
            Vector3 temp = myNavMeshAgent.velocity + (transform.forward * Time.deltaTime);

            defaultPlayer.Move(temp);
        }
    }
```

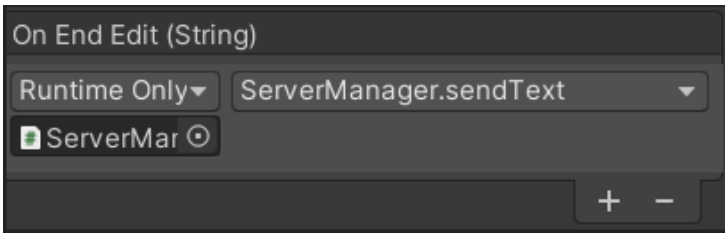
Il est alors possible de transmettre la position du joueur au serveur afin d'actualiser sa position pour tous les autres joueurs. Pour faire fonctionner efficacement le navmesh, il est probable qu'il faille désactiver une option dans les players settings de Unity -> **Batching Static doit être décoché**.

Chat

Pour pouvoir avoir une interface de chat, une succession de canvas et objet TMP (textmesh pro) ont été utilisé.

Le composant text est un l'endroit où le texte est affiché et le composant chat_input est l'endroit où l'utilisateur peut écrire son message.

Sur ce composant d'input, on paramètre aussi le déclenchement d'une fonction lors de l'appui de la touche entrée par l'utilisateur afin d'envoyer son message.



Maintenant, nous allons nous pencher sur les scripts. Tout le code du chat se situe dans les fichiers servermanager et defaultplayer.

Dans servermanager, on retrouve la fonction sendtext() ci-dessus. Cette fonction récupère le texte écrit par l'utilisateur et l'envoie au serveur. Pour ce faire, on s'assure d'abord d'un message non vide, puis on récupère les infos du client souhaitant envoyer message puis on le lui fait envoyer au serveur grâce à la méthode sendMessageServerRpc(). Il faut ensuite penser à supprimer le message tapé par l'utilisateur.

```
0 references | Marine Mangan, 44 days ago | 1 author, 1 change
public void sendText(string message)
{
    if (string.IsNullOrWhiteSpace(message)) { return; }

    var playerObject = NetworkManager.Singleton.SpawnManager.GetLocalPlayerObject();
    var player = playerObject.GetComponent<DefaultPlayer>();

    player.SendMessageServerRpc(message);
    input.text = string.Empty;
}
```

Du coté du serveur, n'ayant pas accès au type string, il nous a fallu trouver une solution… c'est pourquoi, pour le moment, nous utilisons une succesion de networkvariable<char> afin de simuler une variable String.

```
private NetworkVariable<char> mess_1 = new NetworkVariable<char>(NetworkVariableReadPermission.Everyone);
private NetworkVariable<char> mess_2 = new NetworkVariable<char>(NetworkVariableReadPermission.Everyone);
private NetworkVariable<char> mess_3 = new NetworkVariable<char>(NetworkVariableReadPermission.Everyone);
private NetworkVariable<char> mess_4 = new NetworkVariable<char>(NetworkVariableReadPermission.Everyone);
private NetworkVariable<char> mess_5 = new NetworkVariable<char>(NetworkVariableReadPermission.Everyone);
```

Lors de la réception du message par le serveur, celui-ci décompose alors par lettre le message et le renseigne dans les variables correspondantes.

