Rapport de projet de fin d'étude IMA5 2019/2020

<u>Projet 18</u> : Virtual Reality Old Gaming

Etudiants : Fabien DI NATALE
Ibrahim BEN DHIAB
Encadrants : Laurent GRISONI
Valentin BEAUCHAMP

POLYTECH LILLE

IRCICA

Sommaire

Introduction

Pour notre projet de fin d'étude nous avons choisi le sujet Virtual Reality Old Gaming encadré par Laurent Grisoni et Valentin Beauchamp. Ce projet s'inscrit dans un projet de recherche européen intitulé VR4REHAB développé à l'IRCICA par l'équipe MINT. Il est principalement développé par Valentin Beauchamp et supervisé par Laurent Grisoni. Il a pour but de combiner les jeux sérieux (de l'anglais *serious gaming*) avec la réalité virtuelle et ainsi faciliter la rééducation des personnes atteintes d'un handicap ou d'une incapacité physique, en jouant à des jeux rétro en réalité virtuelle. En effet, un jeu sérieux est une activité qui combine une intention « sérieuse » (de type pédagogique, d'entraînement …) avec des ressorts ludiques. Notre apport dans ce projet est l'implémentation de plusieurs fonctionnalités configurables qui seront décrites tout au long de ce rapport.

I. Définition du projet

La mission de VR4REHAB est qu'un utilisateur puisse réhabiliter une partie de son corps en réalisant des mouvements variés et customisables. Le dispositif doit donc être facilement modifiable et accessible à tous, et pour cela le projet utilise un casque de réalité virtuelle portable, l'Oculus Quest, couplé à un Raspberry Pi configuré en point d'accès Wifi qui s'occupe d'émuler le jeu rétro. L'Oculus Quest et le Raspberry Pi sont des dispositifs peu chers mais également très portatifs, rendant leur utilisation idéale dans le cadre de ce projet. En effet, le Quest est un casque de réalité virtuelle tout-en-un, il fonctionne sous Android et n'a ainsi pas besoin d'être lié à un ordinateur pour faire tourner des jeux. Il s'occupe ainsi de lancer l'application générée avec Unity dans laquelle se trouve la scène virtuelle qui récupère le flux vidéo du jeu rétro envoyé par le Raspberry Pi visible ci-dessous :

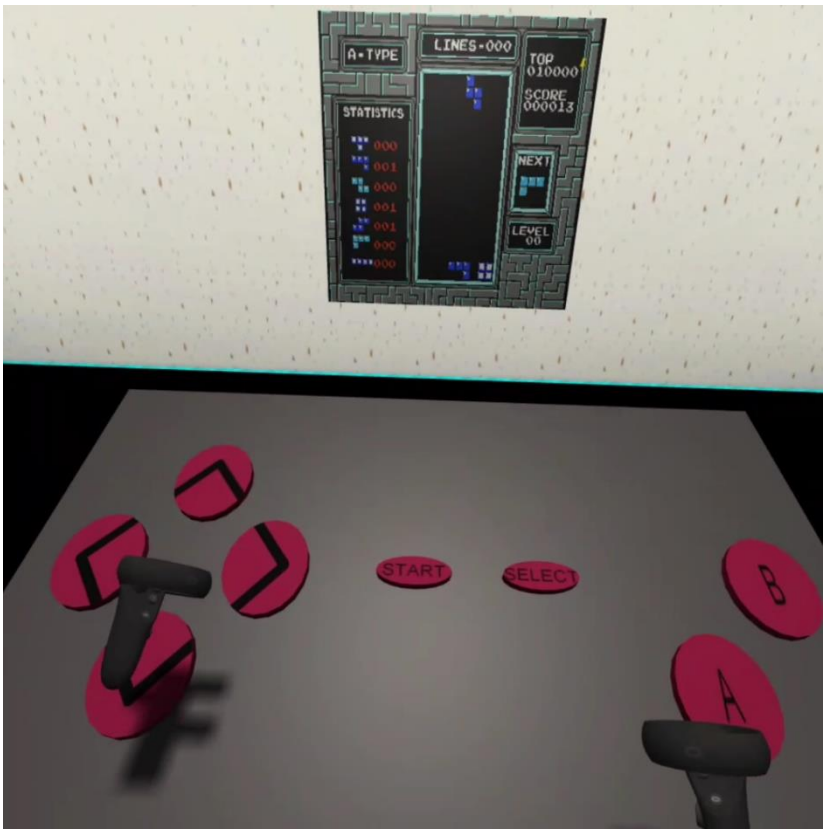

Figure 1 Aperçu de la scène virtuelle dans le casque avec le Game pad et l'écran affichant le flux vidéo du jeu Tetris

