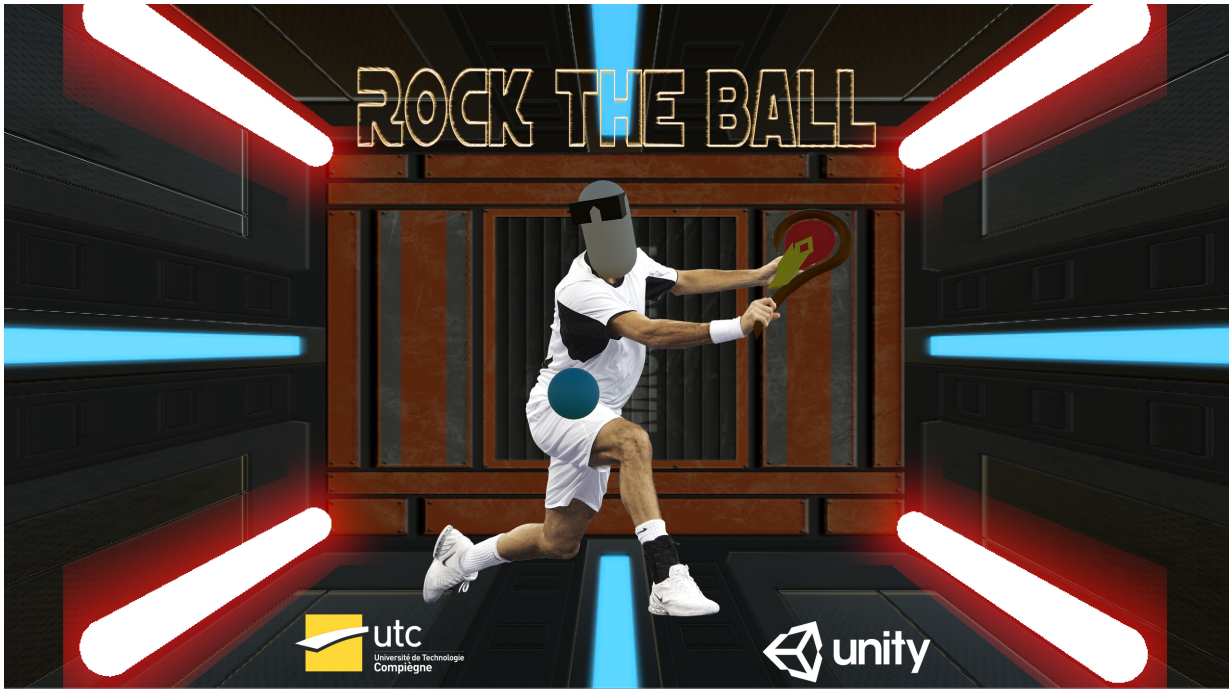

Rock The Ball

Di Vita Rémi - Groisne Louis

Réalisation d'une application de réalité virtuelle

Automne 2017

Encadrants :
Mme. Indira Thouvenin
M. Romain Guyard

utc
Université de Technologie
Compiègne

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Contexte

Dans de la cadre de l'Unité de Valeur RV01 qui initie les étudiants à la Réalité Virtuelle, en mettant l'accent sur l'immersion et les interactions 3D, nous devons réaliser un projet qui nous permettra d'appréhender et de mettre en place un environnement nécessaire à une application de réalité virtuelle.

Ce projet est réalisé par Rémi Di Vita, étudiant en GI05 STRIE, et Louis Groisne, étudiant en GI04 ICSI. Ce projet est encadré par Indira Thouvenin et Romain Guyard.

1.2 Présentation du projet

"Rock the Ball" est un jeu de sport opposant deux joueurs humains en réseau. La zone de jeu a la forme d'un pavé droit à l'intérieur duquel se trouvent les deux joueurs. Les règles de ce sport sont inspirées de Pong transposées dans un univers en trois dimensions. La nécessité de maîtriser les rebonds sur toutes les surfaces du terrain et l'utilisation d'une raquette rappellent également certains aspects du Squash. Le principe de base du jeu consiste à renvoyer une balle à l'aide de ses mains ou de sa tête. Si un joueur laisse passer la balle derrière soit, le joueur adverse marque un point. Les projectiles sont en apesanteur et peuvent rebondir sur les murs, sol et plafond du terrain.

