Enseignant référent : M. Eric CARIOU
Encadrants de Stage : M. Brice SAINT-MARTIN et M. Cyril DE VEYRINES
Etablissement : Université de Pau et des Pays de l'Adour - 64000 Pau
Entreprise d'accueil : Safran Engineering Services - 64510 Bordes

Remerciements

En premier lieu, je souhaite remercier mes encadrants de stage, M. Brice SAINT-MARTIN et Cyril DE VEYRINES, pour leur patience, leur sympathie ainsi que leurs précieux conseils et nombreuses connaissances qu'ils ont pu m'apporter. Mais aussi merci à M. Eric CLAVE chef de service auprès de Safran Engineering Services et responsable des stages pour son accueil chaleureux, son écoute et sa bonne humeur.

Je remercie également toute l'équipe pédagogique de l'Université de Pau et des Pays de l'Adour pour la qualité de l'enseignement offert, ce qui me permet aujourd'hui de faire un stage d'une aussi grande qualité.

Enfin je désire aussi remercier tous les collègues de SES, pour leur aide, pour la bonne ambiance et pour leur sympathie dans le cadre de ce stage.

Table des matières :

I. Présentation de l'entreprise

Safran est un groupe industriel et technologique de plusieurs entreprises spécialisées dans trois domaines : l'Aéronautique, l'Espace et la Défense. Créé le 11 mai 2005 par la fusion de *Snecma* et *Sagem*, *Safran* est le plus ancien motoriste d'aviation au monde.

En mars 2010, la branche ingénierie et technologie de Labinal fusionne avec *Teuchos*, membre du groupe *Safran*, et prennent ensemble le nom de *Safran Engineering Services*.

En mars 2016, le Groupe décide de regrouper toutes ses sociétés sous une bannière commune et une seule marque : Safran. Les sociétés intègrent toutes Safran dans leurs dénominations ainsi qu'un descriptif de leur activité. Toutes les filiales s'unissent sous une seule et même signature de marque *"Powered by trust"*.

Figure 1 : Changement de nom des entreprises Safran

Je fais mon stage auprès de *Safran Engineering Services* dans le site de Bordes (64510), dans le bâtiment *Aéropolis*, au service *Solutions Logicielles*. À ne pas confondre avec le site situé au même endroit de *Safran Helicopter Engines* (anciennement *Turbomeca*).

SAFRAN
Engineering Services

Figure 2 : Logo Safran Engineering Services

Aujourd'hui *Safran Engineering Services*, appartient au groupe *Safran* et est une filiale de *Safran Electrical & Power* (anciennement *Labinal Power Systems*). *Safran Engineering Services* offre des services en ingénierie de haute technologie dans les domaines de l'aéronautique, de l'énergie et du transport terrestre.

Avec plus de 3700 ingénieurs et techniciens, l'entreprise dispose de plusieurs entités opérationnelles réparties dans le monde (France, Allemagne, Royaume-Uni, Espagne, États-Unis, Mexique, Maroc, Canada, Brésil et Inde) combinant l'expertise en matière de systèmes électriques, d'aérostructures, de systèmes mécaniques, de logiciels ainsi que de système électroniques embarqués.

II. Le stage

1. Problématique et Contexte

L'intitulé du stage était de réaliser des "Proofs of concepts" (des démonstrations de faisabilité) pour *Safran Engineering Services* en réalité virtuelle avec les technologies *Unity*, un moteur de jeu et le *HTC Vive*, un casque de réalité virtuelle. Il s'agit aussi de faire de la recherche et du développement afin de simplifier le développement de la réalité virtuelle pour Safran Engineering Services autour de leurs utilisations dans des cas comme : la visualisation et la manipulation de maquette 3D, de la formation dans un environnement virtuel, de la collaboration avec des scènes multi-utilisateurs etc…

Le projet de faire de la réalité virtuelle a commencé en 2018. Le but pour *Safran Engineering Services* est de pouvoir répondre à des besoins clients. Ces besoins sont multiples : Virtualiser des espaces pour pouvoir les transformer à moindre coût, communiquer autour d'un projet, former des personnes à des postes sans avoir à bloquer la chaîne de production…

« Dans un environnement industriel où tout se veut digitalisé, nous voulons trouver des solutions avec nos clients en les accompagnant dans leur réflexion afin d'utiliser au mieux les technologies d'aujourd'hui et de demain. Dans ce cadre, la réalité virtuelle est un outil puissant permettant de virtualiser des environnements existants ou bien de maquetter des projets. »

Brice Saint-Martin, responsable de projet réalité virtuelle chez Safran Engineering Services.

2. Les Moyens

2.1. Le HTC Vive

Le *HTC Vive* est un casque de réalité virtuelle développé par *Valve* (Les créateurs de *Steam*) et *HTC*. Pour fonctionner le casque est fourni avec 2 manettes sans fils, un boitier de connexion (USB, HDMI/DisplayPort et Alimentation) pour brancher le casque sur son ordinateur. Et enfin 2 *Light House* : des petits boitiers émettant de l'infrarouge permettant de récupérer la position spatiale du casque et des manettes.

Figure 3 : Le HTC Vive

Une des particularités du HTC Vive c'est qu'il utilise la technologie "Room Scale", contrairement à une utilisation statique où l'utilisateur doit toujours rester sur la même position, cela permet à l'utilisateur de pouvoir se déplacer virtuellement de la même manière qu'il se déplacerait réellement. Les limites réelles étant la salle ou le périmètre préalablement défini par l'utilisateur. Ces limites sont aussi représentées virtuellement dans le casque afin d'éviter que l'utilisateur sorte de la zone de jeu ou se cogne sans s'en rendre compte.

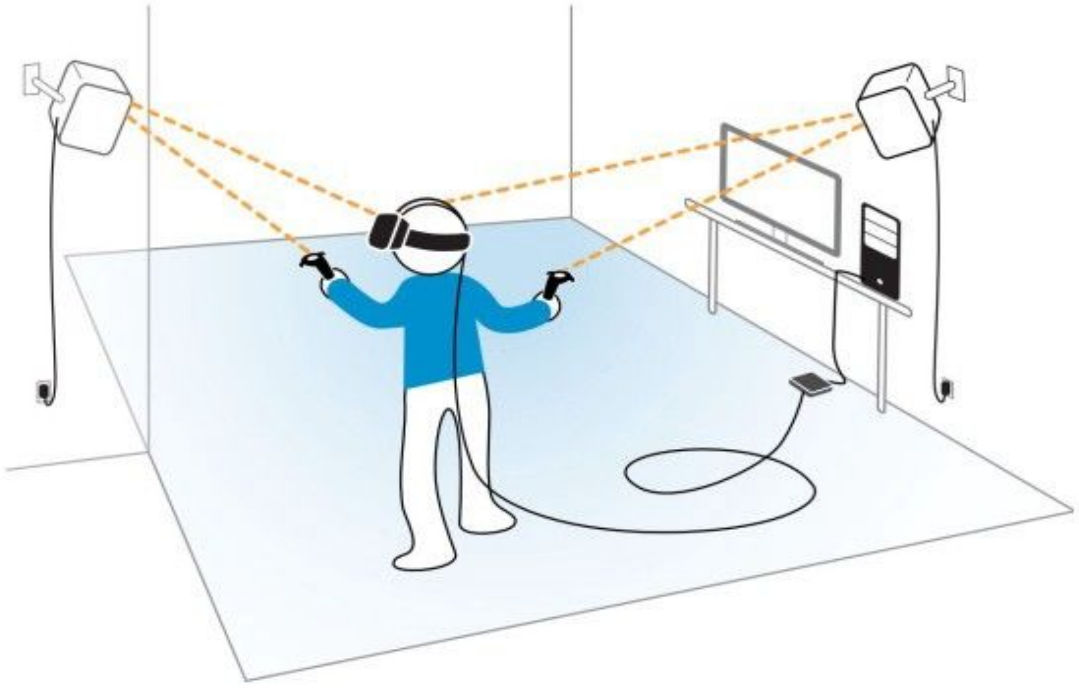


Figure 4 : Le “Room Scale”