Une partie du projet été déjà réalisée, il était fonctionnel mais les seules configurations possibles étaient : changer le nombre de Game pad (manette virtuelle) et lancer une animation de ce Game pad. L'animation permettait de faire bouger le Game pad autour du joueur. Du côté du Quest, l'application est codée sous Unity avec des scripts en C# permettant de contrôler les objets et la scène Unity. Les scripts permettent aussi de gérer le flux vidéo et tous les messages provenant du Raspberry. De plus, ces scripts envoient les commandes correspondant à la pression des touches virtuelles vers Raspberry pour pouvoir effectuer l'action appropriée à la pression de cette touche dans le jeu. Par exemple, sur Tetris, les flèches sur le Game pad déplace les briques à gauche ou à droite, ou les fait descendre plus rapidement, et A et B effectuent une rotation de la brique.

Notre objectif est de créer un maximum de fonctionnalités polyvalentes et facilement utilisables. Ainsi, après une réunion d'information et un brainstorming avec des membres de l'équipe MINT, nous avons défini les objectifs de notre PFE. Pour commencer, nous avons dû créer deux fonctionnalités, la première avait pour objectif de donner la possibilité au thérapeute de modifier l'emplacement des boutons virtuels à l'aide d'une interface web. La seconde fonctionnalité avait pour objectif de faire marcher l'utilisateur, pour cela nous avons eu l'idée d'animer l'écran de jeu afin d'encourager l'utilisateur à le suivre pour pouvoir continuer à jouer.

De plus, une troisième idée de fonctionnalité nous a été donnée, notre tuteur souhaitait que l'on donne la possibilité au patient de jouer grâce aux mouvements des contrôleurs Oculus, ce qui supprimerait les boutons virtuels. Pour cela, il a été proposé d'utiliser la librairie Gina développé par l'équipe MINT.

Enfin, en parallèle de ces tâches, nous devions compléter le serveur web afin qu'il puisse envoyer les configurations nécessaires pour changer les multiples modes de fonctionnement du jeu.

II. La partie Unity

Nous avons choisi de créer les scripts le plus indépendamment possible en utilisant de nouveaux objets afin de ne pas modifier les objets déjà existants. Dans le but de garder le projet le plus polyvalent possible, nous avons choisi de coder dans des nouveaux fichiers tout en séparant chaque fonctionnalité permettant ainsi de rendre nos scripts facilement modifiables.

A. Première option

Pour cette première option que nous intitulons *placement libre*, nous avons commencé par créer un nouvel objet Unity nommé *PadLibre* de la forme d'un Game pad sans le socle afin d'avoir seulement les boutons tout en les gardant reliés autour d'une structure. Ceci nous permet de ne pas modifier l'objet *GamePadObject* déjà présent qui permet d'afficher d'une à trois manettes NES autour du joueur.

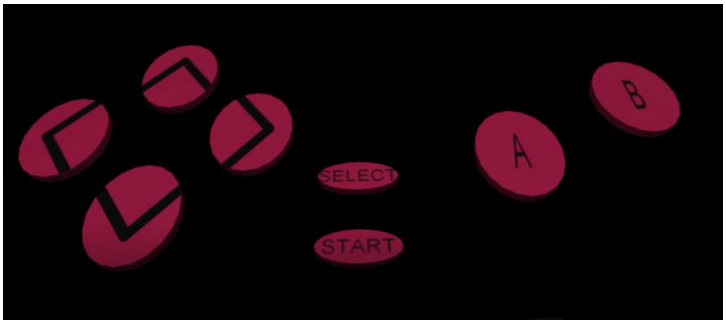

Figure 2 Boutons sans leur socle

Ainsi, une fois notre objet créé nous devons faire un script qui permettra de lire le fichier de configuration et placer les boutons en fonction de celui-ci. Cependant, avant cela, nous avons dû nous mettre d'accord sur la structure du fichier JSON de configuration envoyée par le Raspberry. Nous avons donc choisi la forme suivante, pour chaque bouton on ajoute la ligne suivante avec x, y et z les coordonnées du bouton :

"ButtonA" : {"x" : …, "y" : …, "z" : …}

Ensuite, nous avons pu nous lancer dans la programmation du script. Nous avons commencé par créer la fonction de lecture du fichier de configuration JSON. Nous avons continué par le placement des boutons, pour que celui-ci soit bon nous avons dû faire plusieurs tests afin de définir les limites du placement libre et ainsi éviter que le thérapeute place des boutons hors de portée de l'utilisateur. Nous avons ainsi ajouté en commentaire les limites de placement et les avons configurées sur la partie web.

