Documentation technique

IRIT Lab VR

Développé par Lucas Borja dans le cadre de son stage du 31/03/2025 au 16/07/2025

Encadré par Olivier Négro

Sommaire

Sommaire 2
Introduction 3
1-Manuel d'utilisation 4
1.1-Prise en main à la manette 4
1.2-Fonctionnalités de la Tablette Virtuel 5
1.3-Lecture des données affichés dans l'espace 3D 9
1.4-Installation et mise en route du logiciel 11
2-Processus de création 11
2.1-Création du modèle 3D 12
2.2-Mise en place d'une scène VR dans Unity 14
2.3-MQTT dans Unity 20
3-Fonctionnalités du logiciel 21
3.1-Contrôle du Rail 21
3.2-Affichage des positions réelles et estimées des nodes 23
3.3-État ouvert et fermé des portes 24
3.4-Représentation des mesures de Ranging et des erreurs associées 25

Introduction

IRIT Lab VR est une application en VR (Virtual Reality) qui permet de visualiser les résultats des expérience faite sur la plateforme LocURa4IoT (Localisation and UWB-Based Ranging testbed for the Internet of Things) de l'IRIT (Institut de Recherche en Informatique de Toulouse). Elle permet plus précisément d'afficher les estimation de position et les mesure de ranging en récupérant les informations publiées sur le réseau MQTT de la plateforme. On est aussi capable de publié sur le réseau afin de piloter le rail. L'application a été développée sur le moteur de jeu Unity et est faite pour tourner sur les casques MetaQuest. Ce sont des casque Stand Alone qui ne nécessitent donc pas d'un ordinateur pour fonctionner, ainsi l'expérience est complètement portable.

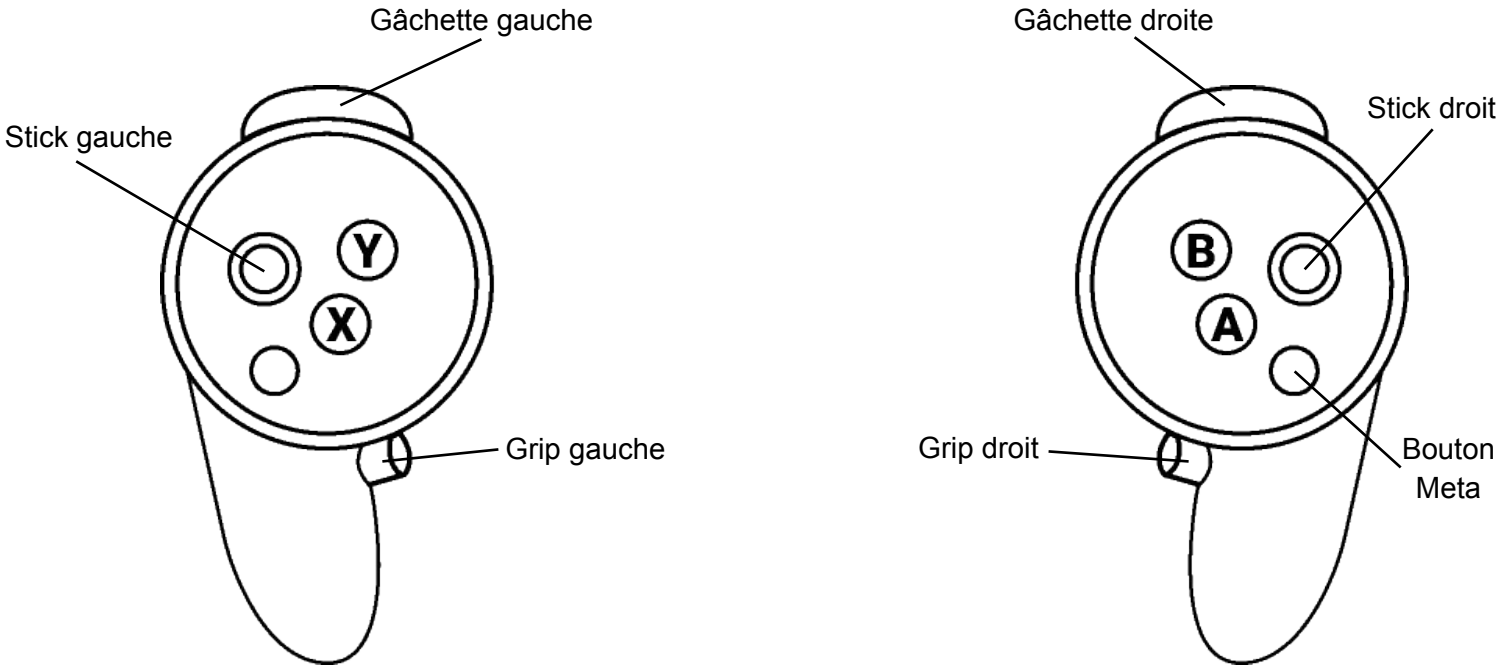
Cette application à été réalisée dans le cadre de mon stage de deuxième année au sein du Master cinéma XR (eXtended Reality) de l'ENSAV (École Nationale Supérieure d'AudioVisuel) qui s'est déroulé du 31 mars 2025 au 16 juillet 2025. Dans cette documentation technique je vais, dans un premier temps, dresser un manuel d'utilisation complet de l'application, je reviendrait ensuite sur les étapes de création du projet pour enfin aborder le fonctionnement des différentes fonctionnalité et les éventuelles possibilité de modification.

1-Manuel d’utilisation

Dans cette première partie on va voir en détail comment utiliser le logiciel, à savoir comment interagir avec via les manettes, quelles sont les différentes fonctionnalités de la tablette virtuelles, comment lire les données affichées dans la scène 3D et enfin l’installation et le lancement du logiciel.

1.1-Prise en main à la manette

Le logiciel ayant été fait à partir du template XR d’Unity, les interactions possibles sont les interactions de base de tout logiciel en VR. Je vais ici les rappeler et expliquer ce qu’elle permettent concrètement dans le cas présent.



Manette MetaQuest 3

Le déplacement :

Les deux méthodes de déplacement dans l’espace sont les suivante :

- Le déplacement « fluide » avec le **stick gauche**. Avec ce mode on se déplace de façon fluide dans la direction dans laquelle on pousse le stick comme dans un jeu vidéo classique. Ce mode est intuitif mais selon la sensibilité de votre oreille interne il peut vous donner une nausée similaire au mal des transport, c’est pourquoi il existe un deuxième mode de déplacement.
- La téléportation avec le **stick droit**. En poussant le stick droit vers l’**avant** vous verrez apparaître une courbe en cloche et un cercle à l’endroit où elle touche le sol. En relâchant le stick vous serez téléporté là on se trouvait ce cercle. Ce déplacement plus saccadé permet de ne plus souffrir de nausée.

La Vue :

Pour regarder autour de soit on peut tout simplement tourner la tête mais si on souhaite rester dans une position fixe on peut aussi changer l'orientation de la vue en poussant le **stick droit** vers la **droite** ou vers la **gauche**.

Attraper des objets :

Les deux **grips** (droite et gauche) permette d'attraper un objet quand on les maintient enfoncé. Cela permet notamment d'attraper la tablette virtuelle ou les poignées des porte qui ne sont pas pilotée pas un capteur pour les ouvrir. Attention relâcher le grip signifie que vous lâchez l'objet attrapé dans la VR.

Interaction avec le menu :

Pour interagir avec tout type d'interface (que ce soit les boutons de la tablette ou le menu du MetaQuest lui même) il suffi de viser le bouton avec lequel on veut interagir et de cliquer sur la **gâchette** de la manette avec laquelle on vise.

Afficher le menu MetaQuest en jeu :

Il suffit d'appuyer sur le **bouton Meta** sur la manette droite. Ce menu vous permettra de quitter le jeux ou d'accéder aux paramètres du casque.

Sauter :

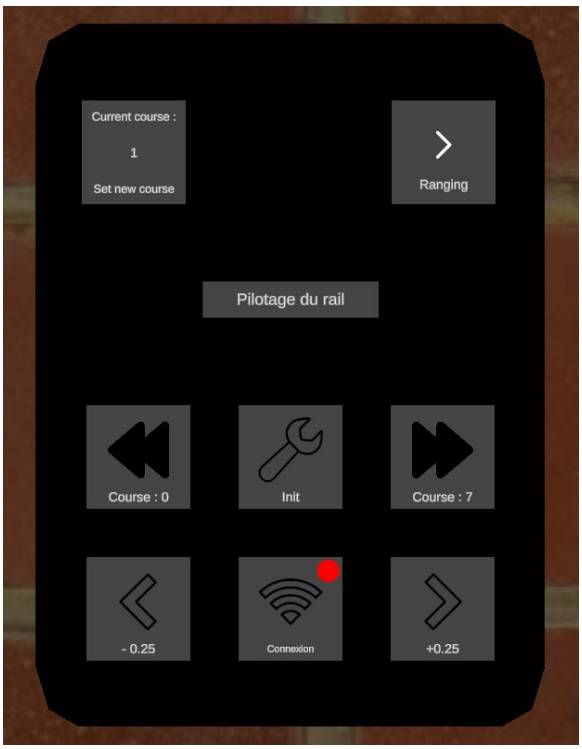
Il est possible de sauter en appuyant sur le bouton **A**.

1.2-Fonctionnalités de la Tablette Virtuel

Dans la scène une tablette virtuelle est présente, elle a été construite sur le modèle de la tablette qui permet de piloter le rail de sept mètres de la plateforme LocURa4IoT mais elle c'est vu dotée d'option en plus spécifique aux fonctionnalités de ce logiciel. Je vais ici en détailler les différents menus.