En plus des accessoires de base du *HTC Vive* dont j’ai eu l’occasion d’utiliser, j’ai aussi pu tester l’adaptateur sans fil de *Vive*, qui permet d’utiliser le casque sans être encombré par le fil qui se branche normalement du casque à l’ordinateur. Cet accessoire utilise un capteur fonctionnant avec la technologie *60GHz WiGig* (qui signifie *Wireless Gigabit Alliance*, aussi connu sous le nom *60GHz Wi-Fi*) de Intel pour communiquer avec l’ordinateur. Le capteur est à placer de façon directement visible par le casque et requiert aussi l’installation d’une carte sur un port PCI-Express. Donc un des défauts est qu’il est pratiquement impossible pour un utilisateur d’un ordinateur portable d’utiliser cet accessoire. L’adaptateur se place à l’arrière du casque *HTC Vive*, et s’attache via des scratches. Il est ensuite alimenté par une batterie que l’on peut placer dans sa poche par exemple.

Figure 5 : Adaptateur sans fil Vive

J'ai aussi eu l'occasion de développer avec deux *Vive* Trackers. Ce sont des petits objets fonctionnant un peu comme des manettes. Leur intérêt est de permettre aux développeurs de pouvoir traquer d'autres objets de la vie réelle afin de pouvoir les afficher virtuellement, par exemple avec une raquette dans une application simulant une partie de Tennis. Dans mon cas j'ai eu l'occasion de développer une petite application permettant de détecter les pieds de l'utilisateur à l'aide des deux *Vive* Trackers.

Figure 6 : Vive Trackers

2.2 Le Leap Motion

Le *Leap Motion* est une technologie qui a commencé à être développée en 2008, par Michael Buckwald et David Holz, mais c'est aussi le nom de l'entreprise, *Leap Motion, Inc.*, qui le fabrique actuellement. Cette technologie utilise une caméra infrarouge afin de pouvoir détecter et représenter via une animation temps réel ses mains ainsi que chacun de ses doigts sur un écran.

Figure 7 : Le Leap Motion

Pour le stage le *Leap* était attaché sur le casque *HTC Vive* et nous avons branché la caméra via un câble *micro-USB 3.0* sur la prise *USB* du casque. Avec cette configuration on peut utiliser cette technologie afin de pouvoir se passer des manettes du *HTC Vive* quand on utilise le casque de réalité virtuelle. Mais pour faire fonctionner cela il faut passer par un développement sous *Unity*.

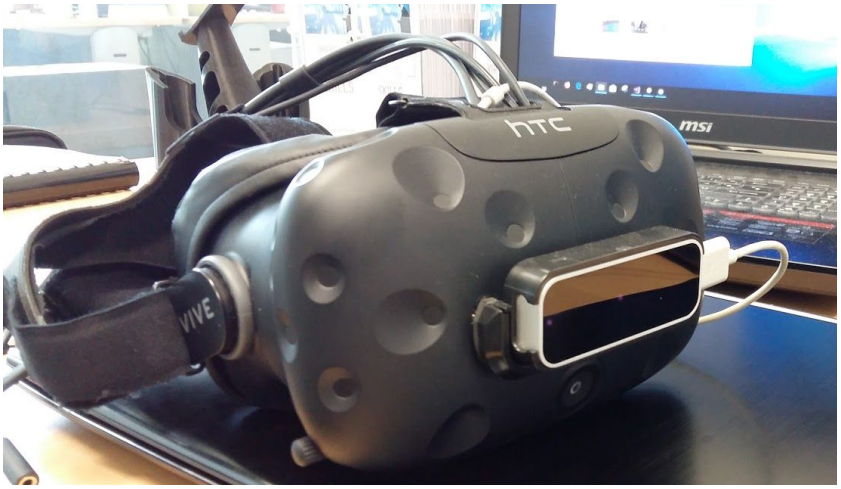


Figure 8 : Le Leap avec Le HTC Vive

2.3. Unity

Unity est un moteur de jeu qui est très répandu dans l'industrie du jeux-vidéos par le fait qu'il soit facilement multiplateforme (sur PC, mobile, console de jeux...) ainsi que pour sa facilité de prototypage. Le *HTC Vive* et le *Leap* sont des technologies compatibles avec le moteur de jeu *Unity*. Pour cela il faut aller dans l'*Asset Store*, c'est un marché en ligne du moteur de jeu permettant de récupérer les *packages Unity* (= librairies) de *HTC Vive* (*Steam VR*) et du *Leap Motion* qui sont fournis gratuitement par leur propriétaire respectif.

Figure 9 : Logo Unity

L'interface *Unity* (voir figure 9) fonctionne comme suit :

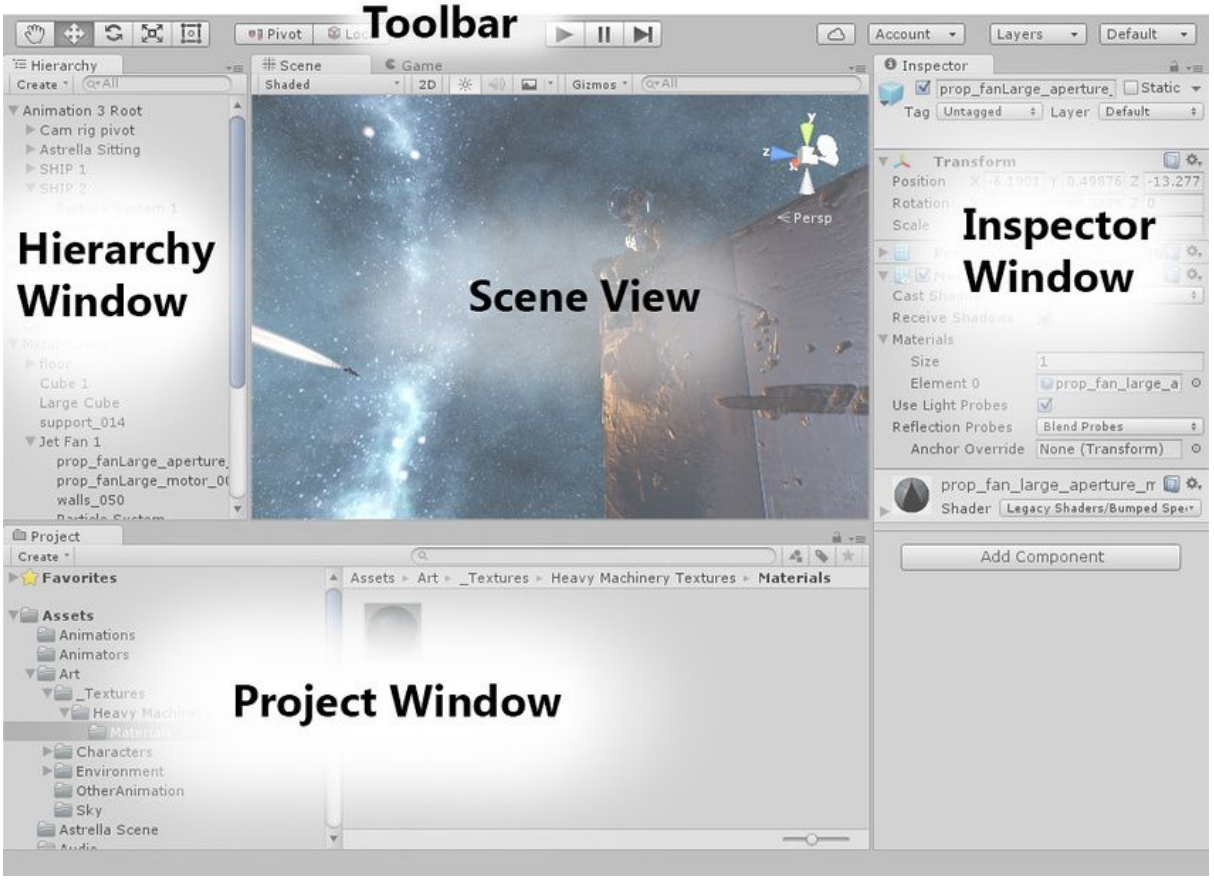


Figure 10 : Interface de Unity

La barre d'outils en haut contient des boutons essentiels pour *Unity*, comme le bouton *play*, utilisé pour démarrer le jeu afin de le tester. Les boutons en haut à gauche permettent de

déplacer des objets situés sur la scène (écran du milieu). La scène permet de visualiser le jeu en utilisant une caméra libre. Contrairement à l'onglet *"Game"* qui permet de visualiser le jeu via le point de vue du joueur.