Une fois cela fait il nous restait à envoyer les commandes correspondant aux boutons appuyés à la Raspberry. Pour cela nous avons repris la fonction utilisée pour l'envoi des commandes dans l'objet *GamePadObject*. A la fin de la mise en place de cette option nous avions compris comment les scripts fonctionnaient et comment codé en C#.

B. Deuxième option

Pour la deuxième option, nous devons commencer par identifier les fonctions que ce script doit avoir avant de nous lancer. Afin de créer cette seconde option le script doit être capable de faire bouger l'écran de façon dynamique, c'est à dire que l'écran ne peut pas disparaître d'un point pour réapparaître 5 mètres plus loin. Nous devons pouvoir gérer la vitesse de mouvement de l'écran et limiter ses déplacements selon l'envie du praticien et donc pouvoir rendre cela configurable par le praticien. Du côté du pad (la manette NES dans le jeu) nous devons faire en sorte qu'elle suive le joueur tout au long de la partie pour ainsi pouvoir laisser le joueur libre de ses déplacements.

Nous divisons alors cette deuxième option en deux parties et donc deux scripts, le premier permettra de faire bouger l'écran et le second permettra de garder la manette fixe par rapport à la vue du joueur.

Pour la première partie nous avons créé quatre fonctions, la première permet de démarrer le mouvement tout en sauvegardant la position de base de l'écran. La seconde permet d'arrêter le mouvement et remettre l'écran à sa position de base (la position enregistrée lors du lancement). La troisième permet de traduire une vitesse en mètres par seconde et une direction en mouvement. Et la dernière fonction (de protection) permet de changer la direction si l'objet sort des délimitations entrées en argument de la fonction. Ces deux dernières fonctions sont regroupées en une seule car le mouvement implémenter est simple mais ceci sera surement la structure finale de ce script. Nous avons ainsi un écran qui fait des aller-retours sur la distance entrée dans la fonction de protection et la vitesse entré dans la fonction de mouvement.

Une fois cela fait nous passons à la seconde partie (le second script). Pour ce second script nous prenons la même formation que le premier script. Cette fois ci, nous avons seulement trois fonctions, la première est la fonction de lancement permettant d'activer la manette et le suivi. La seconde permet d'arrêter le suivi et la dernière permet de replacer la manette et la faire pivoter à chaque image.

Avec les tests nous avons vite remarqué que la manette utilisée était trop grande et nous donnait envie de tourner la tête pour cliquer sur les boutons au bord de la manette faisant alors bouger la manette rendant les boutons sur le côté très difficile à atteindre. Nous avons alors fait certains changements dans les deux premières fonctions permettant l'activation et la désactivation de la manette. Nous déplaçons alors les boutons vers le milieu de la manette en fonction de leur positionnement. C'est à dire que plus le bouton est sur le côté et plus il est ramené vers le centre et inversement, plus il est vers le centre et moins il bouge. De plus, si cela ne convient pas aux médecins, nous pouvons augmenter le pourcentage de déplacement ou réduire le *scale* de l'objet ce qui revient à réduire l'objet dans sa globalité et ainsi rapproché les boutons mais aussi les rendre plus petit. Cette solution a été laissée en commentaire afin de donner la possibilité de l'utiliser si nécessaire.

C. Troisième option

Pour cette troisième option qui est d'utiliser les mouvements de la manette pour contrôler le jeu, nous avons été recommander tout d'abord d'utiliser la librairie Gina développé par l'équipe MINT. Cette librairie a pour but de créer des événements à partir de mouvements, pour cela elle se découpe en trois blocs. Le premier bloc est le décodeur, il prend en entrée un flux d'information brut venant d'un matériel quelconque (une Kinect, une souris, le positionnement du doigt sur une tablette etc…) et le traduit en information utilisable par les autres blocs. Ce bloc permet d'uniformiser l'information afin de limiter le nombre de bloc suivant et surtout permettre l'utilisation de plusieurs sources d'entrées en même temps. Le second bloc est un filtre, il va permettre de filtrer les mouvements entrant afin de supprimer les tremblements et autres défauts que peut avoir l'information afin d'avoir des mouvements les plus propres et les plus facilement exploitables possibles. Le troisième et dernier bloc permet de créer des évènements à partir des informations reçues et ainsi interagir avec le système.