Menu contrôle de rail :

Le menu de contrôle du rail se présente avec l'interface suivante :



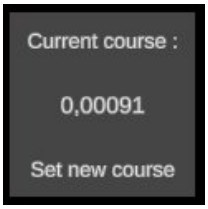
Ce premier menu permet de piloter le rail de sept mètre de la plateforme en publiant des requêtes MQTT.

En bas au centre on a l'indicateur de **connexion**. Celui-ci est rouge lorsque l'on est pas connecté au réseau MQTT et devient vert dans le cas contraire.

La position du chariot du rail est comprise entre 0 et 7. Les boutons **Coures : 0** et **Course : 7** permettent de déplacer le rail respectivement à l'une de ses positions extrêmes. Les boutons **-0.25** et **+0.25** permettent respectivement de déplacer le chariot de 25cm vers le point 0 et de 25cm vers le point 7.

Le Bouton **Init** réinitialise complètement le rail. Il faudra le faire bouger à nouveau pour récupérer sa position en VR.

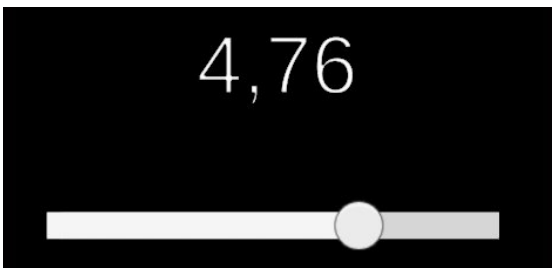
En haut à gauche on à l'indicateur de la position de actuelle du chariot sur le rail, il se met à jour dès que le rail bouge. Si on clic dessus on à accès à un sous menu du contrôle du rail.



Menu contrôle de la position du chariot :

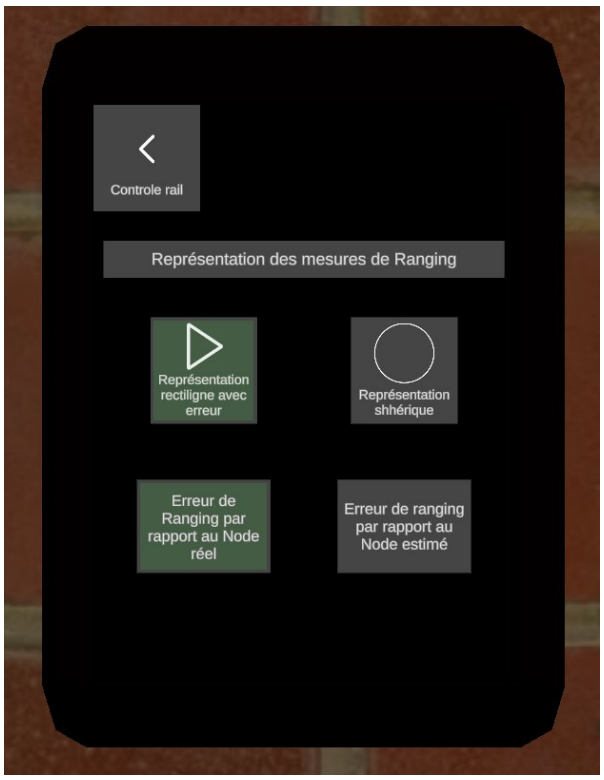
Le menu de contrôle de la position du chariot sur le rail ce présente comme ceci :

En faisant glisser le curseur on peut sélectionner une valeur de position comprise entre 0 et 7, il suffit ensuite de cliquer sur le bouton **Set Course** et le chariot se déplacera à la position indiquée. On peut revenir au menu de contrôle du rail avec le bouton **Retour**.



Menu des mesures de Ranging :

En appuyant sur le bouton en haut à droite dans le menu de contrôle du rail on accède à ce menu qui nous permet de choisir comment représenter les mesure de ranging.



Ce menu nous donne la possibilité de représenter les mesure de Ranging de deux manières différentes, avec des segments ou avec des sphères. Les options d'affichages actuellement sélectionnées sont surlignées en vert.

Représentation rectiligne avec affichage des erreurs :

La première option nous permet de représenter les mesures de ranging avec un segment suivant la droite reliant le node détectée à l'ancre qui le détecte. On trace donc en vert un segment partant de l'ancre et d'une longueur égale au ranging mesurée et on trace en rouge un segment allant de l'extrémité de ce segment jusqu'au node détecté.

Les deux boutons su bas servent à choisir si l'on veut représenter l'erreur de ranging par rapport à la position réel du node mesuré ou par rapport à sa position estimé. Dans le premier cas, les segments sont donc tracés selon la droite reliant la position de l'ancre et la position réelle du node et dans le second selon la droite reliant l'ancre et la position estimée.

Représentation sphérique du ranging :

Lorsqu'une ancre détecte le node recherché ce qu'elle obtient comme information c'est la distance à laquelle ce trouve ce node, mais pas sa direction sans information supplémentaire. On à donc le choix ici de représenté les mesure de ranging par des sphères qui sont santré sur l'ancre qui a réalisée cette mesure et dont le rayon a une longueur de la valeurs du ranging mesuré.

1.3-Lecture des données affichés dans l'espace 3D

Je vais ici détaillé le code couleur de ma scène Unity afin de permettre une lecture correcte des données qui y sont représenté

Représentation des Nodes :

Les **cubes gris** semi transparents représentent **la position réelle** d'un node qui n'est **pas connecté** au broker MQTT (pour l'expérience en cours)

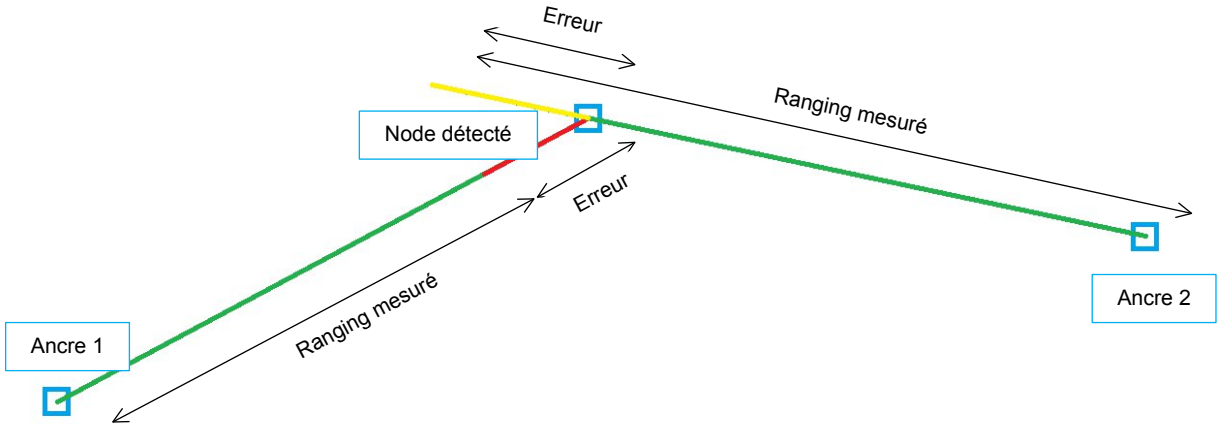
Les **cubes bleu** clair représentent la **position réelle** d'un node qui est **connecté** au broker (pour l'expérience en cours).

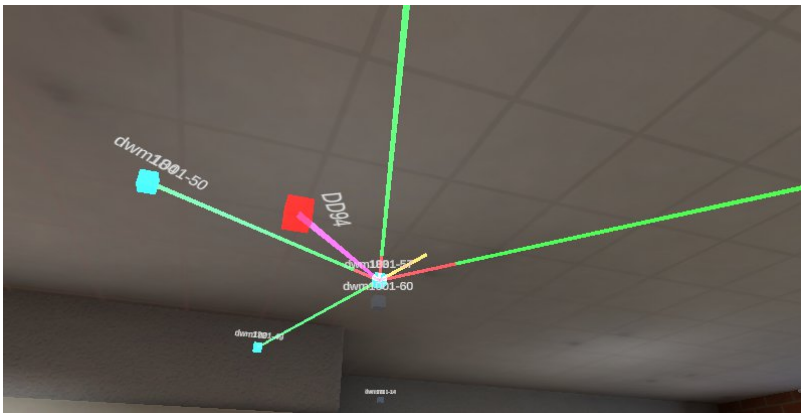
Les **cubes rouges** représentent les **positions estimées** des nodes dont on cherche à mesurer la position. Si ce node a une position réelle connue enregistrée sur le broker MQTT alors le cube rouge est relié par un **ligne violette** au cube bleu correspondant au node réel.

Représentation des mesures de ranging :

Les lignes **vertes** représentent le **ranging** et les lignes **rouges** représentent **l'erreur de ranging**, mais il y a deux cas possibles.