La hiérarchie à gauche montre le contenu de la scène. Chaque objet (aussi appelé *GameObject*) de la scène peut être hiérarchisé, et donc posséder un parent et/ou des enfants. Le déplacement ou la rotation d'un objet parent entraîne la même action auprès de ses objets fils. Ce qui permet de créer des objets composés, comme une voiture qui serait composé de roues, carrosserie, portes, capot ...

En bas c'est la fenêtre de projet qui permet de parcourir les "assets" (les fichiers) du jeu. Parmi les "assets" on peut trouver des "prefabs" qui ne sont pas à confondre avec des *GameObjects* (les objets de la scène). Les prefabs sont en fait comme des plans d'objet. Si on modifie un prefab, on va modifier tous les objets qui sont des instances de ce prefab.

Chaque objet possède un ou plusieurs composants, que l'on peut voir dans la fenêtre *"inspector"* à droite lorsqu'on sélectionne un objet ou un prefab. Le composant de base de tout objet est le composant *"Transform"*. Ce composant informe sur la position et la rotation de l'objet par rapport à la scène ou un autre objet. Il permet aussi de récupérer son parent ou un de ses fils. Beaucoup de composants sont proposés de base avec *Unity*. Mais l'utilisateur a la possibilité d'en créer à volonté en générant des assets spécifiques nommées "script".

Même si le "runtime" de *Unity* a été programmé en *C++*, on peut utiliser *Visual Studio* et *C#* en union avec l'interface *Unity* de base dans le but de laisser l'utilisateur la possibilité d'éditer des objets "script" et ainsi faire des choses plus complexes que ce qui a été proposé par défaut. En fait à chaque fois que l'utilisateur crée un élément "script" cela crée en parallèle une classe *C#* du même nom et héritant de *"MonoBehaviour"* (c'est la classe de base dont chaque script hérite).

Figure 11 : Logo Visual Studio

Via la fenêtre "inspector" de *Unity* l'utilisateur peut attacher les scripts sous forme de composants aux différents objets de la scène (*GameObject*) ou aux prefabs afin de pouvoir réaliser différentes tâches. Tous les attributs publics de la classe vont pouvoir être éditable directement dans "l'inspector". Ensuite l'utilisateur possède quelques méthodes de base qu'il

peut surcharger comme *“Start()”* ou *“Update()”*. *“Start()”* permet d’exécuter une fois du code à la création de l’objet et *“Update()”* permet d’exécuter du code à chaque frame.

J’ai aussi utilisé *Visual Studio* en dehors du contexte de *Unity*, généralement pour créer des petits utilitaires comme un programme console qui permet de supprimer les couleurs en doublons dans un modèle 3D afin de réduire son poids, un programme qui permet de lancer une application lorsqu’une autre est fermée, des programmes d’exemples pour présenter des designs patterns.

3. Projets et Problématiques

3.1. Salle Attente VR

Le but de ce projet a été d'appréhender les outils à ma disposition. J’avais déjà quelques bases sur *Unity* et sur le casque *HTC Vive*, ainsi qu’avec le C# et la programmation orientée objet grâce à mes cours de C++ et surtout de Java à l’Université (C# est très similaire à Java), ce qui m’a permis de débuter assez rapidement. J’ai commencé par créer un projet avec une scène assez simple : une surface plane et la compatibilité avec le *HTC Vive* grâce au *package* de *Steam VR*.

Puis j’ai petit à petit essayé de faire des choses plus complexes : pouvoir attraper des objets et se téléporter en utilisant les manettes. Heureusement ces fonctionnalités de base sont déjà inclues dans le *package Steam VR*. Je n’ai donc pas eu besoin de les programmer de zéro. Mais pour faire tout cela il a fallu que je m’inspire de projets exemples fournis avec le *package Steam VR*, ainsi que de lire de la documentation sur Internet.

Ensuite le deuxième but de ce projet a été de réaliser une salle d’où l’on peut lancer n’importe quelle application de réalité virtuelle. Pour cela j’ai créé une interface qui affiche le contenu d’un dossier pouvant contenir des exécutables. Ensuite j’ai développé un script générant un *GameObject* en forme de laser, qui peut être activé en appuyant sur un bouton de la manette. Je l’ai ensuite placé en fils du *GameObject* représentant la manette dans la hiérarchie, ce qui permet au laser de suivre les mouvements de la manette. Puis j’ai programmé le laser afin de pouvoir sélectionner les exécutables à lancer. Pour cela je vérifie l’objet que le laser touche, si c’est un objet exécutable et que le joueur appuie sur un bouton, je lance l'exécutable au chemin contenu dans un attribut d’un des composant de l’objet sélectionné. Pour rappel les composants sont des représentations d’instances de classes C#.