1.3 Objectifs

Les objectifs de ce projet ont été définis dans le cahier des charges que nous avons réalisé pour la première revue de notre projet. Le but essentiel de ce projet est de définir

des interactions et un environnement qui donne lieu à une expérience avant tout sportive. Cela repose principalement sur les dimensions du terrain de jeu virtuel et sur la vitesse de la balle qui contraignent les joueurs à se déplacer rapidement et en permanence pour pouvoir renvoyer la balle.

1.4 Matériel

Le jeu sera développé sur le moteur de jeu Unity. Les joueurs seront chacun équipés d'un HTC Vive et d'un seul Vive Controller. Les deux joueurs seront connectés en réseau grâce aux interfaces réseaux dédiées de Unity : Unity Network. L'espace physique nécessaire pour chaque joueur a une taille correspondante à la taille de leur zone de jeu virtuelle, autrement dit environ 2,5m de largeur sur 3,5m de longueur et 2,5m de hauteur. Le jeu se jouant en réseau, l'espace physique de chaque joueur n'est pas nécessairement adjacent l'un à l'autre.

FIGURE 1.1 – Casque de réalité virtuelle Htc Vive

Chapitre 2

Réalisations

2.1 Environnement visuel et sonore

Les deux joueurs sont projetés dans un environnement de forme rectangulaire aux parois futuristes et éclairé par des néons de couleur rouge et bleu. Cet environnement a une taille d'environ 3m de largeur, sur 10m de longueur, sur 2m50 de hauteur.

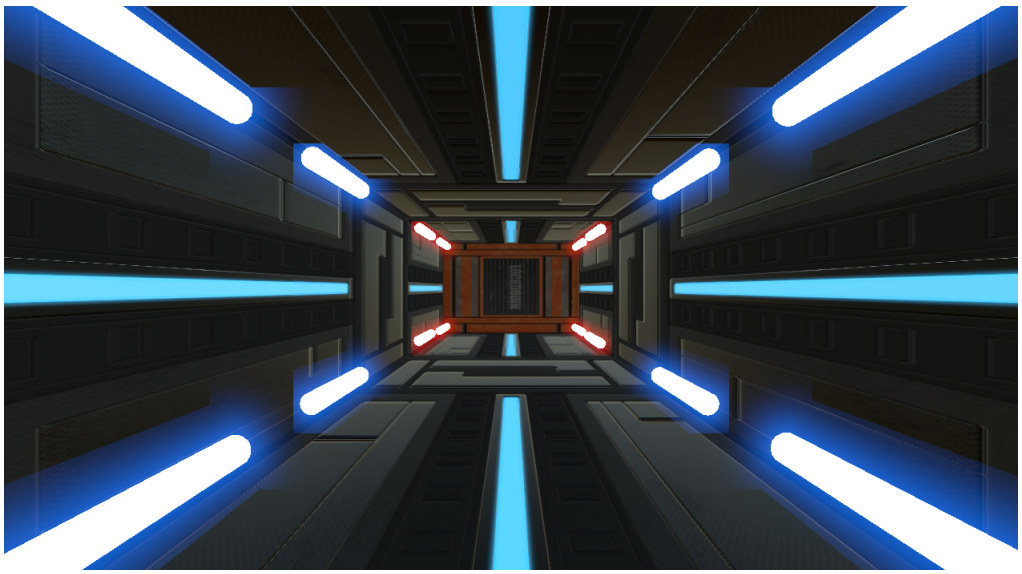

FIGURE 2.1 – Espace de jeu

La musique d'ambiance se caractérise par des sonorités rétros qui se combinent avec l'ambiance visuelle produite par les néons colorés pour rappeler l'inspiration du jeu Pong et la nature Arcade de notre jeu. Seuls la tête et la raquette de chaque joueur sont représentés dans l'environnement virtuel pour que chaque joueur puisse percevoir son adversaire. La tête est représentée par une capsule blanche et la raquette est représentée par un modèle 3D

spécialement réalisé pour ce projet. Le score du match en cours est affiché sur la raquette du joueur. Le score du joueur est affiché en gras et le score de son adversaire est affiché normalement. Ce score est mis à jour à chaque but.