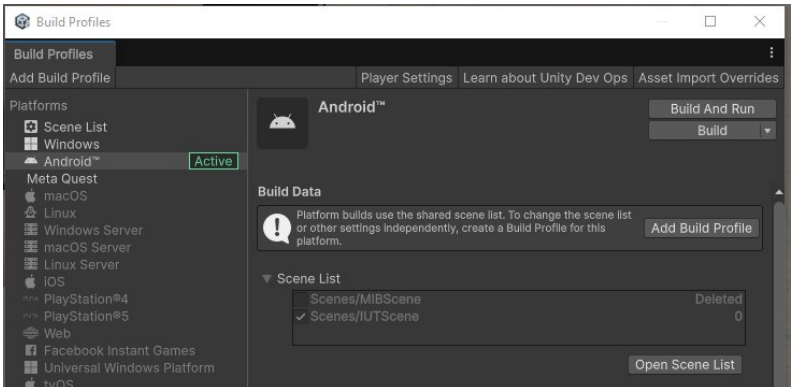
Dans le cas d'une **sous estimation** du ranging, le segment vert part du node ancre et s'arrête avant le node détecté et l'erreur apparaîtra rouge (le segment rouge étant tracé de l'extrémité du vert à la position du node mesuré). Dans le cas d'une **sur estimation** le segment vert dépassera le node détecté et le segment rouge recouvrira la partie du segment vert qui dépasse le node détecté, de fait l'erreur apparaîtra jaune avec le mélanges des deux lignes.





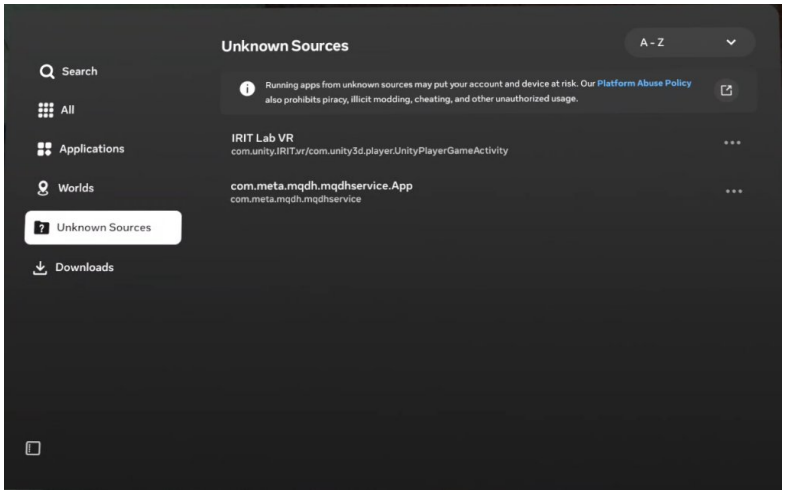
1.4-Installation et mise en route du logiciel

Le logiciel se présente sous la forme d'un .apk, pour le buid depuis le projet Unity il suffi daller dans l'onglet File -> Build Profile puis de cliquer sur Build (ATTENTION : il faut bien que la platefome active pour le build soit Android)



Ensuite il suffit de l'installer dans le casque. Pour cela on branche le casque au PC et dans l'application Meta Quest Developer Ub (Il faut utiliser un casque préalablement passé en mode développeur pour cela), dans l'onglet Device Manager on peut cliquer sur Add Build et ajouter notre apk.

Dans le Casque MetaQuest 3 on peut rettrover le logiciel dans la librairie rangé dans le dossier Unknown Sources.



2-Processus de création

Dans cette partie je vais aborder les différentes étape de la création du projet durant pas période de stage en commençant par la création du modèle 3D, puis la mise en place de la scène Unity et enfin l'implémentation des fonctions MQTT dans cette dernière.

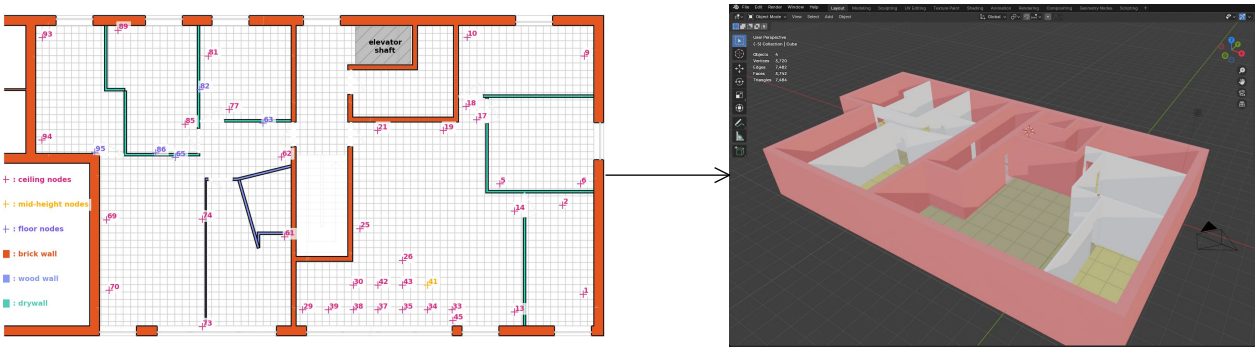
2.1-Création du modèle 3D

La création du modèle 3D de la plateforme LocURa4IoT c'est fait en trois étapes, la modélisation, la création des UV et le texturing. Ces trois étapes ont été principalement réalisées sur Blender, un logiciel du 3D libre et gratuit.

Modélisation 3D :

Au début la question s'est posée d'utiliser ou non un scanne 3D, avec les test qui avait été effectués on en a conclu que cette solution n'était pas adaptée à la situation. Le modèle 3D obtenu à partir du nuage de point qui nous ai donné par le scanne est certes très précis en dimension mais il est aussi assez grossier sur sa géométrie et contient un nombre de polygones bien trop grand. De plus c'est un modèle en un seul bloc et pas divisé en plusieurs partie (pour pouvoir séparer les parties fixes des parties interactives.

On a donc opté pour la solution de modéliser à parti d'un plan. On part d'un cube auquel on donne la forme du sol, on dessine le tracé des murs avec des subdivisions et on extrude ces tracés pour faire sortir les murs du sol. Ce faisant on a déjà la forme globale du mesh 3D. Ensuite il a fallu creuser les trous pour accueillir les portes et les fenêtres, ajouter les détails tel que les plaintes ou encore les escaliers, etc. Les portes ont été modélisé avec un add on de Blender appelé Archimesh. Suite à cela j'ai séparé les différents éléments de mon modèle (sol, mur, plafond...) et j'ai procédé à un un « nettoyage » de la géométrie des modèles pour qu'il ait le moins de polygones possible pour ne pas prendre trop de performance une fois importés dans Unity.

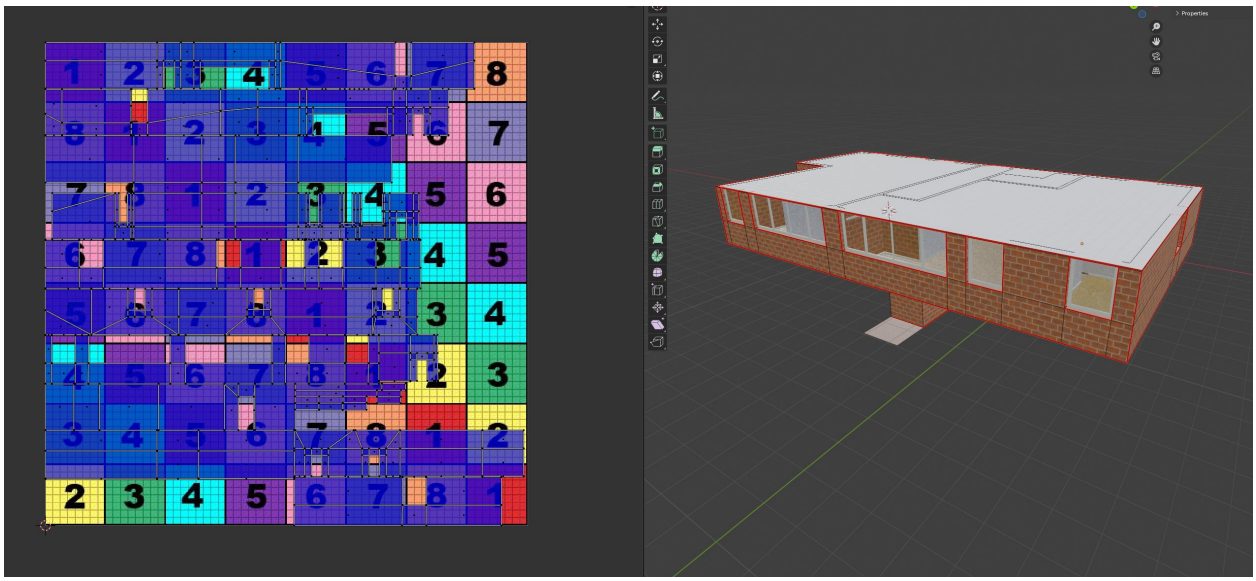


ATTENTION : Une fois la modélisation terminer ne pas oublier de vérifier si la scale est normalisée, si les valeurs de scale en x, y et z ne sont pas à 1 il faut selectionner l'onbjet et faire ctrl A puis clique sur scale. Cela permet d'appliquer les modifications de taille qu'on a réalisé sur notre mesh 3D afin de ne pas avoir de problème plus tard avec les UV ou les lightmaps.