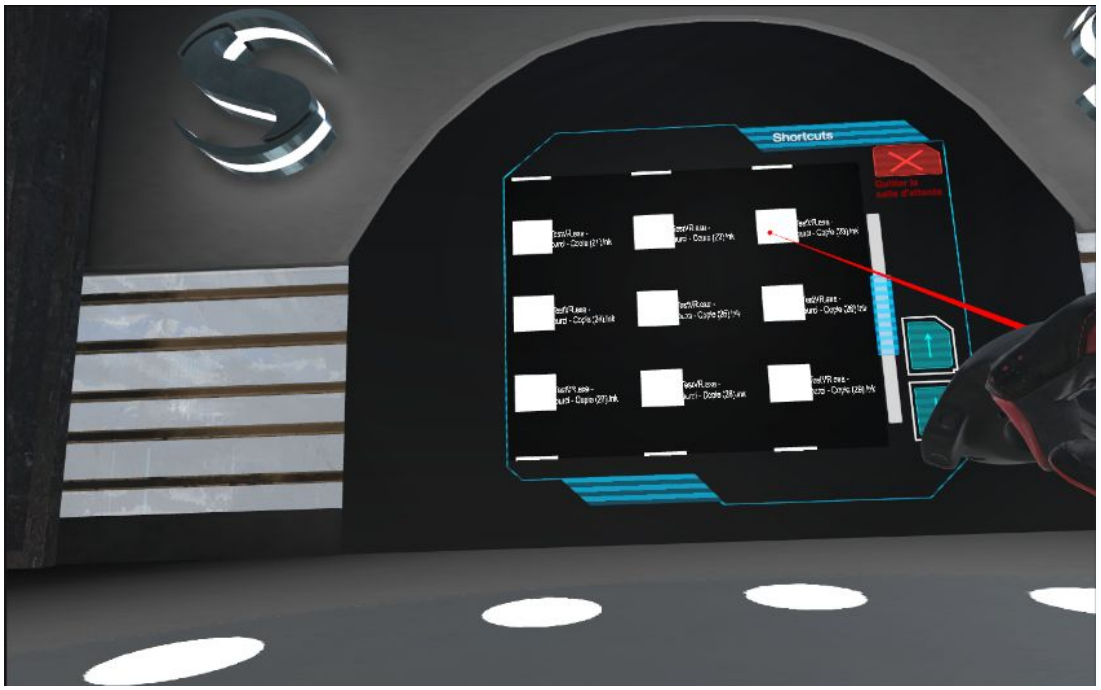


Figure 12 : Interface de sélection d'application dans la salle d'attente VR

Un des besoins était de faire en sorte que la salle d'attente réapparaît lorsque l'utilisateur ferme une des applications VR. L'objectif est de faire comme un hub principal pour tous les projets de réalité virtuelle de l'entreprise. Mes encadrants ont d'abord pensé faire un service pour s'occuper de ça. En développant l'application et en lisant de la documentation, je me suis rendu compte que depuis Windows Vista il est impossible à un service sur Windows d'exécuter une application car les services ne sont plus associés à aucun utilisateur, et il en a besoin pour exécuter une application. Cela a été fait par Windows dans un but de sécurité. J'ai donc créé une application console classique s'exécutant en arrière-plan. J'ai réussi à faire fonctionner l'application assez rapidement en reprenant le code que j'avais déjà écrit pour le service.

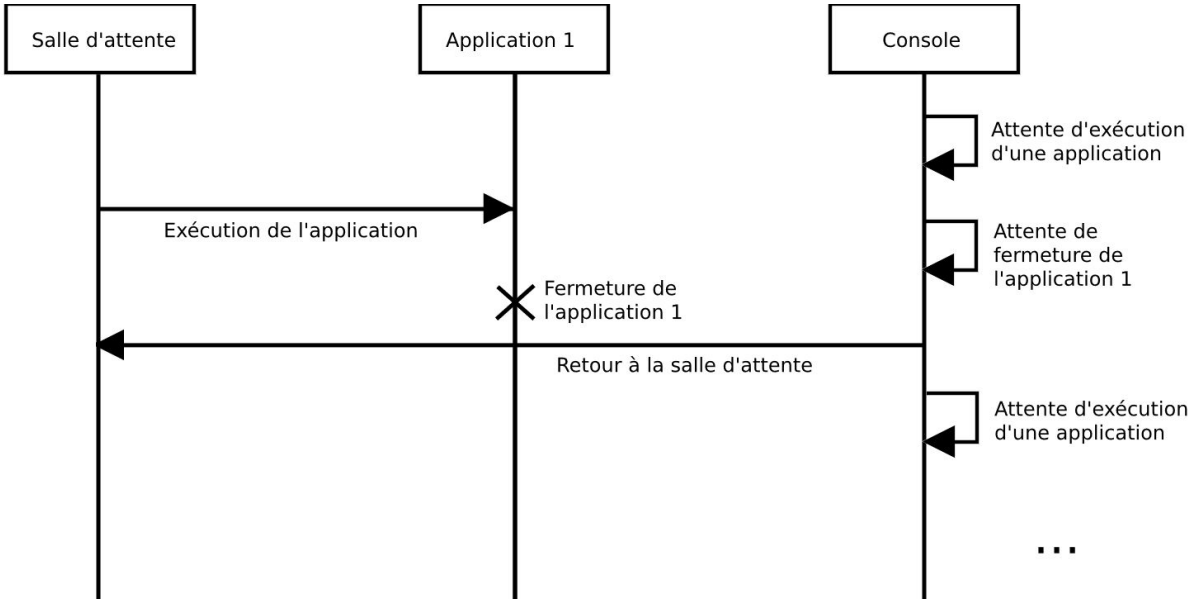


Figure 13 : Diagramme de séquence du fonctionnement de la console

Avec ce projet j'ai appris beaucoup de chose. Tout d'abord, même si j'avais les bases, j'ai eu l'occasion de m'améliorer avec *Unity* et d'un peu mieux comprendre les notions de base comme les *prefabs, GameObjects, scripts*… Mais cela m'a aussi permis de comprendre le fonctionnement de la création d'interface (*GUI*) ainsi que la création d'un laser de sélection avec *Unity*. J'ai aussi découvert comment développer des services ainsi que leur fonctionnement via Windows et au passage apprendre à utiliser *Visual Studio*.

3.2. Laser de téléportation avec le Leap

Ce projet m'a permis de découvrir la programmation avec la technologie du *Leap*. Le but initial du projet était de débugger une application de manipulation de maquette 3D développé avec le *Leap* et le *HTC Vive*, puis de rajouter la téléportation avec la manette.

Un des "bugs" de l'application était que lorsqu'un effet "éclaté" (éloignement des différents éléments pour mieux les visualiser individuellement) était appliqué sur la maquette 3D. Le repositionnement des éléments sur leur position de base ne se faisait pas au bon endroit. En examinant le code j'ai réussi à situer le problème et à le corriger assez rapidement. J'ai ensuite réalisé quelques optimisations, notamment sur l'interface de l'utilisateur ainsi que sur l'environnement.

Concernant le deuxième objectif, ne trouvant pas la téléportation avec la manette dans une application avec *Leap* très pratique (le fait de devoir enlever le casque à chaque fois pour récupérer la manette), j'ai dû donc recréer de zéro un système équivalent à la téléportation avec Steam VR mais utilisant le *Leap*. Avec ce nouveau système il suffit de pincer l'index et le pouce pour faire apparaître un laser. Cela a été assez rapide à faire car la détection de l'action de pincer avec les doigts avait déjà été réalisé par mes encadrants pour permettre à l'utilisateur d'attraper un objet de la même manière. Pareillement, pour faire le laser j'ai réutilisé le code de celui de la salle d'attente. Ensuite en visant l'endroit où on veut aller et en récupérant la position de contact entre le sol et le laser, j'ai fait en sorte qu'on s'y téléporte en séparant l'index et le pouce.

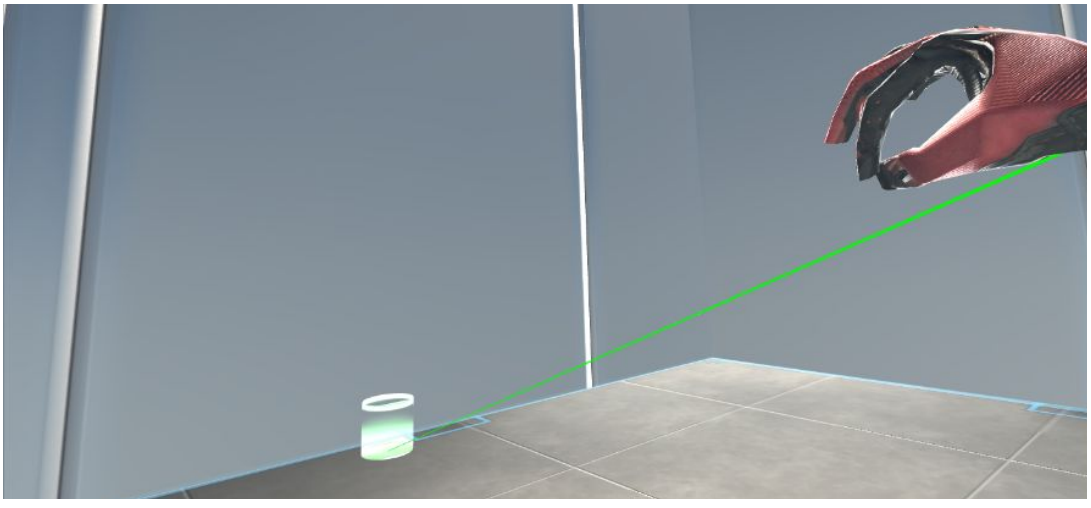


Figure 14 : Laser de téléportation avec le Leap

Un problème que j'ai eu est que la position du joueur dans la scène dépend en fait du périmètre de jeu qu'il a défini avec le HTC Vive. Donc lorsque le joueur sélectionne un endroit pour s'y téléporter, c'est le centre du périmètre qui se retrouve sur le point souhaité (donc généralement le centre de la salle où on est). Le joueur se retrouve quant à lui aussi loin du point qu'il est loin du centre du périmètre. Pour régler ce problème j'ai dû calculer la distance du casque (sans compter la hauteur) et du centre du périmètre, afin de la soustraire lors de la téléportation.