FIGURE 2.2 – Modèle 3D du joueur

Le seul autre élément d'interface du jeu est un texte indiquant la fin du match et le joueur gagnant.

Les sons liés à la balle, à savoir les rebonds sur les murs et les frappes avec une raquette, sont spatialisés. De cette façon, ils permettent de donner aux joueurs une information sur la position de la balle, ce qui leur permet de mieux se situer dans leur espace virtuel.

2.2 Déplacements

Les joueurs peuvent se mouvoir dans l'espace virtuel en se déplaçant dans leur espace physique. Les déplacements physiques sont à l'échelle 1 / 1 par rapport aux déplacements dans l'espace virtuel. Les murs ne produisent aucune collision avec les joueurs. Si un joueur s'approche trop d'un mur, il le traversera simplement. Il aurait été possible de provoquer des collisions entre les murs et les joueurs mais cela aurait induit un décalage entre les déplacements physiques et virtuels. Nous ne voulions pas prendre le risque d'engendrer un motion sickness pour les joueurs, ce choix nous a donc semblé le plus sûr.

Afin de profiter d'une expérience optimale, l'environnement virtuel et l'environnement physique de chaque joueur doivent être en parfaite correspondance pour que les joueurs puissent être libres de leurs mouvements. Autrement dit, si un joueur se trouve au milieu de son espace physique, il doit également se trouver au milieu de sa zone de jeu virtuelle. La

rotation du joueur doit également correspondre entre ses deux espaces. Cette contrainte peut être respectée en procédant à un calibrage rigoureux de l'espace physique utilisé avec SteamVR. Cependant, dans la pratique, le calibrage est trop imprécis. Pour palier ce problèmes, les joueurs ont la possibilité de se téléporter au centre de leur zone de jeu virtuelle en appuyant sur la gâchette du Vive Controller. A tout moment, si la position ou la rotation du joueur sont mal calibrées, le joueur peut se déplacer au centre de son espace physique, se tourner dans la bonne direction et appuyer sur la gâchette pour faire correspondre son espace physique et son espace virtuel.

La zone centrale est une zone neutre du terrain, aucun des deux joueurs ne doit y pénétrer. Lorsqu'un joueur s'approche trop près de cette zone, un filet apparaît pour signaler au joueur qu'il ne doit pas dépasser cette limite. Cependant, aucune collision ne se produit entre le filet et le joueur, pour les mêmes raisons que celles qui ont été évoquées concernant les collisions entre les murs et les joueurs.