Création des UV :

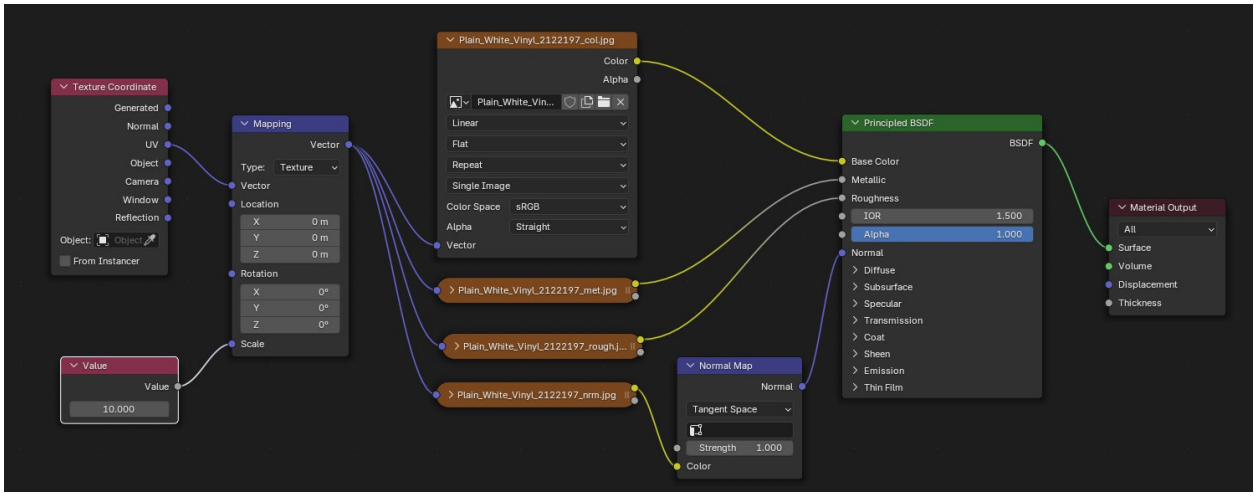
Ensuite pour chaque modèle 3D il a fallu déplier les UV. Les UV sont l'équivalent d'un patron pour un cube, c'est le modèle 3D déplier à plat. Il faut donc dans l'onglet UV

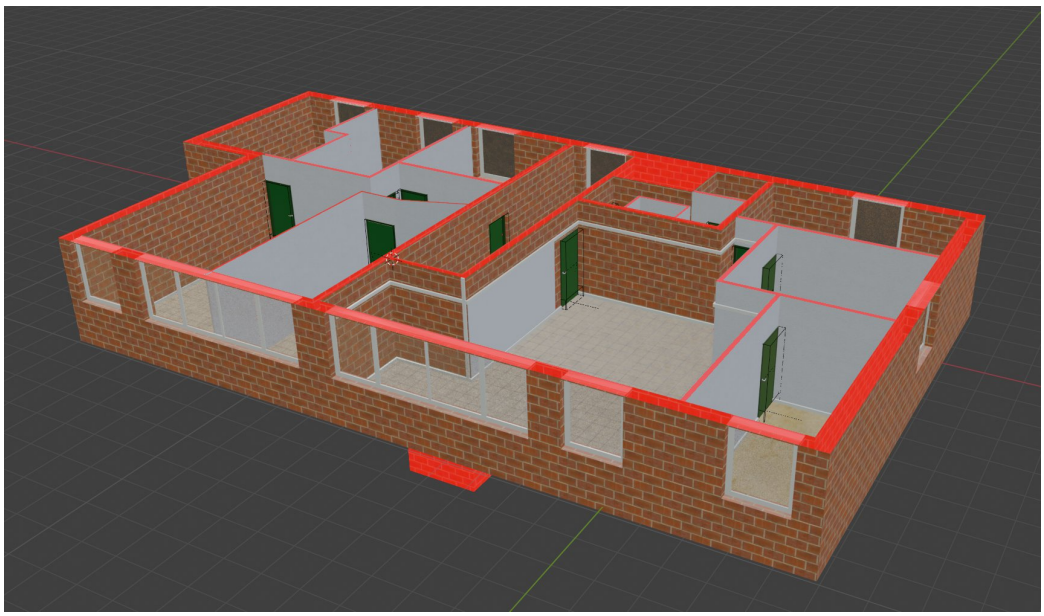
editing de Blender définir les coutures du modèle (les edge où on va couper). On fait cela jusqu'à ce que plus aucune face ne soit déformer après dépliage. Cette étape est importante car les UV servent à appliquer correctement les textures aux modèles 3D. Cela servira aussi pour Unity car les lightmaps se servent des UV.



Shading et textures :

Enfin, on prépare les textures dans l'onglet Shading. Toutes les textures utilisée sont libre et viennent de l'add on Blenderkit. Les textures récupérée ne se contente pas de définir la couleur des éléments, ils en définissent aussi des paramètres comme la métallicité, la rugosité ou les détaille fin du relief (comme sur un mur en brique par exemple). Chacune des ses information son stocké dans un fichier texture (dont la résolution doit être de 1*1 pour Unity)



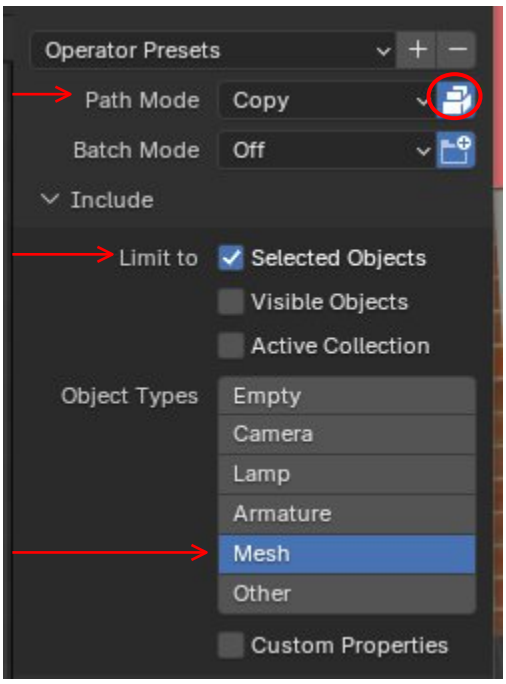


2.2-Mise en place d’une scène VR dans Unity

Le projet Unity à été créé à partir du template VR disponible sur Unity Hub, ce template permet d’avoir accès à toutes les fonction de base de la VR pour notre scène (prise en charge des manette, de la détection de mouvement du casque, des interactions spécifiques à la VR, etc.). Les fonctions de base étant déjà mise en place il reste cependant quelque étape de mise en place avant de passer à l’intégration des fonctions MQTT.

Import du modèle 3D dans Unity :

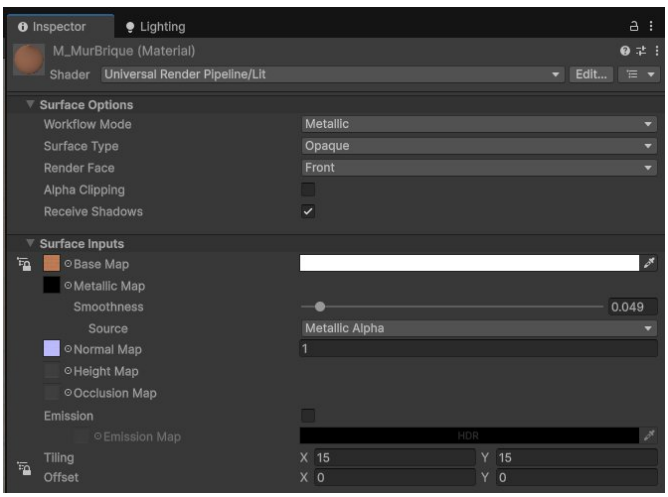
Dans Blender on exporte le modèles 3D au format FBX, attention il y a quelques réglage à faire. Dans un premier temps pour le Path Mode il faut choisir Copy et cliquer sur l’icône à droite, cela permet de récupérer les matériaux associé aux meshs 3D dans le fichier FBX. Ensuite il est recommender de ne sélectionner que les éléments de la scène que l’on veut exporter et de cocher l’option Limit to Selected Object, aussi pour être sûr de ne pas exporter une lumière ou une caméra on peut sélectionner



Dans le projet Unity il suffit ensuite de Import new asset, une fois importer on peut le glisser dans la scène. Après ça, ce n'est pas finit car il reste quelques réglage à faire.

Matériaux dans Unity :

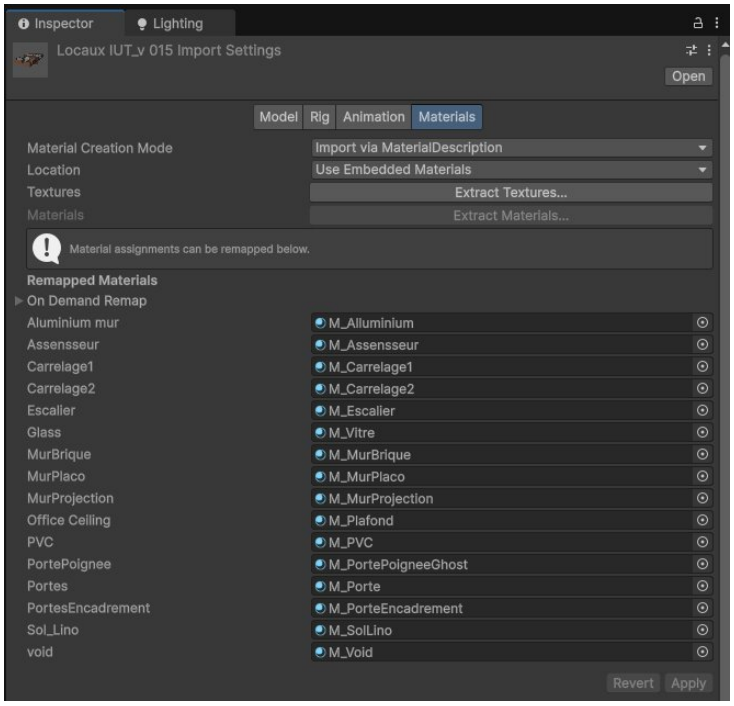
Les moteur d'Unity n'est pas compatible avec le shader graph de Blender. Il faut donc refaire les matériaux des objet dans Unity à partir des texture que l'on a utilisé. On importe donc les textures (base color, Normal map, etc.) dans le projet et on cré un matériaux dans lequel on inclura les textures ainsi que le paramètre de scale relatif au UV (ici Tiling).