Cette librairie est très complète et pratique, cependant pour qu'un objet soit utilisable par la librairie, elle doit posséder le premier bloc permettant de convertir l'information brute de l'objet en information exploitable par les autres blocs de la librairie. Malheureusement, le bloc pour Android n'était pas implémenté. Cette librairie est donc inutilisable pour notre projet sans la modifier. Sous les conseils du développeur de la librairie et de notre tuteur nous n'allons pas créer le bloc de traduction et nous allons donc utiliser les fonctions proposées directement par Oculus et Unity pour arriver à jouer juste à partir des mouvements de la manette de l'Oculus et donc supprimer la manette virtuelle.

Nous devons alors coder cette troisième option à partir de rien. Nous avons commencé par choisir la structure que nous allons utiliser dans notre script. La structure que nous avons choisie d'utiliser est la même structure que celle de la librairie Gina, une fonction qui détecte et récupère les mouvements, une fonction qui traite les mouvements et une fonction qui traduit le mouvement en évènement afin d'envoyer les informations au Raspberry (le jeu). Pour la fonction qui récupère les mouvements, nous avons commencé par chercher quand et comment récupérer les mouvements afin que cela soit le plus rapide possible et le plus efficace possible. Afin d'éviter la surcharge du flux de données et surtout pour éviter que des mouvements non souhaités soient pris en compte par le jeu. Nous avons décidé de prendre en compte les mouvements des manettes uniquement quand le joueur appuie sur une des deux gâchettes. Nous avons donc dû chercher les différentes méthodes permettant de récupérer la position des manettes et l'état des boutons de la manette, en particulier l'état de la gâchette. Les méthodes sont les suivantes :

- *Input.GetAxis("Oculus_CrossPlatform_PrimaryIndexTrigger")* nous donnant la puissance de pression sur la gâchette gauche.
- *Input.GetAxis("Oculus_CrossPlatform_SecondaryIndexTrigger")* nous donnant la puissance de pression sur la gâchette droite.
- *OVRInput.GetLocalControllerPosition(OVRInput.Controller.LTouch)* nous donnant la position de la manette gauche.
- *OVRInput.GetLocalControllerPosition(OVRInput.Controller.RTouch)* nous donnant la position de la manette droite.

Nous avons maintenant les fonctions principales nous permettant de gérer la récupération d'informations, nous stockons alors toutes ces informations dans une liste dont nous limitons la capacité à 60*2*2, 60 pour le nombre de rafraîchissements par seconde, 2 car il y a deux écrans dans le casque (pour chaque œil) et le second 2 car on veut conserver l'action des deux secondes précédentes au maximum. Une fois que la position a été enregistrée, nous passons la liste à travers le filtre 1€. Ce filtre est développé par l'INRIA en collaboration avec l'université de Waterloo et a été amélioré au fil du temps afin de le rendre accessible dans un grand nombre de langages de programmation. Nous réinitialisons ce filtre à chaque fois que les gâchettes sont lâchées afin de ne pas confondre un arrêt d'enregistrement des positions avec un mouvement. Une fois cela terminé, nous sommes passés à la partie traitement des données, elle se démarque en deux parties, la première a pour but de délimiter le mouvement et la seconde a pour but de récupérer les propriétés du mouvement pour les envoyer à la partie de l'interpréteur.

Pour délimiter les mouvements, la première partie doit vérifier s'il y a eu un mouvement pendant un arrêt d'enregistrement et doit rassembler les dernières données correspondant au même mouvement. Donc pour détecter un mouvement pendant un arrêt d'enregistrement, nous avons simplement mis une condition afin de, si la manette se déplace trop entre deux prises de mesures, considérer cela un autre mouvement. Pour rassembler les points enregistrés correspondant au même mouvement, nous utilisons des vecteurs, nous partons du dernier vecteur enregistré et nous remontons la liste en vérifiant s'il n'y a pas trop de changements entre le vecteur précédent et le vecteur de référence. Le vecteur de référence est la moyenne des vecteurs que nous considérons déjà dans le mouvement donc il est mis à jour après chaque vecteur analysé et considéré comme appartenant au mouvement. La seconde partie du traitement des données enregistre la somme totale des déplacements de la manette selon les 3 axes.

Une fois les données traitées, nous passons à l'interprétation du mouvement détecté, dans ce bloc nous regardons si le mouvement a été assez ample et sans trop de perturbation. Si un des mouvements est détecté, nous activons un flag, enregistrons le mouvement détecté afin qu'une autre fonction s'occupe de l'envoi correspondant au mouvement détecté vers le Raspberry. Notre script peut reconnaître seulement les mouvements rectilignes selon les axes mais pourra être amélioré par la suite.