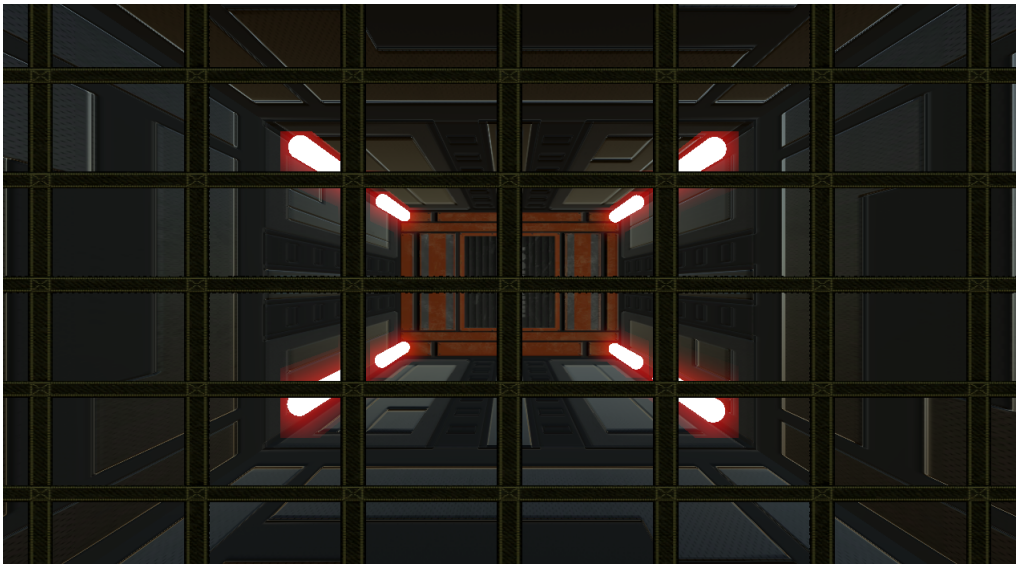

FIGURE 2.3 – Filet au centre du terrain

2.3 Interactions avec la balle

Le but du jeu est de marquer des points en envoyant la balle contre le but de son adversaire. Pour ce faire, le joueur peut frapper la balle à l'aide de sa raquette ou de sa tête. La physique des collisions avec la balle est précise et permet d'envoyer la balle dans la direction dépendante de la direction du geste fait avec la raquette ou la tête. Cela permet aux joueurs d'ajuster leur geste pour envoyer la balle dans la direction souhaitée. L'intensité de la force appliquée par une frappe est également dépendante de la vitesse du geste du

joueur. La vitesse de la balle est cependant limitée pour éviter que la balle n'atteigne une vitesse trop élevée qui la rendrait beaucoup trop difficile à renvoyer. Lorsque la balle rentre en contact avec un but, elle est détruite. Cela déclenche un effet de particules ainsi qu'un son pour signaler au joueur qu'un but vient d'être marqué.

2.4 Déroulement d'une partie

D'un point de vue réseau, l'un des deux joueurs est le serveur (au sens client - serveur) et l'autre est le client. Le joueur serveur est responsable de la continuité de la partie, c'est lui qui fait apparaître la balle au début du match en appuyant sur la gâchette, et qui la fait réapparaître après chaque but. Au début du match, le premier service doit toujours être réalisé par le joueur bleu. Chaque point commence par un service, le joueur qui sert doit envoyer la balle sur la zone affichée en bleu sur un des quatre murs. Chaque fois que le service est faux, le joueur adverse marque un point. A chaque point, le joueur qui sert est celui qui a perdu le point précédent. Le premier joueur à atteindre le score de 15 points gagne le match et un texte est affiché aux deux joueurs pour leur indiquer la fin de la partie.

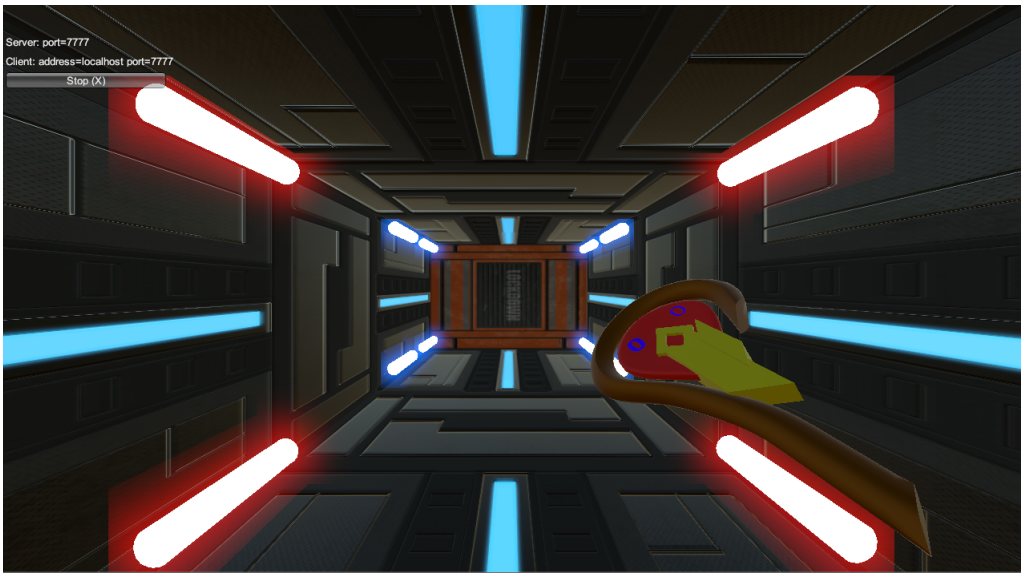

FIGURE 2.4 – Point de vue du joueur lors d'un partie

2.5 Mode entraînement

Le mode entraînement a été ajoutée suite à la deuxième revue de projet. Il a pour but d'introduire l'utilisateur aux différentes mécaniques de jeu et interactions qu'il va devoir

maîtriser pour agir dans l'environnement virtuel. Cet entraînement est découpé en 4 phases élémentaires permettant chacune de décrire une règle ou une interaction. A la fin de chaque étape, un texte invite le joueur à appuyer sur la gâchette du Vive Controller pour passer à l'étape suivante.