ATTENTION : Le paramètre de roughness (rugosité) n'est pas pris en charge de la même manière que les autres logiciels de 3D dans Unity. Premièrement ici c'est un paramètre de Smoothness (soit l'inverse de la roughness) de plus contrairement aux autre texture on ne peut pas lire directement une map de smoothness dans un matériau d'Unity, cette map doit se trouver dans la couche alpha de la map de metalic.

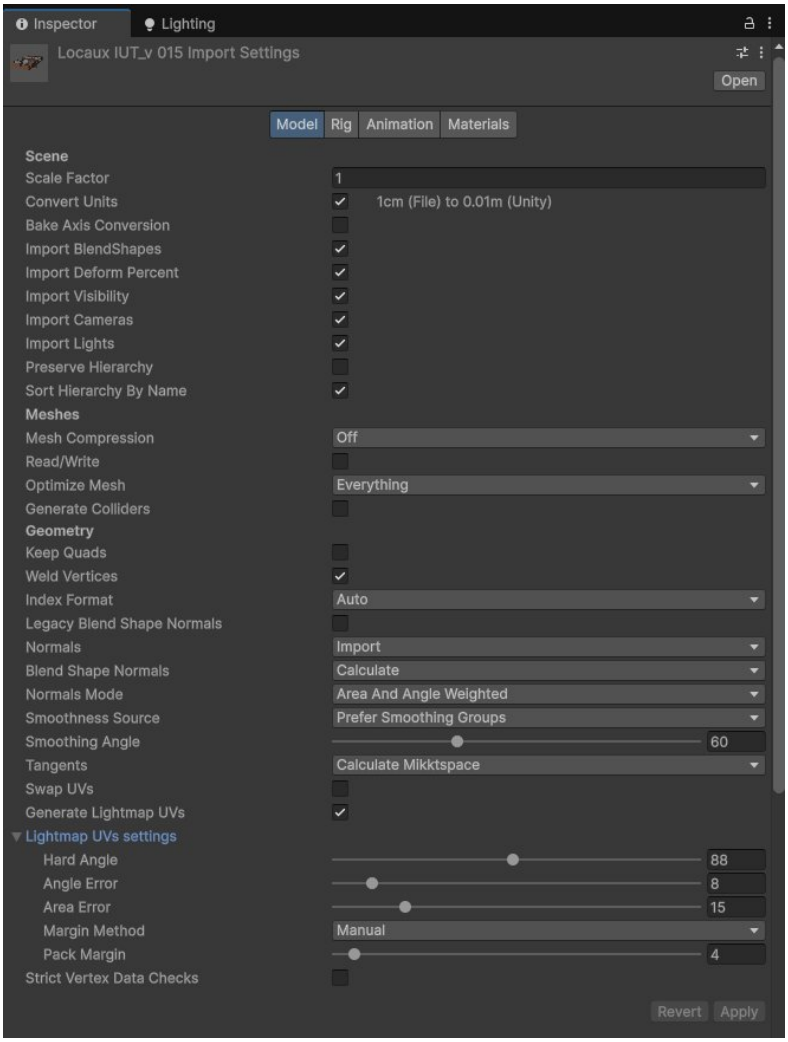
Pour répondre à ce problème j'ai utilisé GIMP, un équivalent gratuit et libre de Photoshop grâce auquel j'ai pu inverser les map de Roughness et les mettre en alpha des map de metalic. (à noté qu'en l'absence de map de metalic je l'ai créé moi même en faisant une texture entièrement blanche si l'objet est en métal ou noir sinon). De plus GIMP m'a aussi permis de changer la couleur de certaines textures pour mieux se rapprocher de la réalité (chose possible dans le shader graph de Blender mais pas dans Unity)

Une fois ces matériaux préparer il faut remplacer les matériaux dans l'onglet Material de l'inspector, ainsi les nouveau matériaux seront appliqué automatiquement au modèle 3D à la place des ancien (d'où l'importance d'exporter le FBX avec les matériaux).

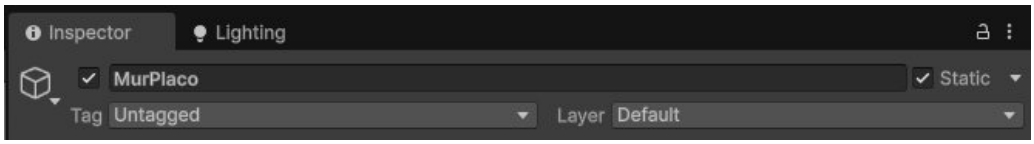


Lumières dans Unity :

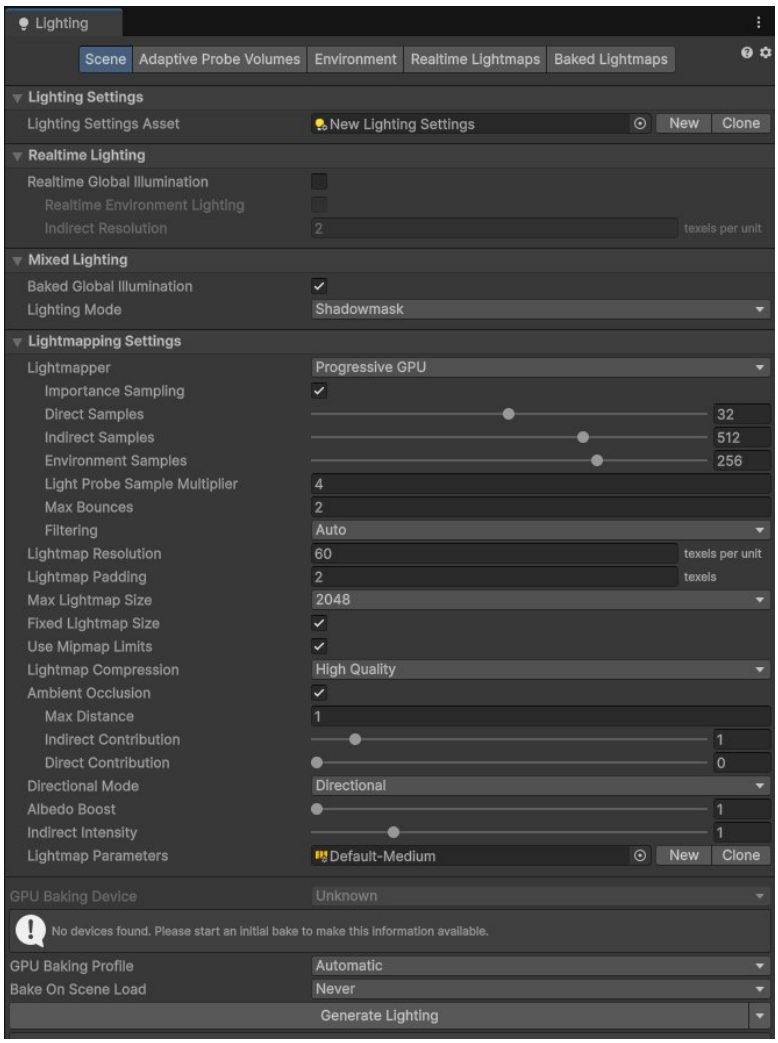
Pour économiser des ressource pour le logiciel on va pré-calculer la lumière ambiante, ainsi elle ne sera pas calculée en temps réel. Pour cela on va procéder à un baking qui va créer, pour chaque objet 3D, des lightmaps basées sur ses UV. Ces lightmaps vont venir appliquer les effet d'ombre et de lumières sur les textures de l'objet. Pour utilisé ces lightmaps il faut aller dans l'inspector du modèle importé et dans l'onglet Modèle coché l'option Generate Lightmap UVs et paramétrer la Margin Methode sur Manual (pour bien utiliser les UV qu'on a déplié dans Blender).



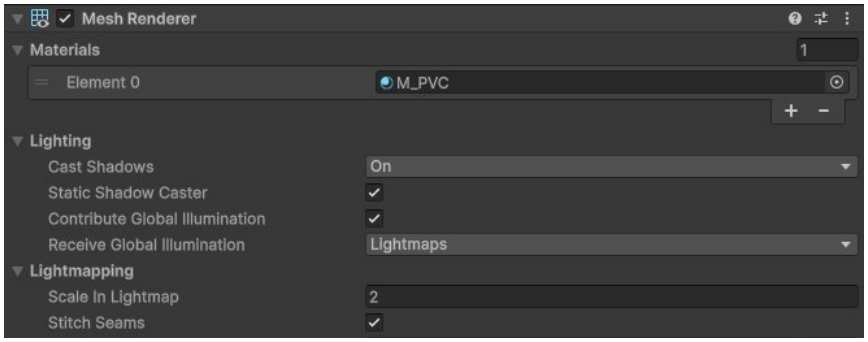
Pour utiliser les lightmaps il faut définir chaque objet qui n'est pas voué à bouger dans la scène en Static dans l'inspector.