Un autre problème était qu'on ne voulait pas que l'utilisateur se téléporte s'en faire exprès. En effet le Leap n'étant pas 100% précis au niveau des mouvements et peut parfois interpréter un pincement là où il ne devrait pas y en avoir. La solution pour cela a été de faire la téléportation seulement lorsque l'utilisateur active un certain bouton.

Avec ce projet, j'ai eu l'occasion d'apprendre comment utiliser les techniques de debugging avec Visual Studio (Utilisation des points d'arrêt, parcourir le code pas-à-pas, inspection de variables...). Cela m'a aussi permis d'avoir un aperçu de la programmation avec le *Leap*. Ce qui me sera très utile pour plus tard.

3.3. Création de package pour les inputs Vive et Leap

L'objectif de ce projet était que je crée des *events* pour les actions pouvant être réalisées avec une manette *Vive* et avec ses mains via le *Leap*. Et ensuite d'en faire un *package* afin de pouvoir réutiliser ce système dans d'autres projets et simplifier notre utilisation de la réalité virtuelle. Un des besoins étant de pouvoir aussi repérer quand l'utilisateur fait un balayage avec sa main dans n'importe quelle direction souhaitée. Par exemple pour parcourir un menu.

La première étape de ce projet a été de me documenter sur l'utilisation des *events* en C#. J'ai donc décider d'utiliser les *events* avec les objets *delegate*. Un *delegate* est un moyen de dire à C# quelle méthode appeler quand un *event* est déclenché. Par exemple, si on clique sur un bouton d'un formulaire, le programme appellera une méthode spécifique. C'est ce pointeur qui est un *delegate*. Les *delegate* sont efficaces, car on peut notifier plusieurs méthodes qu'un événement s'est produit, si on le souhaite.

J'ai donc utilisé ces *events* en C# pour repérer, en utilisant les *packages Steam VR* et du *Leap*, toutes sortes d'inputs manettes, ainsi que des mouvements de mains comme lever un certain nombre défini de doigts, le fait de pincer avec son indexe et le pouce, la direction de la paume de la main ou encore la proximité de la main avec un objet défini.

Le problème avec le balayage c'est que le *package Leap* ne permet pas de repérer cela de base. J'ai donc commencé par créer un système calculant la vélocité de la main en se basant sur sa position et le temps. Le calcul est le suivant :

Vélocité de la main = (Position de la main à l'instant T - Position de la main à l'instant T-1) / le temps en seconde pour arriver au frame précédent

À partir de là il suffit de lever un *event* lorsque la vélocité atteint une certaine valeur et parcourt une certaine distance. La valeur max de vélocité et la valeur de la distance à parcourir étant configurable. Ce système permet de repérer des mouvements de balayage avec la main du *Leap* dans toutes les directions.

3.4. Création de package multi-utilisateur

Le but de ce projet a été de créer un *package* permettant d'utiliser la réalité virtuelle avec plusieurs utilisateurs en se connectant via le réseau. Pour ce faire, j'ai utilisé le *HLAPI* (*High Level API*) de *UNet* (*Unity Networking*) créé par *Unity* qui est généralement utilisé pour faire des jeux multijoueurs.

3.4.1 Présentation de UNet

Figure 15 : Logo Unet

Le *HLAPI* de *UNet* est un système qui a été conçu pour faciliter la vie du développeur et donc de s'occuper en arrière-plan de toutes les tâches récurrentes du développement d'un jeu en réseau. Pour fonctionner, il utilise un serveur et des clients. Dans le cas d'un hébergement local, c'est une des machines clients qui fait aussi tourner le serveur. C'est dans ce cas d'utilisation que j'ai utilisé *UNet*.

Une difficulté du développement d'application en réseau avec de la réalité virtuelle est qu'il est très compliqué de pouvoir tester localement sur une seule machine deux clients, en effet les machines ne sont pas assez puissantes pour pouvoir faire tourner deux applications en Réalité virtuelle simultanément. Heureusement, dans mon bureau, j'étais équipé de deux ordinateurs puissants (avec une carte graphique *GTX 1080 ti* et un processeur *i7*), connectés via un câble ethernet afin de réaliser ces tests. Le problème c'est qu'Unity, pour des raisons de sécurité, empêche deux applications en environnement de développement de pouvoir se connecter via *UNet*. J'ai alors dû utiliser des build de test de *Unity* ce qui m'a permis de pouvoir faire du debugging sur les deux machines en même temps via visual studio.

Avec *UNet* le développeur a à sa disposition la création de méthodes en plaçant des attributs personnalisés au-dessus. Pour pouvoir exécuter du code d'une méthode sur le serveur depuis un client, il faut utiliser l'attribut *[Command]* et si on veut exécuter le code sur un client depuis le serveur, l'attribut est *[ClientRPC]*. Pour simplifier on parle de méthode CMD ou RPC. L'utilisation de ces méthodes est possible uniquement sur l'objet utilisateur. Ils sont généralement utilisés dans le but d'envoyer des messages, faire de la synchronisation complexe, demander un changement "d'autorité" ou encore de changer une variable chez un client en particulier…

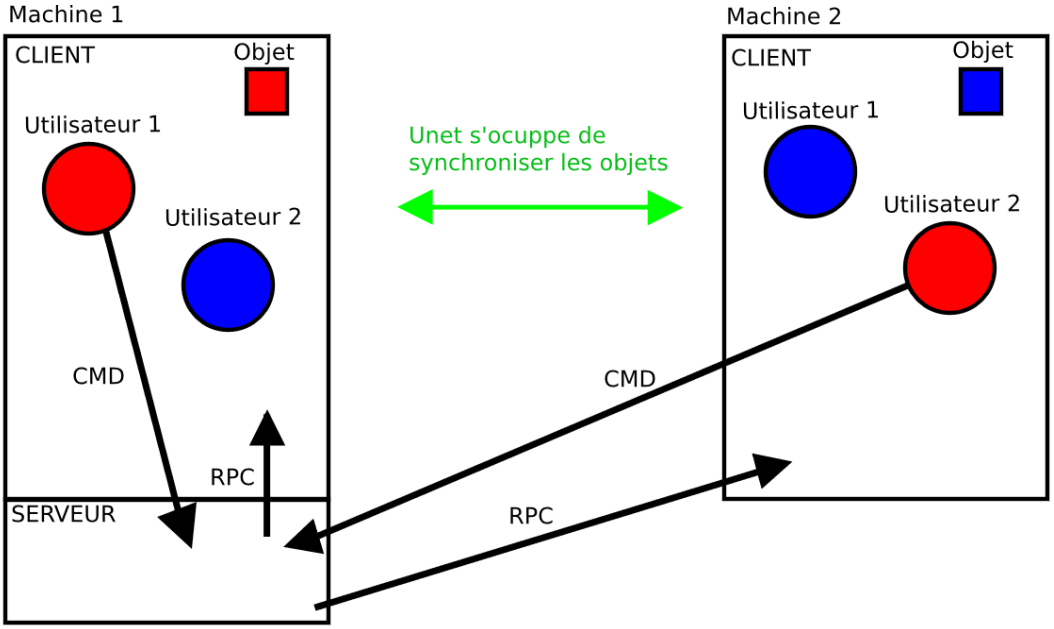


Figure 16 : Fonctionnement simplifié de UNet (en rouge : objets avec autorité local)

UNet utilise aussi le principe "d'autorité" pour savoir qui peut contrôler un objet. "L'autorité" est par défaut côté serveur pour tous les objets, mais il est possible de changer "l'autorité" pour un objet, afin qu'il soit géré par un client. Seuls les objets représentant les utilisateurs ont leur "autorité" côté client local par défaut. *UNet* s'occupe automatiquement de synchroniser la position et la rotation des objets pour chaque client.