Accueil

Le joueur est accueilli par une voix lui indiquant qu'il doit frapper dans la balle qui se trouve devant lui pour commencer l'entraînement. Cette balle porte également une indication "Shoot Me" pour attirer l'attention du joueur sur cet élément.Tant que le joueur n'a pas appuyé sur la gâchette pour passer à l'étape suivante, il a la possibilité de s'entraîner à frapper la balle correctement. Pendant cette première phase d'entraînement, les deux buts sont désactivés et renvoient la balle comme les autres murs pour que la balle revienne toujours vers le joueur.

FIGURE 2.5 – Effet visuel ajouté à la balle lors du mode entraînement

But et incrément du score

Au début de cette étape, une voix indique au joueur de viser le mur en face de lui pour marquer un but. Pendant cette phase, le but du terrain adverse est activé, ce qui permet au joueur de marquer un but. Lorsqu'un but est marqué, la balle est détruite. Un effet de particules et un son sont déclenchés à ce moment pour donner un retour au joueur, lui signalant qu'il vient bien de marquer un but.

Service

Au début de cette étape, une voix indique au joueur le fonctionnement du service pour engager un point et l'invite à essayer de servir correctement. Tant que le service est faux, la balle réapparaît au point de départ. Un son vocal est également déclenché pour indiquer au joueur lorsque le service est faux ou lorsqu'il est réussi. Une fois que le service a été réussi, le joueur peut passer à l'étape suivante.

Entraînement libre

Il s'agit de la dernière phase de l'entraînement. Cette phase permet à l'utilisateur de s'entraîner à frapper la balle correctement et à l'envoyer dans la direction qu'il souhaite. Durant cette phase, les buts de chaque joueurs sont désactivés et la balle rebondit sur le mur comme sur n'importe quel autre.

Chapitre 3

Problèmes rencontrés

3.1 Réseau

Comme évalué dans notre analyse de risques, le réseau nous a posé beaucoup de problèmes liés à la fois à la conception et au développement. N'ayant aucune compétences préalables dans l'utilisation des interfaces réseau de Unity, nous avons suivi le rapide tutoriel "Unity - Multiplayer Networking" (https ://unity3d.com/fr/learn/tutorials/s/multiplayer-networking). Ce tutoriel permet d'acquérir les fondamentaux de la mise en oeuvre du réseau grâce aux interfaces de programmation de Unity : UnityEngine.Networking. Notre manque d'expérience dans ce domaine a été la source des trois problèmes principaux suivants.

Nous avons dû redévelopper à plusieurs reprises certaines parties de notre application en raison d'erreurs de conception

Nous avons dû traiter des problématiques tout à fait différentes de ce que nous connaissions. La nécessité d'avoir une synchronisation permanente entre le client et le serveur soulève beaucoup de problèmes et de questions à se poser systématiquement. La question principale étant de savoir qui a l'autorité pour décider d'un changement dans l'état du jeu. Par exemple, si la balle est frappée par un joueur, est-ce le client, le serveur ou les deux qui doivent appliquer la force nécessaire ? Cette question doit être posée le plus tôt possible pour le développement de chaque fonctionnalité car elle détermine fortement la façon dont la fonctionnalité sera implémentée.