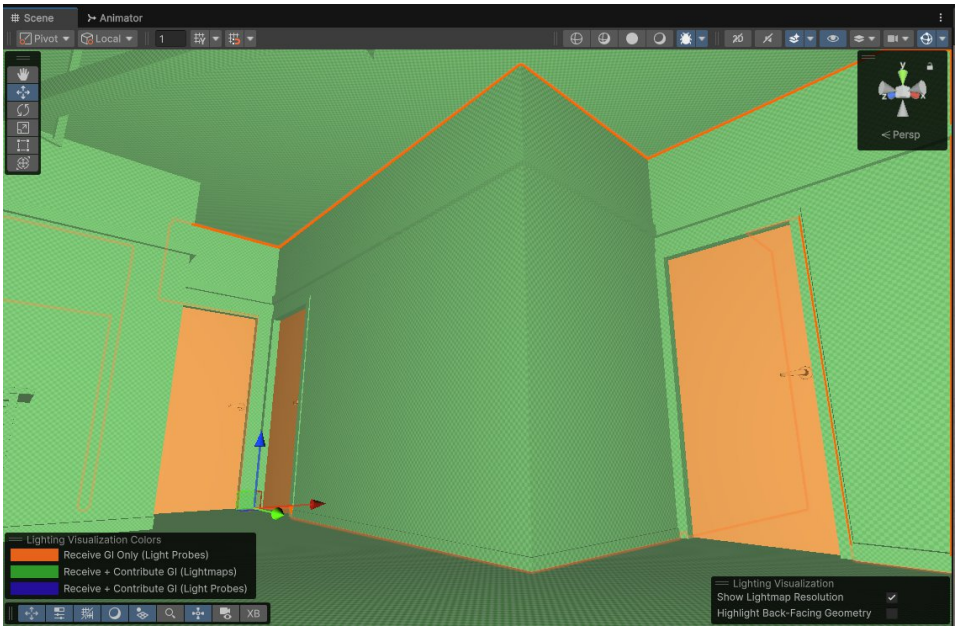
Ensuite c'est dans la fenêtre Lighting que l'on procède au Baking. On peut en régler certains paramètre comme la qualité (plus la qualité est grande et plus long sera le rendu) mais le plus important est de bien choisir Progressive GPU pour le Lightmaper, sinon il utilisera le CPU.



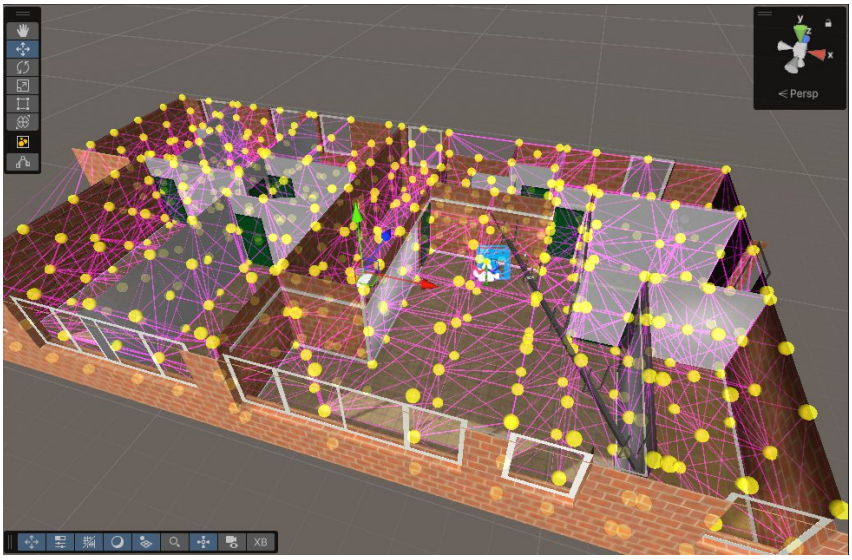
Si la résolution des ombres et lumière est trop faibles sur un modèle il est possible d'augmenter la taille des light map dans l'onglet mesh renderer dans l'inspector du modèle.



On peut se rendre compte de la taille des lightmaps en débug mode, l'idéal est que les carré qui compose les damier vert soit à peut près de la même taille d'un modèle à l'autre et le plus petit possible



Ensuite pour gérer la lumière sur les objet non statique on peut installer des lightprobs le long de chaque surface (mur, plafond, sol)



Enfin, on définit le volume du reflexion probes pour qu'il englobe tout l'espace 3D, ceci nous permet de calculer correctement les effet de réflexion de la lumière.

Objet interactif dans Unity :

Certains objets de la scène sont interactifs, par exemple pour la tablette qui est un objet que l'on peut attraper il suffit le lui appliquer la fonction XR Grab Intercatable du template XR d'Unity et de le paramétrer comme on le souhaite pour.

Concernant les 3 portes qui n'ont pas de capteur de mouvement il est possible de les ouvrir manuellement dans la scène. Pour cela elles ont aussi un XR Grab Interactable mais le point d'attache se trouve sur les pognées et j'ai rajouté un hinge joint au niveau du point de pivot de la porte pour que le seul degré de liberté de cette dernière soit une rotation d'un certain angle.

2.3-MQTT dans Unity

Pour pouvoir utiliser MQTT dans Unity il faut importer le package MQTTnet à l'aide de l'add-on NuGet qui nous permet d'importer facilement des package C# comme MQTTnet. Cependant il est important de noter qu'avec la version d'Unity utilisé pour ce projet (Unity 6) les version de MQTTnet à partir de la 5.0 sont incompatible, nous avons donc utilisé une version antérieure. Ceci ne pose aucun problème pour communiquer avec le broker de la plateforme. Une fois ce package importé, on peut utiliser les librairies C# MQTTnet, MQTTnet.Client et MQTTnet.Server.

A partir de ça on crée deux scripte qui vont piloter la communication entre Unity et MQTT, un sobrement appelé MQTT et l'autre appelé TopicSubscribeHanndeling. Le premier scripte va créer un client MQTT dans unity puis va définir les 4 fonction de base de MQTT : la connection, la déconnection, l'abonnement à un topic et la publication de message (à noter qu'il y a en réalité 2 fonction de connection en fonction de si on se connecte à un réseau avec un cryptage TLS)

Le scripte TopicSubscribeHandeling a deux fonction. La première est d'initialiser la connexion en déterminant les mot de passe, login, si le réseau est encrypté ou non, etc. (à noté que si l'on ne rempli rien dans l'interface de connexion on se connecte

automatiquement à l'adresse iop 7.13 qui n'est accécible que pour le réseau local du labo. Cette fonction d'initialisation permet aussi de s'abonner à tout les topic qui nous sont nécessaire qui se troucverons dans la liste de SubscribeHandeling définit dans ce scripte et qu'on peut remplir depui l'inspector d'Unity.

Sa deuxième fonction consiste à prendre en charcge chaque message reçu en recherchant le topic auxquel il est abonnée dans la liste des SubscibeHandeling et en réalisant la ou les actions associées à ce topic.

3-Fonctionnalités du logiciel

Dans cette dernière partie nous allons détailler les différente fonctionnalités de l'application à commencer par le contrôle du rail, l'affichage des positions des nodes, la récupération de l'état des portes et enfin la représentation des mesures de Ranging

3.1-Contrôle du Rail

On s'abonne au topic *rail/1/course/indication* via la liste subscibe handeling pour pouvoir récupérer en temps réel l'information sur la course (position) du chariot sur le rail. Pour pouvoir interpréter cette information il faut pouvoir interpréter le json envoyé sur le topic. Pour cela on crée dans un scripte C# associé une Public Class qui se contentera d'être une liste de paramètres qui ont le même nom et le même type (chaîne de caractères, nombre entier, nombre flottant, etc.). Par la suite on pourra, dans une fonction C# appelée à chaque fois qu'on publie sur le topic, créer et remplir un objet de cette classe avec les information du json grâce à la fonction jsonUtility.

ATTENTION : La Public Class doit être créée EN DEHORS de la Public Class Mono Behaviour dans laquelle on exécute tous les scripte C# utilisé dans Unity.

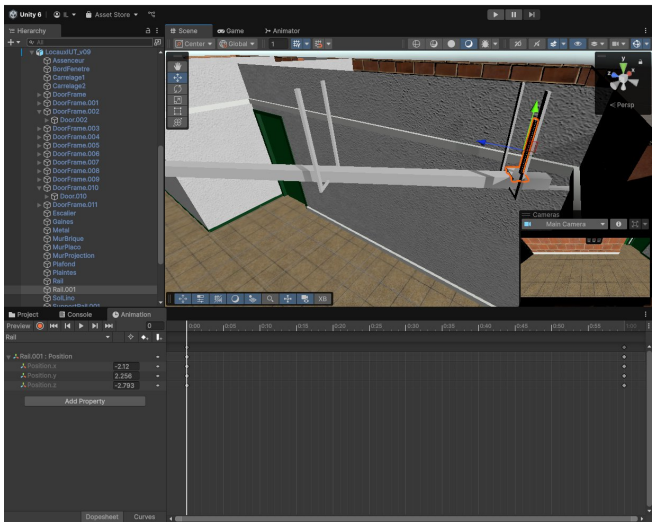
Indication de la course :

Comme première application de cette récupération de donnée dans Unity, on veut afficher l'information de course sur la tablette virtuelle (comme pour la vraie tablette). On va donc créer un scripte C# que l'on va associer à la tablette et on va y créer la Public Class que l'on nome « CouseIndication ». Ensuite dans le MonoBehaviour on va créer une fonction HandleUpdate qui devra être appelée à chaque fois que l'on reçoit un message sur ce topic. Pour cela on va, dans la liste Subscibe Handeling, associer au topic *rail/1/course/indication* l'objet 3D de la tablette et d'appeler la fonction HandleUpdate.