```
mess_1.Value = message[0];
if(message.Length > 1) mess_2.Value = message[1];
if (message.Length > 2) mess_3.Value = message[2];
if (message.Length > 3) mess_4.Value = message[3];
```

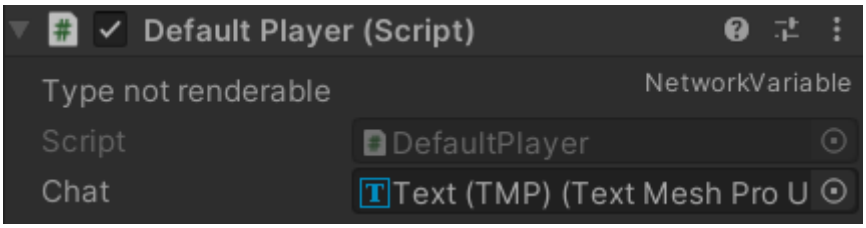
Enfin, nous avons une méthode qui s'active lors d'un changement sur l'une de ces variables. Si un changement est détecté, on ajoute alors le nouveau caractère au chat.

```
private void OnEnable()
{
    mess_1.OnValueChanged += onNewMessage;
    mess_2.OnValueChanged += onNewMessage;
```

```
private void onNewMessage(char oldMessage, char newMessage)
{
    if (!IsClient) { return; }