3.4.2 Création du Package

Pour le projet, après m'être suffisamment documenté sur *UNet*, j'ai donc commencé par créer un modèle simple permettant de représenter l'utilisateur (sa tête et ses mains) via le réseau. J'ai fait en sorte que ce modèle suive le mouvement côté client de l'utilisateur. *UNet* s'est ensuite automatiquement occupé de synchroniser la position de l'utilisateur entre les différents clients en passant par le serveur.

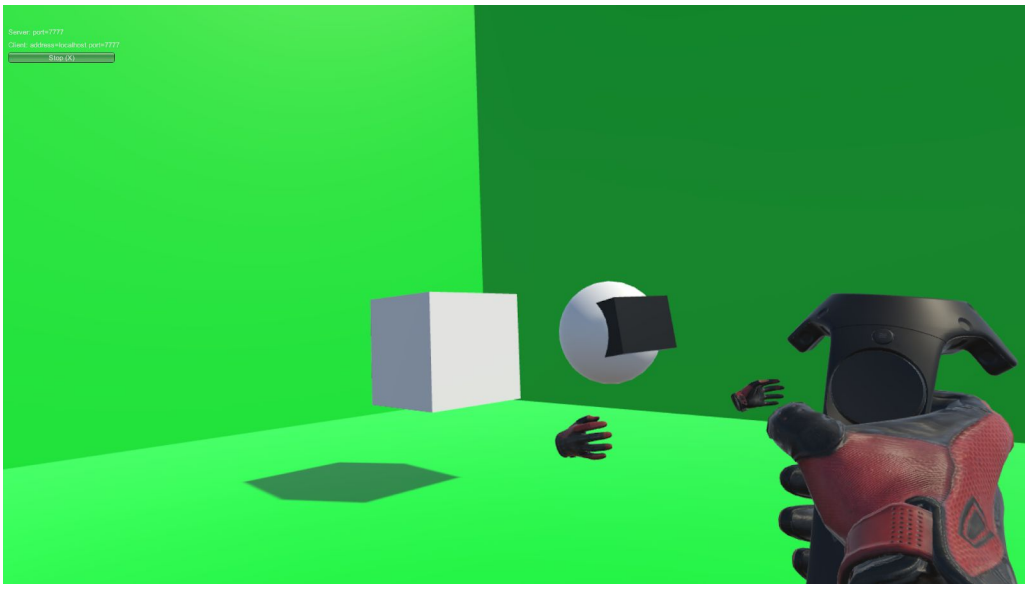


Figure 17 : Utilisation du package multi-utilisateur

Un problème est apparu lors de la mise en place de la possibilité d'attraper des objets. En effet, par défaut chaque objet a son "autorité" sur le serveur. Donc uniquement la machine avec le serveur peut déplacer l'objet. Les autres clients ne peuvent pas déplacer l'objet, car *UNet* synchronise la position par rapport au serveur. La solution a été d'utiliser une méthode CMD pour demander au serveur d'utiliser une méthode RPC afin de demander aux clients de changer "l'autorité" des objets attrapés. Le but étant de faire en sorte que le client ait l'autorité sur cet objet dès qu'il l'attrape.

3.5. Recréation du ShowRoom avec toutes les fonctionnalités

Le but de cet autre projet était de recréer un ancien projet, réalisé par mes encadrants, nommé "showroom" en le rendant multi-utilisateurs et utilisant des librairies plus récentes. Le projet initial est composé d'une salle où on peut via une interface importer des modèles 3D, puis il y a différentes "stations" avec lesquelles on peut effectuer différentes actions sur le modèle. Comme le déplacer, lui changer sa taille, faire un "éclaté" de ces composants, faire une coupe…

Ce projet a posé beaucoup de problèmes et a été le plus long. La première difficulté a été de remplacer tout le code de l'ancienne librairie *VRTK* par la nouvelle *Steam VR* et de faire en sorte qu'elle marche de la même manière. Mais j'ai été grandement aidé par les *packages* que j'avais créé pour les inputs. De la même manière, pour le multi-utilisateur, j'ai utilisé le *package* que j'avais créé, ce qui m'a fait gagner énormément de temps.

La deuxième difficulté a été de pouvoir importer le modèle 3D via le réseau, afin que tous les utilisateurs puissent voir le même objet. En effet comme l'objet est importé pendant l'exécution sur un client, les autres clients n'ont aucune idée de son existence. Il faut donc trouver un moyen pour envoyer l'objet sur tous les clients. Pour cela j'ai sérialisé, en une

série de tableau de bytes, les informations de l'objet, puis j'ai utilisé les méthodes *CMD* et *RPC* pour envoyer ces informations aux clients et enfin j'ai désérialisé les informations afin de reconstruire l'objet à partir de ces informations. Le problème était que cette méthode était assez lente. Une autre solution plus rapide, qui a été mis en place plus tard, a été d'utiliser *TCP* (*Transmission Control Protocol*) pour envoyer les informations au lieu des méthodes de communications *UNet*.

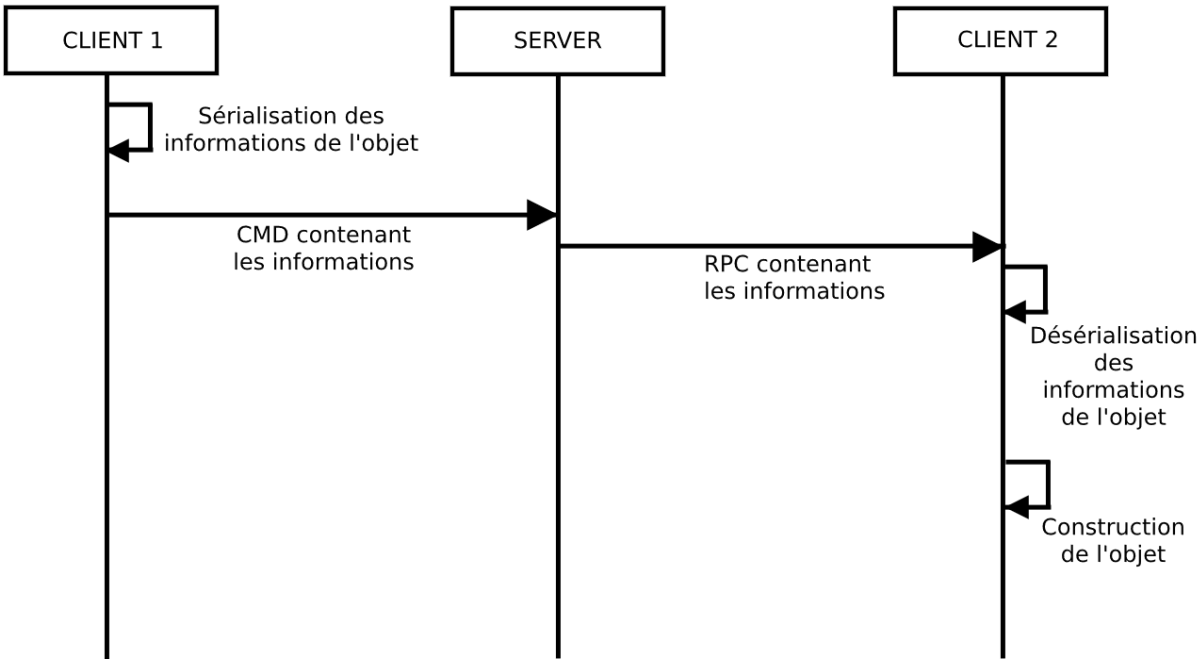


Figure 18 : Diagramme de séquence pour l'import d'un modèle 3D avec UNet

Une troisième difficulté a été la synchronisation de la position du modèle 3D, en effet pour synchroniser UNet associe automatiquement au lancement un ID pour chaque objet que l'on souhaite synchroniser. Or dans notre cas l'objet est créé pendant l'exécution et UNet a aucun moyen de savoir si les deux modèles sont les mêmes. Une solution a été de créer un objet invisible pouvant contenir le futur modèle 3D qui est créé au lancement et possède donc le bon ID. Ce que l'on synchronise, c'est juste cet objet, le modèle 3D se contente lui, de se mettre à la même position que l'objet invisible.