Le réseau est beaucoup plus performant sur des versions compilées de l'application plutôt que sur des versions tournant directement dans l'inspecteur de Unity

Nous nous sommes rendu compte de ce dernier point seulement quelques jours avant la revue de projet finale. Si nous avions eu connaissance de cela plus tôt nous aurions pu faire des choix plus judicieux sur la façon de mettre en place certaines fonctionnalités. En effet, nous avions toujours testé jusque-là via l'inspecteur ce qui produisait donc des résultats biaisés et non représentatifs des performances réelles de notre application. L'exemple le plus représentatif de ce problème est la synchronisation de la position et de la vitesse de la balle côté client. En utilisant l'inspecteur, la synchronisation est beaucoup plus imprécise, ce qui nous a conduit plusieurs fois à prendre de mauvaises décisions sur la façon de gérer cette synchronisation.

Nous n'avons pas mis suffisamment à contribution les composants réseau standards de Unity

Concernant ce dernier point, Unity met à disposition plusieurs composants permettant de répondre à la plupart des besoins de base liés à une application réseau. Cependant nous n'avons pas suffisamment étudié les possibilités offertes par ces composants. Nous avions besoin que les deux joueurs connaissant en permanence l'état actuel de la balle pendant un point. Cet état inclut la position et la vitesse de la balle. Unity possède un composant NetworkTransform permettant de synchroniser la position d'un objet sur le réseau jusqu'à 30 fois par seconde. Cependant, même avec une synchronisation de la position de la balle 30 fois par seconde, la trajectoire de la balle n'était pas fluide, elle était saccadée et très désagréable pour le joueur client. Nous avons décidé à ce moment de développer une solution via des scripts pour résoudre ce problème. Cependant, le composant NetworkTransform possède déjà un attribut "interpolateMovement" qui permet justement, par interpolation linéaire, de lisser la trajectoire d'un objet dont la position est transmise sur le réseau. Nous avons fait l'erreur de ne pas suffisamment chercher la solution du côté des composants standards, ce qui nous aurait pourtant fait gagner beaucoup de temps puisque nous avons mis longtemps à implémenter notre propre solution très élaborée pour un résultat imprécis et une architecture beaucoup plus complexe.

3.2 Physique

Une des interactions principales de notre jeu est le fait de pouvoir frapper dans la balle en utilisant sa tête ou sa raquette.

Les composants standards de Unity que sont les Colliders et les Rigidbodies sont très performants pour gérer des collisions entre des objets ayant une vitesse faible. Cependant la vitesse de la raquette et de la balle est fréquemment très élevée comparée à ce que Unity peut traiter correctement grâce à ces deux composants simples. La raison de cette instabilité est que les objets virtuels n'ont jamais une position continue dans l'espace et dans le temps. Autrement dit un objet o1 en mouvement a une position p1 à la frame f1 et une position p2 à la frame f2. Si un deuxième objet o2 immobile se trouve sur la trajectoire de l'objet o1 entre les frames f1 et f2, deux cas de figure peuvent se produire.

Dans le premier cas, les colliders de o1 et o2 ne sont pas en contact à la frame f1 mais sont en contact à la frame f2. Dans ce premier cas, le moteur physique de Unity détecte la collision correctement. Ce premier cas survient à chaque fois si les vitesses des objets sont suffisamment faibles par rapport à la taille de leur Collider.

Dans le deuxième cas, les colliders de o1 et o2 ne sont pas en contact à la frame f1 et ne sont pas en contact non plus à la frame f2. Dans ce deuxième cas, le moteur physique de Unity ne détecte aucune collision entre les deux objets puisqu'à aucun moment les Colliders ne se sont superposés. La conséquence d'une telle situation est que les deux objets vont se traverser. Ce cas survient très régulièrement lorsque la vitesse des objets est élevées par rapport à leur taille, ce qui induit des écarts très importants dans les positions successives de l'objet à chaque frame.

C'est pourquoi, dans un premier temps, les collisions étaient très instables et la raquette traversait régulièrement la balle lorsque le joueur frappait trop fort avec (vitesse de la raquette élevée). Pour pallier ce problème, nous avons développé un code détecteur et correcteur d'erreur. A chaque frame, on procède à un raycast entre la position actuelle et la position précédente de la raquette. Si le rayon touche la balle, c'est que le moteur physique a raté une collision et que la raquette vient de traverser la balle. Dans ce cas, on applique manuellement une force sur la balle dépendante de la vitesse actuelle de la raquette.