    Debug.Log("adding text "+newMessage);
    chat.text += newMessage;
}
```

Il est également important de bien **renseigner le gameobject** chat dans le script defaultplayer comme ceci :

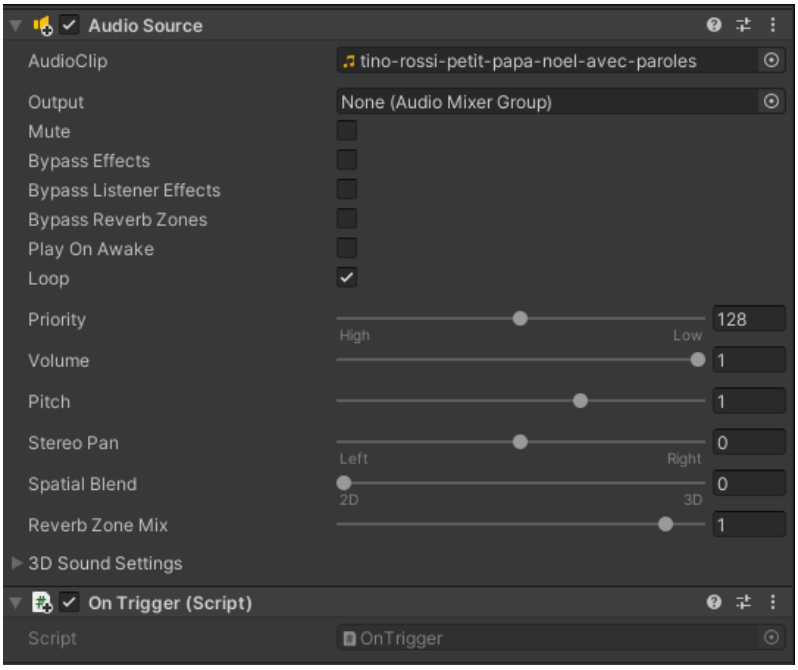


Interactions

Il y a 2 types d'interactions, le son et les vidéos.

Pour ajouter du son :

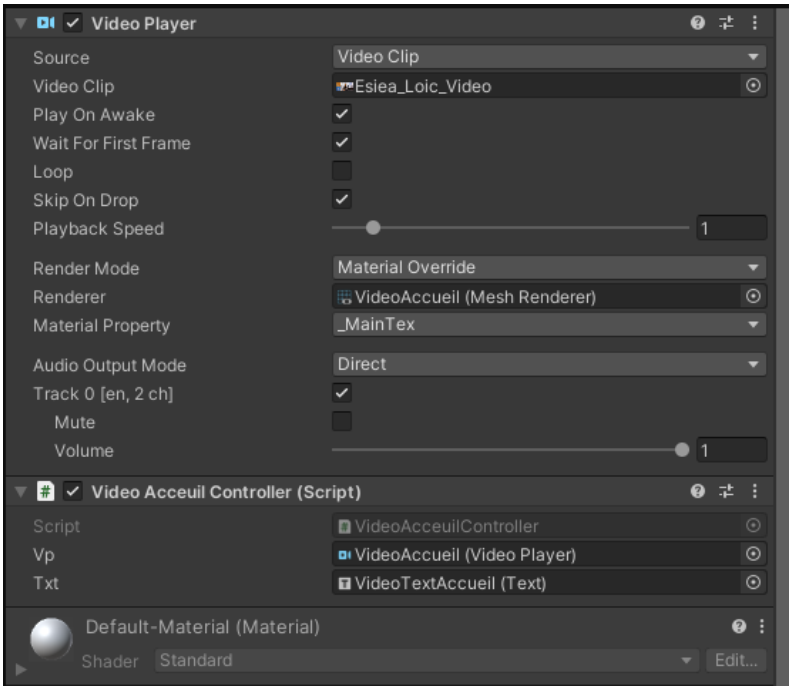
Pour ajouter du son à une scène, il suffit d'ajouter un « audio source » dans l'inspecteur d'un objet d'une scène puis d'ajouter le son que l'on souhaite importer dans la section « Audio Clip ».



Pour ajouter un son dans une pièce, il faut ajouter un « collider » dans l'espace de la scène où l'on veut ajouter du son. Il faut ensuite dans le script récupérer le son de la scène et le déclencher lorsqu'il entre en collision avec le collider.

Pour ajouter une vidéo :

Tout comme le son, l'ajout d'une vidéo se gère dans l'inspecteur. Il faut ajouter un « vidéo player » puis ajouter la vidéo souhaitée dans la section « vidéo clip ». Les différents paramètres (lecture, pause, time code, etc) peuvent se gérer dans un script.



Pour ajouter une vidéo sur un modèle 3D d'écran, il faut ajouter un quad devant l'écran. Il vous faudra redimensionner le quad à la taille de votre écran. Si votre écran n'est pas de forme rectangulaire, vous pouvez utiliser l'outil Unity « ProBuilder » (outil de modélisation 3D) pour ajouter votre quad à la forme de votre écran. Une fois que le quad est positionné vous pouvez ajouter votre vidéo sur le quad via l'inspecteur.

Bugs et améliorations possibles

Voici une liste non exhaustive de quelques améliorations auxquelles nous avons pensés :

Actuellement, le système de chat repose sur une liste de variable char puisque netcode ne possède pas encore de network variable de type string. Ce n'est pas très optimisé et en plus ne permet d'envoyer que des phrases de 20 caractères maximum, il faudrait donc remplacer dès que possible ce système avec quelque chose de plus robuste.

Il serait bien et plus confortable pour les joueurs de voir le nom des autres au-dessus afin de pouvoir se repérer plus efficacement. Il serait également plus simple pour les visiteurs de savoir qui sont les guides avec un indicateur visuel sur leur modèle ainsi que dans le chat.

Actuellement, lors du déclenchement d'un son par un joueur, celui-ci se joue pour tout le monde. Logiquement, il faudrait que celui-ci ne soit écoutable que par le joueur l'ayant déclenché.

Il n'y a pas eu d'aspect sécurité sur le projet jusqu'à maintenant. Il faut donc faire un point sur tout cet aspect.

Il faudrait avoir un système de login et de sauvegarde des données poussé. Ainsi il serait possible pour les guides et professeur de se connecter facilement depuis leur compte esiea et pour les visiteurs de garder leurs données pour les recontacter facilement par la suite.

Malheureusement, nous avons également rencontrés quelques bugs qui n'ont pas encore pu être résolu voici ceux répertorié à ce jour :

Lorsqu'un joueur se déplace dans l'école certains modèles disparaissent.

La vitesse de déplacement du joueur est lente. Il semblerait que changer sa vélocité n'affecte pas le joueur et qu'il soit bloqué à la vitesse de rafraîchissement des positions du serveur.

En mode host, les déplacements ne sont pas pertinents.