III. La partie Serveur web

A. Page initiale

Les outils principaux appréhendés et utilisés pendant cette partie ont été Node.js, javascript, jQuery, PixiJS, et Bootstrap. Node.js est utilisé comme moteur javascript pour pouvoir gérer les données efficacement et en temps réel, l'architecture du système est la suivante :

- 1 serveur websocket (framework Express) pour la page web de configuration
- 1 serveur TCP qui communique cette configuration au casque
- 2 serveurs TCP qui gèrent et envoient le flux vidéo de l'émulateur au casque
- 1 serveur websocket pour la page web de contrôle à distance du jeu
- 1 serveur TCP qui reçoit les entrées du Game pad virtuel pour modifier l'état du jeu

Ci-dessous se trouve les pages web qui avaient déjà été réalisées :

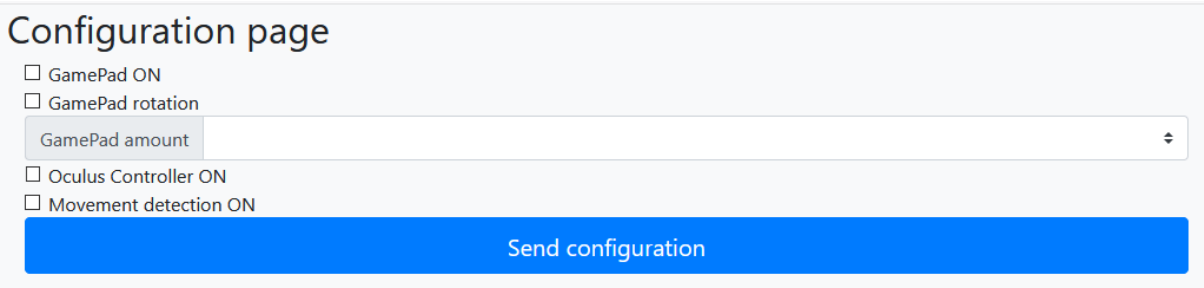

Figure 3 Page de configuration initiale

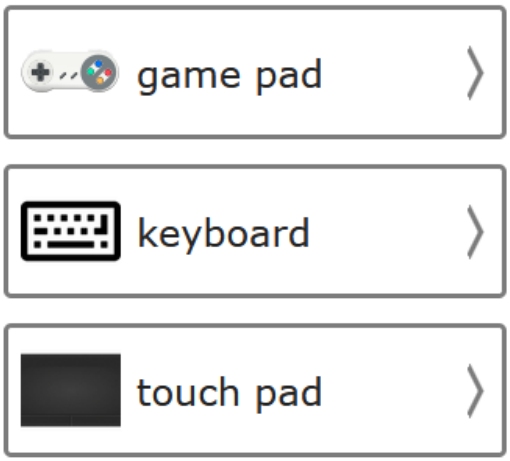

Figure 4 Page de contrôle à distance

Bootstrap est un framework permettant de créer des sites web facilement et rapidement, et contient des modèles en HTML et CSS. Son principal avantage est qu'il soit bien adapté pour les téléphones mobiles. La page web ainsi que le script que nous avons modifié sont ceux de configuration. La page de configuration initiale fonctionne à l'aide d'un fichier JSON de configuration, donc sur la page web, lorsqu'un paramètre est coché et le bouton *Send configuration* est appuyé, ce JSON est modifié (script *index.js*) et le serveur TCP l'envoi au casque, qui est ensuite interprété par les scripts présents dans l'application pour activer les différents modes de jeu.

Les différentes possibilités de configuration initiales étaient :

- *gamepadOn* : activation ou non du Game pad
- *oculusControllerOn* : activation ou non des manettes oculus
- *gamepadAnimation* : animation du Game pad
- *nbGamePad* : nombre de Game pads (entre 1 et 3 manettes virtuelles dans le casque)

L'arborescence finale des fichiers du Raspberry Pi est ci-contre, et les fichiers modifiés ont été *websocket-relay.js* (/root), *index.js* (/public/js), *canvasLayout.js*, *canvasPixi.js* et *index.html* (/public). Le script ***websocket-relay.js*** lance les quatre premiers serveurs abordés plus haut, mais seul celui de configuration, le serveur websocket, a été modifié. Les autres fichiers correspondent à la page web, côté client. Les 3 scripts côté client ont été séparés pour alléger les fichiers et les rendre plus lisible. Le lancement du script Node *websocket-relay.js* est géré par le manager de processus **PM2** qui permet de monitorer les scripts et leurs logs et de les lancer au démarrage du Raspberry.
Après la réunion avec quelques membres de l'équipe MINT, il a été décidé de mettre en place une interface web sur laquelle l'utilisateur pourrait définir soit même la position des touches en jeu. Puis au fur et à mesure l'interface web a été complétée avec les autres fonctionnalités présentées dans la partie Unity.