3.6. Scrum pour le projet aménagement d'usine

Le projet d'aménagement d'usine a pour but de réaliser une salle de manipulation de maquettes numériques avancée. C'est un projet que, à l'heure où j'écris est encore en phase de préparation, mais il est déjà intéressant pour l'utilisation des principes de gestions de projets.

Pour cela mes encadrants ont défini plusieurs besoins utilisant tout ce que j'ai pu apprendre pendant le stage ainsi que de nouvelles fonctionnalités :

- Être plusieurs dans une même scène.
- Avoir une interface « Catalogue » qui permettrait de choisir et de placer des objets dans la scène dynamiquement.
- Pouvoir déplacer des objets de deux façons différentes :
    - En l'attrapant dans la scène
    - En étant au-dessus de la scène et pouvoir avec un laser sélectionner et déplacer un élément.
- Pouvoir ajouter des annotations à n'importe quel endroit, que l'on peut cacher ou afficher
- Mesure entre deux points
- Sauvegarder la scène en l'état
- Charger les éléments dans la scène
- Pouvoir filmer ce que l'utilisateur voit en vue de présentation

Puis l'objectif de départ du projet a été de réaliser un Scrum Board pour ces besoins ainsi que de penser *package* pour la réutilisation. Scrum ("mêlée" en Anglais) est une méthode agile très utilisé dans le monde, le Scrum Board est un outil de cette méthode agile. Le but du Scrum Board est de faire la liste des tâches à réaliser dans le cadre d'un projet puis de les classer dans un tableau en fonction de si la tâche est à faire, en cours, en attente ou terminée (Voir la liste des tâches avec l'annexe 4). Il me reste maintenant encore le mois de Juillet en stage pour pouvoir réaliser ce projet.

III. Conclusion

Avec ce stage j'ai pu avoir un aperçu du monde du travail professionnel, en effet en tant que stagiaire j'ai eu l'occasion de participer à des réunions toutes les semaines avec toutes l'équipe. J'ai dû aussi faire des présentations orales devant mes collègues sur les designs pattern (Singleton et État), dans le but de me préparer pour l'oral de stage, mais aussi pour d'autres présentations futures dans le monde professionnel. J'ai pu aussi voir des notions de gestions de projets dans un cas réel dans le cadre de ce stage, notamment avec le Scrum et des Gantt (voir annexes 1 et 2).

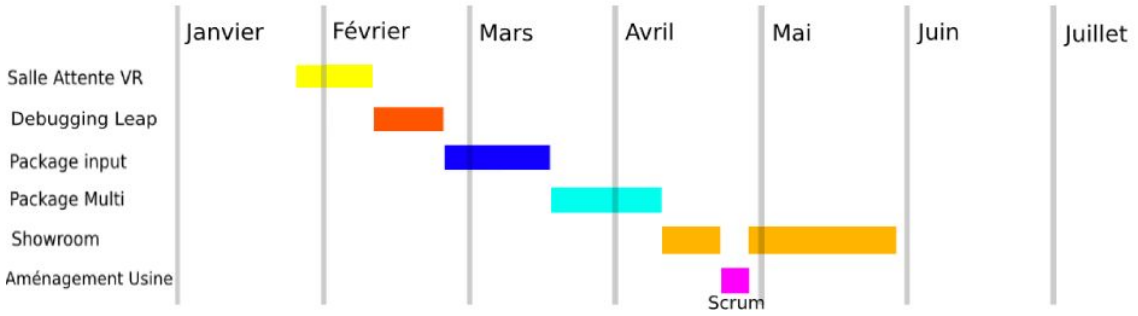
Globalement je suis satisfait de mon travail concernant les différents projets donnés lors de ce stage. Bien que j'aie eu un peu plus de mal que prévu avec la partie du Showroom et que je n'ai pas encore eu le temps de m'attaquer à l'aménagement d'usine, j'ai réussi à réaliser beaucoup d'outils qui seront utiles pour développer la réalité virtuelle plus facilement. Notamment les packages que j'ai créé qui permettent de facilement utiliser les inputs, ou de facilement créer une application multi-utilisateur en réalité virtuelle. Ces packages que j'ai moi-même utilisé pour mon projet de showroom et qui m'ont bien servi. Des clients de *Safran Engineering Services* qui sont passés pour tester la réalité virtuelle, ont même eu

l’occasion de tester mes outils, comme la téléportation *Leap* ou le multi-utilisateurs et en étaient satisfaits.

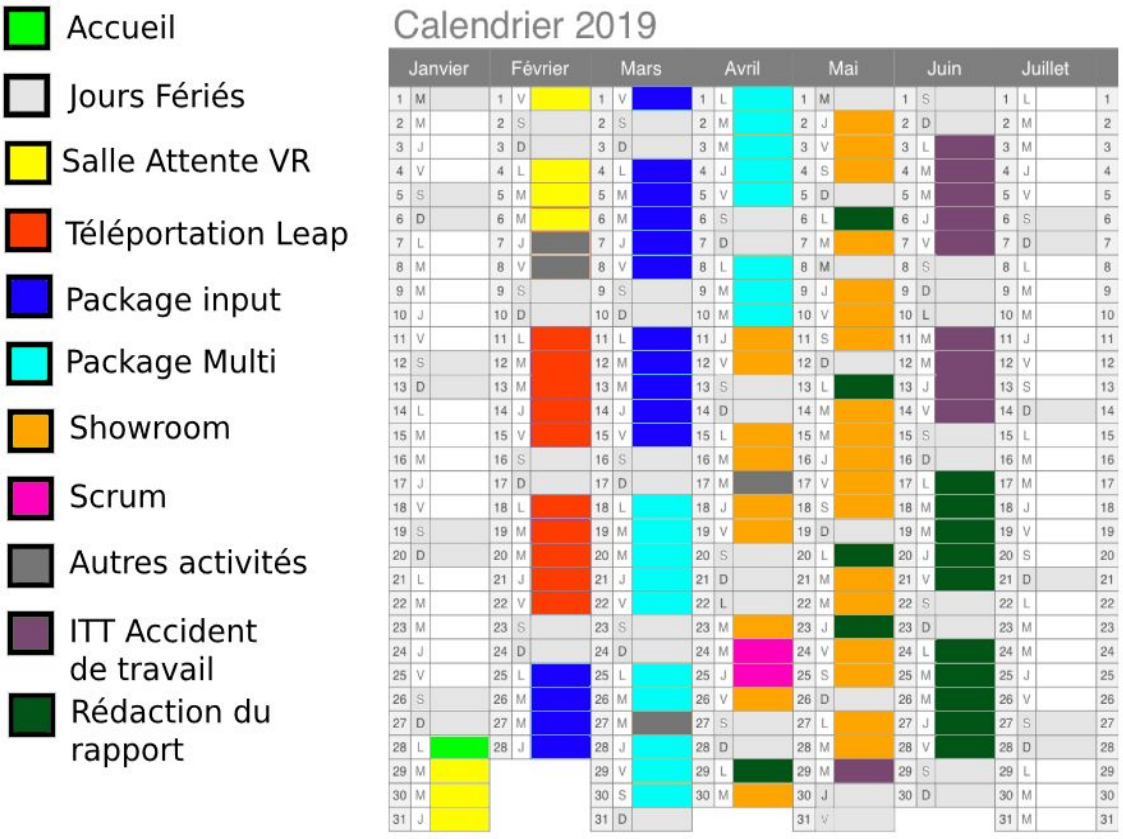
Pour moi ce stage fut vraiment un excellent choix, en effet j’ai eu beaucoup de chance de pouvoir rejoindre une entreprise telle que *Safran Engineering Services*, ce qui m’a permis de programmer avec des outils très high tech comme le *HTC Vive* ou le *Leap*, ainsi que la possibilité de mettre en pratique mes connaissances acquises durant mes cours et d’apprendre tous les jours quelque chose de nouveau, le tout dans un cadre très professionnel.