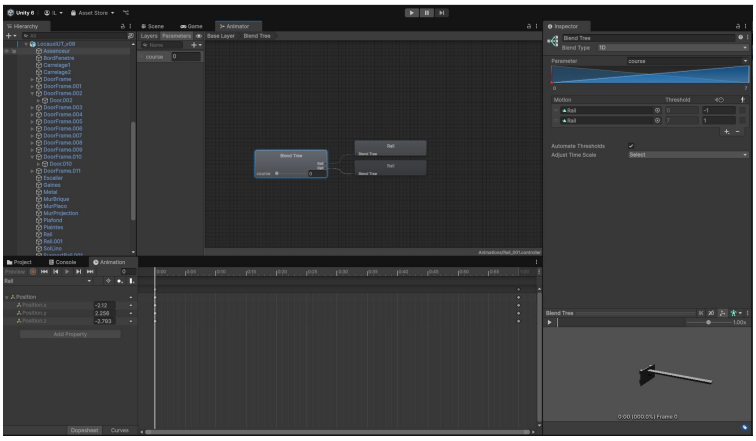
Lorsqu'elle est appelée, cette fonction remplie un objet de type CourseIndication avec les informations du json. De cette objet on va extraire l'information numérique de course et la convertir en chaîne de caractère. En sorti de la fonction on retourne cette chaîne de caractère. En associant cette fonction à un bouton du panel de la tablette on peut y afficher l'information de course.

Équivalence entre rail réel et virtuel :

Le but ici est que lorsque le chariot se déplace sur le rail dans la réalité il se déplace aussi dans l'espace en VR. Pour cela on va utiliser un scripte assez similaire au scripte d'indication de course mais avant cela il faut paramétrer une animation pour le chariot. Dans l'onglet animation on cré 2 pose clés pour le chariot, une aux coordonnée de départ de la course et une aux coordonnées d'arrivée (attention le repère 3D d'Unity ne suit pas la règle de la main droite, donc la coordonnée x devient –x et la cordonnée y devient –z)



Dans l'animation editor on crée un nouveau BlendTree qu'on connecte à l'entrée, ensuite on double clic dessus pour régler les paramètre du BlendTree. On crée un nouveau paramètre qu'on appel « course » et on fait varier sa range de 0 à 7 (pour correspondre à la range de la course contrôlé par la tablette réelle qui va aussi de 0 à 7). Ensuite pour 0 on applique l'animation Rail avec un facteur -1 et pour 7 on applique rail avec un facteur 1. Ainsi, via un scripte C#, on pourra faire déplacer cette plateforme à n'importe quelle position sur la droite entre les position de départ et d'arrivée.



On cré ensuite un scripte C# RailCourseUpdate que l'on associe au modèle 3D du chariot pour caler sa position sur la position réelle du rail. Ce scripte est très ressemblant au scripte RailCourseIndication car il est aussi constitué d'une fonction HandleUpdate qui cré un objet de type CourseIndication et le remplie avec le json du topic *rail/1/course/indication* (on peut notter que l'on a pas besoin de redéclarer dans ce scripte la PublicClass CourseIndication car comme cette classe est publique elle est accessible

par n'importe quel scripte de la scène ou elle a été déclarée) . Pour que cette fonction soit appelé à chaque message reçu sur le topic il faudra aussi rajouter à ce dernier dans la liste des SubscribeHandeling l'objet 3D de la plateforme et y appeler la fonction. La différence avec le scripte précédent c'est que cette fois on prend en paramètre le composant animator de l'objet 3D et qu'on ne converti pas l'info de course en tecte mais qu'on récupère le paramètre course de l'animator et qu'on définit sa valeur sur celle du paramètre du même nom récupéré dans le json. Ainsi à chaque fois que l'on recevra une information de course le chariot se déplacera à la position qui y correspond (par exemple si l'information de course reçue est 4,5 le chariot se déplacera à la position 4,5/7).

Pilotage du Rail :

Pour terminer afin de simuler la tablette de pilotage du rail il nous faut pouvoir le contrôler depuis la VR.

3.2-Affichage des positions réelles et estimées des nodes

Le but de cette fonctionnalité là est de récupérer de façon dynamique les informations de position des nodes publiées sur le broker MQTT et de modifier la scène en conséquence. Ces information se trouves dans des topic dont le chemin prend la forme **localisation/{nom du node}/{type du node}**. Étant donné le nombre de nodes de la plateforme et le fait que l'on veut que le logiciel prenne automatiquement en compte les évolutions de la plateforme (ajout ou retrait de node), on ne va pas s'abonner à ces topic un a un en remplissant la liste des SubscribeHandeling de l'inspector à la main. On s'abonne donc au chain joker *localisation/+/#* via la fonction connectionMQTT du scripte TopicSubscibeHandeling qui nous permettra de recevoir tous les messages envoyés sur les topic de type localisation/{nom du node}/{type du node}. Cependant en l'état on a aucun moyen de gérer les informations des différents nodes individuellement.

Affin d'automatiser la création et la position des nodes dans la scènes nous avons besoin de trier les différent topic inclue dans le chain joker et de créer des scriptes dont le role est de réagir dynamiquement aux informations reçu sur ces topic. L'organisation va donc être la suivante, on a trois types de node dans les topic MQTT (setup, mobile et enstimation), on va donc trier les message entres ces trois type, générer via un scripte un modèle 3D correspondant à un des 3 types selon le message reçu. Ce modèle 3D aura un scripte dépendant de son type qui lui sera associé et qui servira à mettre à jour sa position à chaque message de localisation reçu sur le broker MQTT.

Le tri des message se fait dans la fonction HandleIncomingMessage du scripte TopicSubscribeHandeling. Jusque là ce scripte regardait de quel topic de la liste des SubscibeHandeling le message reçu venait et appliquait la fonction associée. Ici avec les message venant de chain joker on reçoit des message venant des topic de localisation ne

se trouvant pas dans la liste, on va donc ajouté une condition servant à verrifier si on a bien trouvé un topic correspondant au message reçu. Si ce n'est pas le cas on verifie auquel des trois type appartient le topic à l'origine du message et on lance la fonction du scripte NodeGenerator associée. Ensuite on ajoute ce topic à la liste.

Comme son nom l'indique le scripte NodeGenerator sert à créer les nodes dans la scène, il est associé à un objet vide nomé Nodes qui sera le parents de toute les objet de nodes réel (les estimés sont placés dans un autre objet vide). NodeGenerator à 3 fonction associées au 3 type de node, quand l'une d'elle est appelée elle récupère un template de modèle 3D correspondant au type de node, le duplique et le renomme selon les information du topic puis lance la fonction de localisation associé à ce template. En effet chacun des trois template est associé à un scripte de localisation selon si c'est un node fixe, mobile ou estimé (respectivement SetNodePosition, SetMobileNodePosition et SetEstimatedNodePosition). Ces scriptes sont mis en marche par NodeGenerator via la fonction setupMQTT qui peut varier d'un type de node à l'autre selon les paramètres à initialiser mais qui à pour effet commun d'aller chercher dans la liste le topic associé à ce node nouvellement créé et d'y associer un listener qui servira à chaque nouveau messages sur ce topic de lancer la fonction Handleupdate qui récupère les information du json et met à jour la position du node dans la scène en fonction de ces informations.

Le scripte de positionnement des nodes estimé fait appel à une fonction nommée DrawEstimationLine du scripte DrawLine (lui aussi associé à l'objet vide Nodes) qui permet de tracer un objet de type LineRenderer entre deux point, ici entre la position réel et la position estimé du node.

ATTANTION : chaque nouveau node ajouté sera pris en compte s'il est dans l'une des trois catégorie mentionnée plus haut, CEPANDANT si vous souhaitez ajouter une nouvelle catégorie de node il faudra créer un nouveau template (on peut le faire en modifiant légèrement un des template existant), créer un nouveau scripte de localisation (idem ici ça peut être du quasi copier collé d'un déjà existant) à associer au template, une nouvelle fonction dans NodeGénerator pour générer ce type de node ainsi qu'une ligne dans la fonction HandleIncomingMessage pour gérer ce nouveau cas et appelé la nouvelle fonction de NodeGenerator associée.

3.3-État ouvert et fermé des portes

Les portes du laboratoire on des capteurs permettant de savoir si elles sont ouvertes ou fermées. On a donc pensé qu'il serait intéressant de récupérer leur état en temps réel étant donné que cela peut influencer les mesure de ranging et de position des nodes. La méthode employé se repose sur des principes déjà vu plus haut.

Dans un premier, temps on s'abonne au chain joker suivant, *api/3/room/+/sensor/TS0203/id/1/indication*, où le + correspond au nom de la porte. De la même manière que pour les node, on reçoit des message de topic ne figurant pas dans la

liste SubscribeHandeling, cependant ici on ne génère rien dynamiquement puisque les portes sont déjà placées dans la scène. Donc ici lorsque l'on reçoit pour la première fois un message correspondant à l'état d'une porte on recherche la porte concernée dans la scène grâce à son nom, et on active le scripte DoorState associé via sa fonction SetupMQTT. Cette fonction va ajouter un listener au topic associé à la porte (que l'on aura préalablement ajouté à la liste) afin de mettre à jour dynamiquement l'état de la porte grace à la fonction Handleupdate de DoorState.