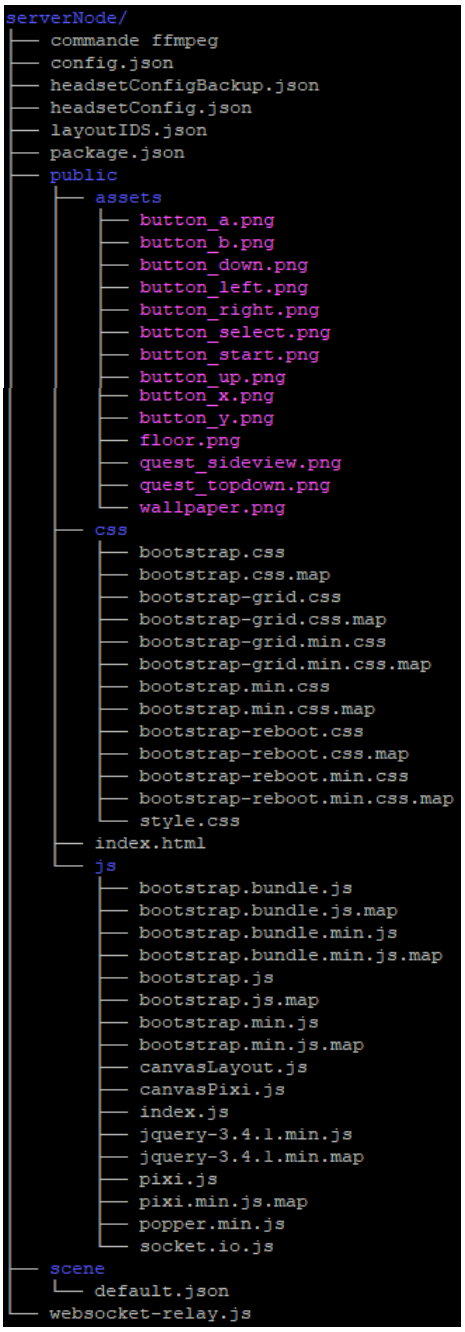

Figure 5 Arborescence du Raspberry Pi

B. Fenêtres Pixi

Valentin Beauchamp nous a conseillé d'utiliser la librairie **PixiJS** qui permet de créer du contenu digital à l'aide d'un moteur de rendu 2D WebGL pour implémenter l'interface de contrôle des touches. A l'aide de l'exemple *Dragging* fourni sur leur site web, nous avons pu implémenter cette interface sur laquelle nous pourrions placer les différents boutons du Game pad et interagir avec à l'aide d'interactions drag and drop. Le script créé qui s'occupe de la génération des modules PixiJS est ***canvasPixi.js***, dedans sont initialisées deux applications ou *canvas*, qui sont des zones définies pour générer les objets 2D. On les a liés à un style CSS nommé **#frame** pour pouvoir les manipuler facilement dans le code HTML de la page web. La première application nommée **app** est utilisée pour placer les boutons sur les axes x et y, tandis que la seconde application **app2** s'occupe de l'axe z, on peut ainsi déplacer les boutons dans l'espace de jeu virtuel en temps réel. Le rendu final est visible ci-dessous :