3.3 Logistique et architecture

Ce projet présente des contraintes très handicapantes pour être testé dans des conditions réelles. La nécessité d'utiliser deux Vives en réseau simultanément présente beaucoup de contraintes liées à la mise en place et au réglage du matériel. D'une part, il faut à chaque fois calibrer les casques en début de session et d'autre part lorsque l'on modifie quoi que ce soit il faut le faire sur deux ordinateurs en même temps puis refaire des versions compilées. Nous avons donc perdu beaucoup de temps à tester dans des conditions réelles. Cependant nous aurions pu éviter beaucoup de problèmes en organisant mieux nos débogages si nous avions mieux conçu notre architecture. Nous avons finalement fait beaucoup de tests dans

des conditions réelles en utilisant au moins un Vive et deux pc en réseau pour tester certaines fonctionnalités très spécifiques. Pour pallier ce problème de logistique, il aurait été crucial de découpler autant que possible les fonctionnalités réseau du reste de l'application. Ainsi nous aurions pu déboguer simplement et spécifiquement chaque fonctionnalité indépendamment du réseau. Nous n'aurions eu besoin que d'une seule machine à chaque fois et les bugs auraient été beaucoup plus faciles à identifier et à corriger.

Chapitre 4

Bilan

4.1 Recommandations pour de futurs projets réseau

Malgré les difficultés que nous avons rencontrées avec l'utilisation du réseau à cause de notre inexpérience, les interfaces de programmation réseau proposées par Unity sont simples d'utilisation. Nous aimerions donc proposer quelques recommandations basées sur notre expérience dans ce projet pour faciliter la mise en oeuvre du réseau dans de futurs projets.

— Déboguer en utilisant des versions compilées de l'application pour connaître les vraies performances du réseau dans cette application. L'écart de performances peut être conséquent et avoir un impact déterminant sur le comportement de l'application.

— Chercher le plus possible des solutions du côté des composants réseau standards de Unity qui comportent déjà des possibilités de paramétrages qui les rendent très flexibles.

— Tenir compte de l'imperfection du réseau et ne pas mettre en place une architecture nécessitant que tous les messages réseaux soient traités. Par exemple notre architecture gérait très mal les situations où de nombreuses collisions se produisaient entre la balle et la raquette dans un laps de temps court. Un message est envoyé du serveur au client à chacune de ces collisions et si un de ces messages n'est pas traité correctement par le client la balle aura une trajectoire complètement différente pour ce dernier. Le point critique de cet exemple est la nécessité que chaque message soit traité parfaitement pour préserver la synchronisation. Ce type de contrainte doit impérativement être évité.

— Ne pas tenter de minimiser la quantité d'informations transmises sur le réseau à tout prix, mieux vaut trop de messages que pas assez. Dans la continuité de la recommandation précédente, ce genre d'optimisation conduit à donner beaucoup d'importance à chaque message transmis. Cela peut amener à des applications plus vulnérables

à la perte de messages réseau. Il vaut mieux transmettre plus d'informations que nécessaire quitte à introduire de la redondance pour éviter d'introduire ce genre de vulnérabilités.

— Aller au plus simple pour synchroniser l'état du jeu entre le client et le serveur. Il est plus fiable de transmettre l'état complet d'un objet à l'instant t plutôt que de se limiter à transmettre ses changements d'état. Par exemple, dans le cas de notre balle, il vaut mieux transmettre plusieurs fois par seconde la position de la balle et sa vitesse (état complet) plutôt que se limiter à transmettre une force à appliquer sur la balle chaque fois qu'un joueur touche la balle (changement d'état). Si l'on ne transmet que le changement d'état chaque fois qu'un joueur touche la balle et que ce message est perdu, le client sera directement désynchronisé. Si l'on transmet régulièrement la position et la vitesse et qu'un message se perd, le prochain message arrivant une fraction de seconde après permettra de réparer l'erreur. L'idée générale est qu'un seul message doit permettre de rattraper toute erreur précédente en définissant complètement l'état de l'objet à l'instant t.