Annexes

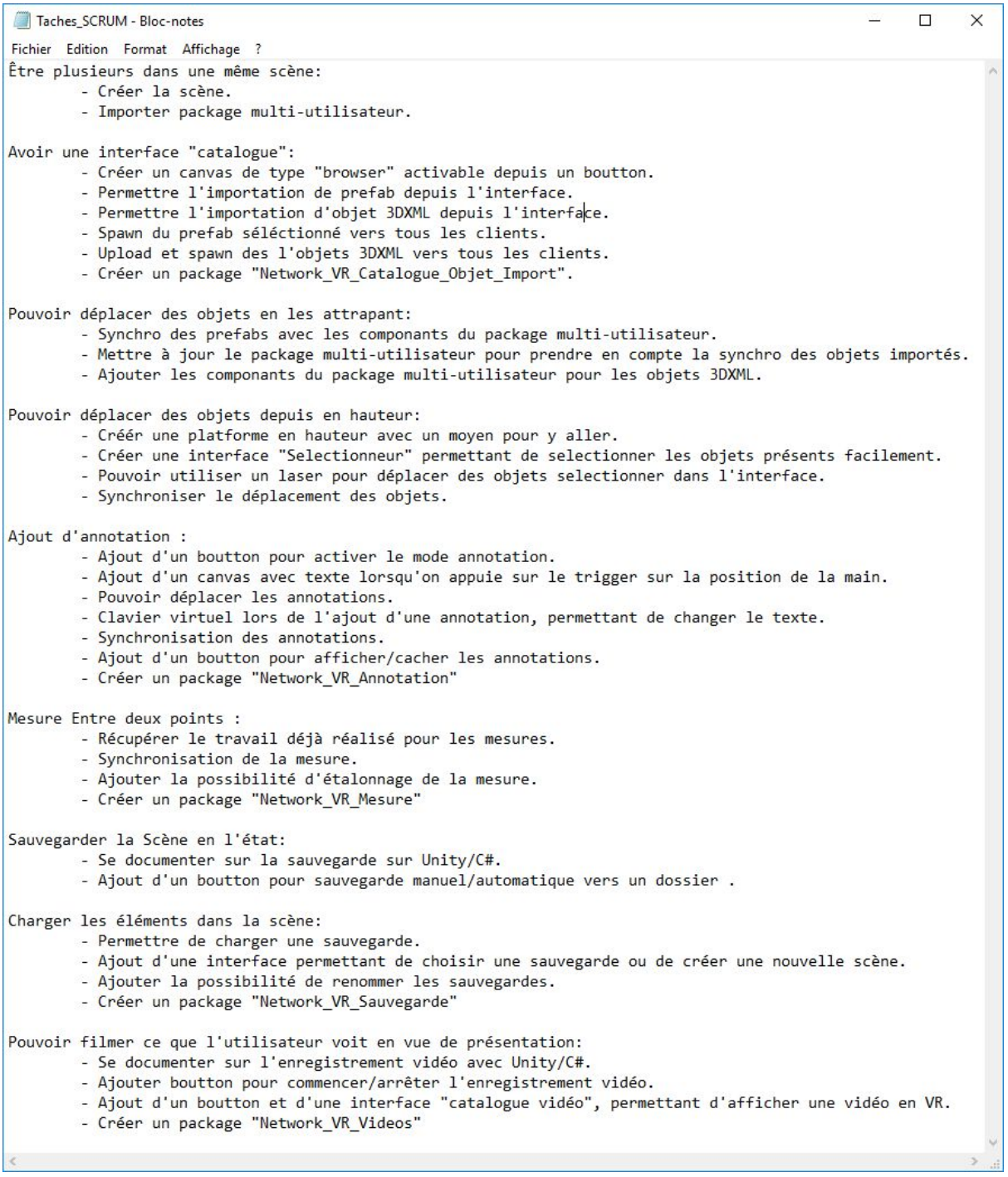
Annexe 1 : Gantt initial Janvier à Juillet



Annexe 2 : Gantt actuel Janvier à Juin



Annexe 3 : Calendrier des activités de Janvier à Juin



Taches_SCRUM - Bloc-notes

Fichier Edition Format Affichage ?

```
Être plusieurs dans une même scène:
        - Créer la scène.
        - Importer package multi-utilisateur.

Avoir une interface "catalogue":
        - Créer un canvas de type "browser" activable depuis un boutton.
        - Permettre l'importation de prefab depuis l'interface.
        - Permettre l'importation d'objet 3DXML depuis l'interface.
        - Spawn du prefab séléctionné vers tous les clients.
        - Upload et spawn des l'objets 3DXML vers tous les clients.
        - Créer un package "Network_VR_Catalogue_Objet_Import".

Pouvoir déplacer des objets en les attrapant:
        - Synchro des prefabs avec les componants du package multi-utilisateur.
        - Mettre à jour le package multi-utilisateur pour prendre en compte la synchro des objets importés.
        - Ajouter les componants du package multi-utilisateur pour les objets 3DXML.

Pouvoir déplacer des objets depuis en hauteur:
        - Créér une platforme en hauteur avec un moyen pour y aller.
        - Créer une interface "Selectionneur" permettant de selectionner les objets présents facilement.
        - Pouvoir utiliser un laser pour déplacer des objets selectionner dans l'interface.
        - Synchroniser le déplacement des objets.

Ajout d'annotation :
        - Ajout d'un boutton pour activer le mode annotation.
        - Ajout d'un canvas avec texte lorsqu'on appuie sur le trigger sur la position de la main.
        - Pouvoir déplacer les annotations.
        - Clavier virtuel lors de l'ajout d'une annotation, permettant de changer le texte.
        - Synchronisation des annotations.
        - Ajout d'un boutton pour afficher/cacher les annotations.
        - Créer un package "Network_VR_Annotation"

Mesure Entre deux points :
        - Récupérer le travail déjà réalisé pour les mesures.
        - Synchronisation de la mesure.
        - Ajouter la possibilité d'étalonnage de la mesure.
        - Créer un package "Network_VR_Mesure"

Sauvegarder la Scène en l'état:
        - Se documenter sur la sauvegarde sur Unity/C#.
        - Ajout d'un boutton pour sauvegarde manuel/automatique vers un dossier .

Charger les éléments dans la scène:
        - Permettre de charger une sauvegarde.
        - Ajout d'une interface permettant de choisir une sauvegarde ou de créer une nouvelle scène.
        - Ajouter la possibilité de renommer les sauvegardes.
        - Créer un package "Network_VR_Sauvegarde"

Pouvoir filmer ce que l'utilisateur voit en vue de présentation:
        - Se documenter sur l'enregistrement vidéo avec Unity/C#.
        - Ajouter boutton pour commencer/arrêter l'enregistrement vidéo.
        - Ajout d'un boutton et d'une interface "catalogue vidéo", permettant d'afficher une vidéo en VR.
        - Créer un package "Network_VR_Videos"
```

Annexe 4 : Liste des tâches Scrum Board