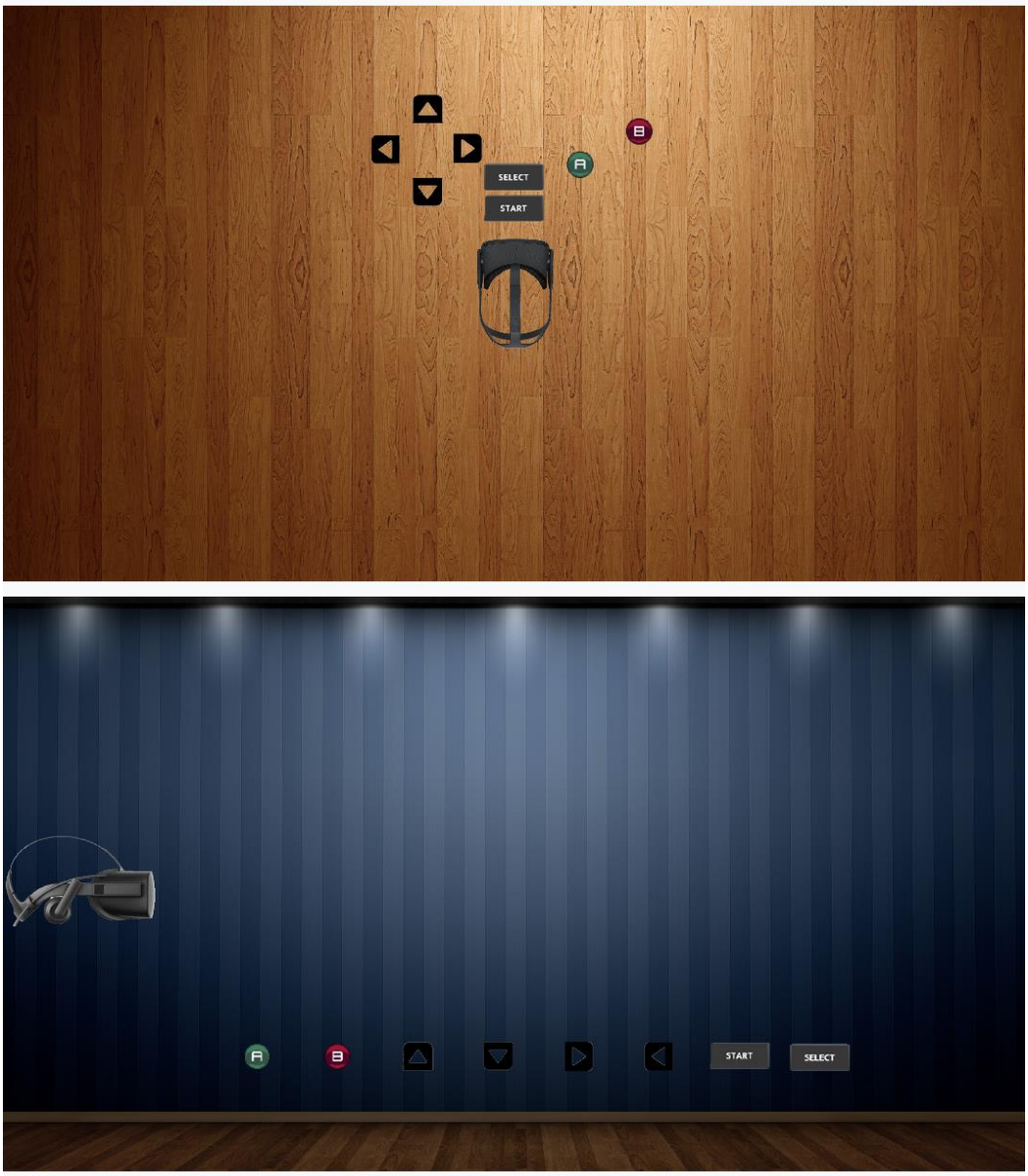

Figure 6 Aperçu des fenêtres de placement des boutons générées par Pixi

Nous avons également fait en sorte que les fenêtres ainsi que tous leurs objets soient redimensionnées lorsque la page web change de taille, cela n'était pas fait automatiquement.

C. Manipulation des fenêtres

Cette partie est consacrée au script ***canvasLayout.js***, qui s'occupe des différentes manipulations en relation avec les fenêtres réalisables sur le site web. Tout d'abord, nous avons mis en place un moyen pour l'utilisateur d'enregistrer des noms (*layout ID*) et d'attribuer à ces ID un placement des touches. Pour cela, une fois un ID créé, il faut le sélectionner dans le menu déroulant *Layout ID*, puis placer les boutons sur les *canvas* comme souhaité, et appuyé sur *Save layout*. Le menu *Default Layouts* permet d'assister l'utilisateur lors de la création d'un placement des touches, pour le moment deux ont été créés, *Normal* et *Higher elevation*, le premier réinitialise simplement les positions à leur état initial, le second permet d'élever toutes les touches sur la seconde fenêtre à hauteur de casque. Cela a été codé de manière à rendre l'ajout d'autre *Default Layouts* simple. Les ID et leurs configurations sont enregistrées dans un fichier *layoutIDS.json* du côté serveur (Raspberry), ainsi, dès qu'un utilisateur accède à la page, une requête est effectuée au serveur pour récupérer les données du fichier, puis remplir le menu déroulant avec.