4.2 Changements liés à l'expérience utilisateur en réalité virtuelle

Grâce aux résultats de nos différents tests nous avons procédé à plusieurs modifications par rapport à notre conception initiale. Au début chaque joueur avait une raquette dans chaque main. Après les premiers tests, nous nous sommes rendu compte qu'il était fortement contre-intuitif de frapper la balle avec la raquette tenue dans sa mauvaise main et qu' elle n'était presque jamais utilisée, nous avons donc choisi de la retirer simplement. Nous voulions au début définir 8 zones de service : 4 dans le terrain de chaque joueur. Chaque joueur devait viser les zones de service dans le terrain de son adversaire. Nous nous sommes rendu compte qu'il était très difficile de servir correctement et nous avons donc modifié le terrain pour ne définir que 4 zones de service au milieu du terrain. Ces 4 zones sont donc les mêmes pour chaque joueur et sont plus proches d'eux, ce qui les rend beaucoup plus faciles à viser.

Comme présenté dans la section 2.2 ("Déplacements"), nous avons développé la fonctionnalité de relocalisation du joueur pour résoudre les problèmes de calibrage des casques et de l'espace. Mais nous avions également remarqué que les joueurs pouvaient être souvent désorientés dans l'espace virtuel. En effet, celui-ci est fermé et de très petite taille ce qui rend la perte de repères facile. La fonctionnalité de relocalisation permet de corriger ce problème en aidant l'utilisateur à se repositionner correctement dans son espace virtuel.

Nous l'avons également mentionné dans la section 2.2 ("Déplacements"), le filet définissant les limites de la zone centrale ne produit pas de collision avec le joueur mais ce n'était

pas le cas au début. Nous nous sommes rendu compte que le décalage entre les déplacements physiques du joueur et son déplacement dans l'espace virtuel était très dérangeant et désorientant lorsque le joueur entrait en contact avec cette limite, c'est pourquoi nous avons choisi de supprimer ces collisions. Enfin, nous avons reçu la suggestion d'ajouter une spatialisation sur les sons produits pas la balle. En effet, la balle peut avoir une vitesse assez élevée par rapport aux dimensions du terrain et il est possible de la perdre de vue par moment. Dans ces situations, la spatialisation du son que la balle produit apporte une aide précieuse au joueur pour se situer par rapport à la balle.

4.3 Perspectives d'amélioration et conclusion

Malgré tout le travail que nous avons fourni, notre application est loin de ce qui avait été défini dans le cahier des charges initial. Nous avons dû retirer beaucoup de fonctionnalités et d'interactions initialement prévues en raison du temps passé sur les points critiques fondamentaux de notre application que sont la physique et le réseau. Beaucoup d'améliorations en termes d'interactions restent donc possibles pour ce projet, comme la notion de bonus obtenus en touchant des zones spécifiques du terrain et qui étendrait largement la palette d'interactions actuelles.

L'architecture de notre projet est également un gros point d'amélioration possible. Il faudrait créer de nouvelles classes pour découpler entièrement les fonctionnalités réseau du reste de l'application. Ce découplage permettra d'utiliser l'ensemble des possibilités d'interaction de l'application sans utiliser les fonctionnalités de Unity Network. Il sera ensuite possible de modifier l'implémentation du mode entraînement pour que celui-ci soit jouable indépendamment de toute fonctionnalité réseau.

En résumé, la révision de l'architecture ainsi que l'ajout d'un système de bonus permettant d'étendre les possibilités d'interactions pourrait constituer un nouveau projet à part entière. Nous nous tenons bien sûr à disposition si un tel projet devait être réalisé par de futurs étudiants.

En dépit de ces perspectives d'amélioration, l'application que nous avons conçue et développée est entièrement fonctionnelle et offre une expérience utilisateur satisfaisante. Nous n'avons pas pu nous permettre de prendre le temps de réaliser des tests utilisateurs détaillés pour avoir un retour constructif sur l'expérience proposée par notre application. Cependant les tests que nous avons réalisés nous-même nous ont donné entière satisfaction et nous avons réellement pris beaucoup de plaisir à tester notre produit final. L'expérience proposée a bien un caractère sportif dès le moment où l'on maîtrise bien ses frappes et que le jeu prend place dans un espace physique de taille suffisante. En ce qui nous concerne, notre réalisation constitue une bonne preuve de concept de l'idée que nous avions initialement développée dans notre première revue de projet.