Pour mettre à jour de la porte visuellement dans la scène, on lui associe une animation avec deux points clés, un correspondant à son état fermé et un autre à son état ouvert. De manière similaire au rail on lui associe un BlendTree qui ici ira de 0 à 1 (correspondant respectivement aux états ouvert et fermé). De cette manière dans le scripte quand l'information de « contact » (donc de fermeture) de la porte sera vrai on paramétrera l'animation de la porte sur 1 et sur 0 sinon. Aussi, étant donnée que les capteurs ont tendance tomber en panne de batterie assez souvent, on affiche les portes par défaut en rouge transparent et on ne leur affecte leur vraie couleur que si l'on reçoit un message sur leur topic, ainsi on ne risque pas d'afficher des informations fausses dans la scène puisque l'on peut différencier visuellement les portes dont les capteurs sont fonctionnels de celle qui ne le sont pas.

ATTENTION : Dans le chain joker le terme TS0203 correspond au type de capteur, si le type de capteur utilisé change il faudra créer un nouveau cas pour ces nouveau capteur (cela nécessitera seulement de changer le terme qui change dans le chain joker). Cependant pour que les scripte actuels fonctionne toujours il faudra que ce soit toujours des capteurs qui renvoient une information true ou false, si vous souhaitez ajouter des capteur qui mesures l'angle de la porte il faudra créer un nouveau scripte qui gère ce cas.

En bonus, afin de faciliter la lisibilité de là scène, j'ai rajouté une fonction qui permet de rendre les porte semi-transparente quand on s'en approche et de les rendre à nouveau opaques lorsqu'on s'en éloigne. Le but est que les utilisateurs comprennent que ces portes ne représentent aucunement une frontière et qu'il peuvent passer à travers.

3.4-Représentation des mesures de Ranging et des erreurs associées

Enfin la dernière fonction qui nous intéresse est la représentation des mesures de ranging dans l'espace 3D. Deux type d'affichage ont été paramétrés, l'un via des lignes reliant les nodes ancres au node détecté et l'autre via des sphères dont le centre est le node ancre.

Les informations de ranging sont reçues sur le broker MQTT dans des topics du type **ranging/{id_du_node_détecté}/{id_de_l'ancre}/indication**. On va donc, dans le

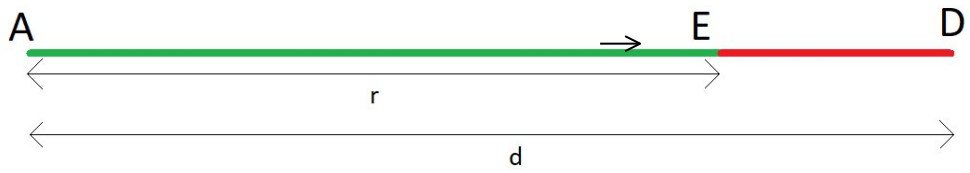
scripte SubscribeHandeling, s'abonner au chain joker *ranging/+/+/indication* et créer un nouveau cas à prendre en compte pour gérer les messages venant de ces topics en appelant une nouvelle fonction de NodeGenerator servant à créer un objet vide qui pilotera les affichage de ranging. Cette objet est crée à partir d'un template d'objet Empty auquel on à associé le scripte Ranging.

Lorsque l'objet empty est crée on cherche son node ancre associé et on en fait son parent. Et enfin de façon analogue aux localisation de node on appel la fonction SetupMQTT de son scripte Ranging pour initialiser les différents paramètres utiles et pour ajouter un listener qui lancera le fonction Handleupdate à chaque message reçu sur le topic de ranging de l'ancre concernée. Cette fonction à pour but de mettre à jour la représentation du ranging à chaque message reçu sur le topic associé. Pour déterminer si la représentation de ces mesure est rectiligne on fait appel à deux paramètres booléen (repRect et repShere) appelé du scripte DrawLine, si l'un est vrai on représente via des ligne et on représente par des sphères sinon. Pour déterminer si ces paramètre sont true ou false on utilise à des fonction qui change leur état dans le scripte DrawLine (SetRepRect et SetRepShere). Ces fonction sont appelé via les boutons du panel Ranging de la tablette, voilà pourquoi ces paramètre sont dans le scripte DrawLine, car nous avons besoin qu'il soient dans le scène au démarrage pour les associer au bouton (contrairement aux scripte Ranging qui est associé à des objet placé dynamiquement dans la scène).

Représentation rectiligne :

Si RepRect est true on choisi la méthode de représentation par ligne reliant l'ancre au node détecté. On a un paramètre appelé de DrawLine lui aussi qui vérifie si l'on prend pour le node détecté sa position réelle ou estimé. Lui aussi est piloté via une fonction appelé par les boutons correspondant de la tablette. Cependant peut importe si l'on choisi la position réel ou estimé la méthode est le même. Ce que l'on veut faire c'est tracer une ligne verte partant de l'ancre et suivant la droite reliant l'ancre au node détecté, cette ligne est d'une longueur égale au Ranging mesuré. Ensuite on veut tracer une ligne rouge qui part du bout de la ligne vert et qui va jusqu'à la position du node détecté. Pour celà on va créer une fonction DrawRangingLine dans le scripte DrawLine qui tracera ces deux ligne. Etant donné que l'on trance des ligne entre deux points il nous faut connaître les coordonnée des 3 point, deux sont connue (la position des nodes) il nous reste donc à déterminé la coordonné de jonction du point de jonction entre les 2 lignes.

Pour résoudre ce problème on se pose donc dans un cas où l'on a une droite (AD) dont les coordonnées des points A (x_A,y_A,z_A) et D (x_D,y_D,z_D) sont connue (A représente l'ancre et D représente le node détecté). On souhaite déterminer les coordonnées d'un point E (x_E,y_E,z_E) se trouvant sur cette droite et l'on connaît pour ça $d = AR$ La distance entre les points A et R et $r = AE$ la distance entre les points A et E (égale au ranging). On défini le vecteur $\overrightarrow{AD}$ de coordonnées $(x_{AD} = x_D - x_A, y_{AD} = y_D - y_A, z_{AD} = z_D - z_A)$



Soit la droite (AD) passant par A de vecteur directeur $\overrightarrow{AD}$, tout point de coordonnées (x,y,z) appartenant à cette droite respecte l'équation paramétrique suivante :

$$\begin{cases} x = x_A + x_{AD}t \\ y = y_A + y_{AD}t \\ z = z_A + z_{AD}t \end{cases}$$ Avec t un nombre réel

Donc les coordonnées de E peuvent s'exprimer de la façon suivante :

$$\begin{cases} x_E = x_A + x_{AD}t \\ y_E = y_A + y_{AD}t \\ z_E = z_A + z_{AD}t \end{cases}$$

Il nous faut donc déterminer t, pour cela on commence par définir le vecteur $\overrightarrow{AE}$ de coordonnées :

$$\begin{cases} x_{AE} = x_E - x_A \\ y_{AE} = y_E - y_A \\ z_{AE} = z_E - z_A \end{cases}$$

$$\begin{cases} x_{AE} = x_A + x_{AD}t - x_A \\ y_{AE} = y_A + y_{AD}t - y_A \\ z_{AE} = z_A + z_{AD}t - z_A \end{cases}$$

$$\begin{cases} x_{AE} = (x_D - x_A)t \\ y_{AE} = (y_D - y_A)t \\ z_{AE} = (z_D - z_A)t \end{cases}$$

On peut ensuite se servir des propriétés du produit scalaire de deux vecteur colinéaire de même sens :

$$\overrightarrow{AD}.\overrightarrow{AE} = AR * AE = d * r$$

Or $\overrightarrow{AD}.\overrightarrow{AE} = x_{AD} * x_{AE} + y_{AD} * y_{AE} + z_{AD} * z_{AE}$

Donc,

$$x_{AD} * x_{AE} + y_{AD} * y_{AE} + z_{AD} * z_{AE} = d * r$$

$$(x_D - x_A)^2 t + (y_D - y_A)^2 t + (z_D - z_A)^2 t = d * r$$

$$t = \frac{d * r}{(x_D - x_A)^2 + (y_D - y_A)^2 + (z_D - z_A)^2}$$

On crée donc une fonction RangingErrorPoint dans le scripte DrawLine qui permet de calculer les coordonnées de E à partir des formules trouvées précédemment. Cette fonction sera appelé par DrawRangingLine et les distance entre les node et les mesure de ranging sont récupéré dans le json publié sur le topic de ranging. A noter que selon les protocoles des expérience de ranging les fichiers json peuvent être différents mais les informations qui nous intéressent s'y trouve toujours et sont toujours nommées de la même manière donc la fonction Unity JsonUtility parvient toujours à les trouver malgré le fait qu'on définisse une classe qui n'aura pas forcément le même format que le json.

Représentation sphérique :

Si RepSphere est true on choisi la représentation du ranging avec des sphères. Ces sphère on pour centre l'ancre qui fait la mesure de ranging et la valeur de leur raillon est égale au ranging mesuré. Une sphère représente donc l'ensemble des points potentiels où pourrait se trouver le node détecté selon une ancre seule, en principe le croisement de ses sphère nous donne donc une estimation de la position du node détecté, en pratique le résultat est difficilement lisible.

Pour faire cela on se contente de dupliquer un template de sphère, de le centrer sur l'objet empty servant piloter les représentations du ranging dans la scène et à lui attribuer à chaque update une scale égale à deux fois le ranging (car son diamètre vaut deux fois le ranging). Pour tenter de rendre cette représentation plus lisible on a défini une liste de couleur dans DrawLine dans laquelle on pioche sans remise à chaque création d'ojet vide de ranging, les sphères ont donc toute une couleur unique mais celà ne rend que très peu service à la lisibilité.