Le script s'occupe également de cacher la partie de configuration des *layouts* lorsque la case *Enable custom layout* est décochée, et l'affiche quand elle est cochée. Cela rend l'interface utilisateur plus compréhensible. Et pour éviter de créer des configurations impossibles, les différentes options non compatibles avec celle sélectionnée sont grisée, pour éviter de les cocher.

Enfin, quand la page est ouverte, c'est la configuration actuellement dans le casque, donc la dernière envoyée, qui est affichée, si, par exemple, la dernière configuration était le placement libre activé avec un certain ID, alors la case *Enable custom layout* sera cochée et cet ID sera déjà sélectionné. Ci-dessous se trouve un aperçu de la page web sur navigateur, la partie avec les fenêtres est en-dessous du bouton *Reset layout* mais a été tronquée (Figure 6) :

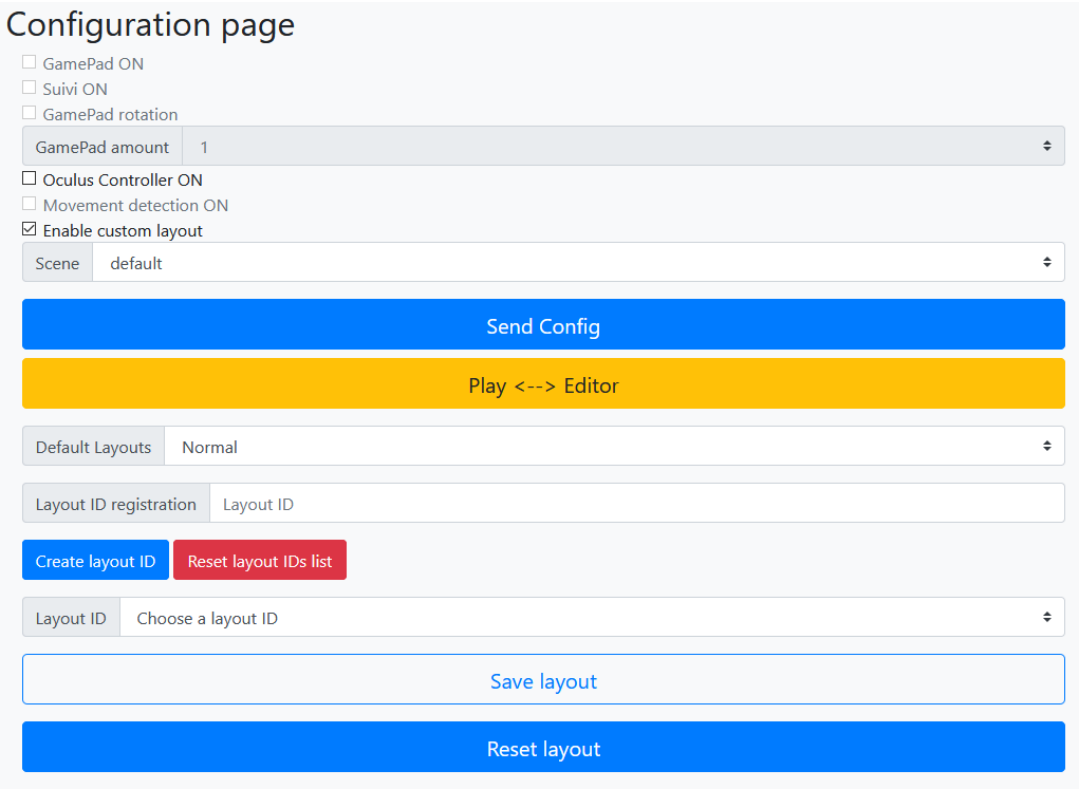

Figure 7 Aperçu de la page web finale (sans canvas)

Conclusion

Nous avons été heureux de pouvoir participer au développement d'un tel projet, il nous a permis d'améliorer nos connaissances en Unity, C# et front-end/back-end en Javascript, et de découvrir l'étendu d'un projet de recherche. Après avoir terminé notre travail, nous avons pu le combiner avec celui de Valentin Beauchamp et de François Brassart, pour obtenir un produit avec toutes les fonctionnalités. Avec les fonctionnalités actuelles, il est possible de jouer aux jeux de plusieurs manières, et de permettre à l'utilisateur de solliciter différents mouvements pour faciliter sa rééducation. L'implémentation future d'autres fonctionnalités est envisageable, et dépendra des besoins du client.