POLYTECH LILLE

Université de Lille

IRCICA

PROJET DE FIN D'ETUDES
P17
Outil d'analyse de mouvements d'interaction

JANVIER-FEVRIER 2020

ETUDIANT : François Brassart

ENCADRANTS : Laurent Grisoni & Valentin Beauchamp

Remerciements

Je tiens tout d'abord à remercier Monsieur Laurent Grisoni pour avoir proposé ce sujet de projet et m'avoir fait confiance pour le réaliser.

Je remercie grandement Monsieur Valentin Beauchamp, membre de l'équipe MINT qui travaille sur le projet VR4REHAB, pour son aide et ses conseils.

Enfin, je remercie toutes les personnes que j'ai pu rencontrer à l'IRCICA et avec qui j'ai pu échanger.

Sommaire

Introduction

Dans le cadre de ma dernière année de cycle ingénieur Informatique, Microélectronique, Automatique, je suis amené à réaliser un projet de fin d'études. Ce projet a pour objectif la mise en œuvre des compétences acquises tout au long de la formation ainsi que l'apprentissage de nouvelles.

Etant parti au Canada pour un semestre d'études de septembre à décembre 2019, j'ai réalisé ce projet en janvier et février 2020. Ce rapport a pour but de présenter le travail que j'ai effectué durant cette période. En début d'année, j'ai donc choisi mon sujet, proposé par Laurent Grisoni dans le cadre de ses travaux de recherche avec l'équipe MINT de l'IRCICA (Institut de Recherche sur les Composants logiciels et matériels pour l'Information et la Communication Avancée).

1. Contexte et cahier des charges

1.1. Projet VR4REHAB

L'équipe MINT de l'IRCICA travaille actuellement sur le projet intitulé « VR4REHAB », acronyme pour « réalité virtuelle pour la rééducation ». L'objectif de ce projet est donc l'utilisation de la réalité virtuelle à des fins de réadaptation. En effet, il a été montré que la réalité virtuelle peut aider à optimiser les protocoles de réadaptation, à accélérer le rétablissement des patients, à motiver le patient dans sa rééducation et à faciliter la réintégration dans la vie quotidienne. Les patients cibles sont principalement des enfants touchés par des maladies chroniques et des handicaps ainsi que des adultes frappés de fatigue et douleurs musculaires.

1.2. Prototype existant

L'équipe a déjà commencé le développement d'un prototype. Le patient porte un casque virtuel et joue à un jeu 2D (de type tetris par exemple), diffusé sur un écran dans la scène virtuelle. Devant lui est disposée une manette de console de type NES. Le patient doit donc interagir avec le jeu grâce à cette manette. Le médecin peut, en direct, modifier la configuration du jeu : nombre de manettes virtuelles dont dispose le joueur, mouvement des manettes dans l'espace, dans le but de compliquer ou faciliter le jeu et de faire plus ou moins travailler le patient.

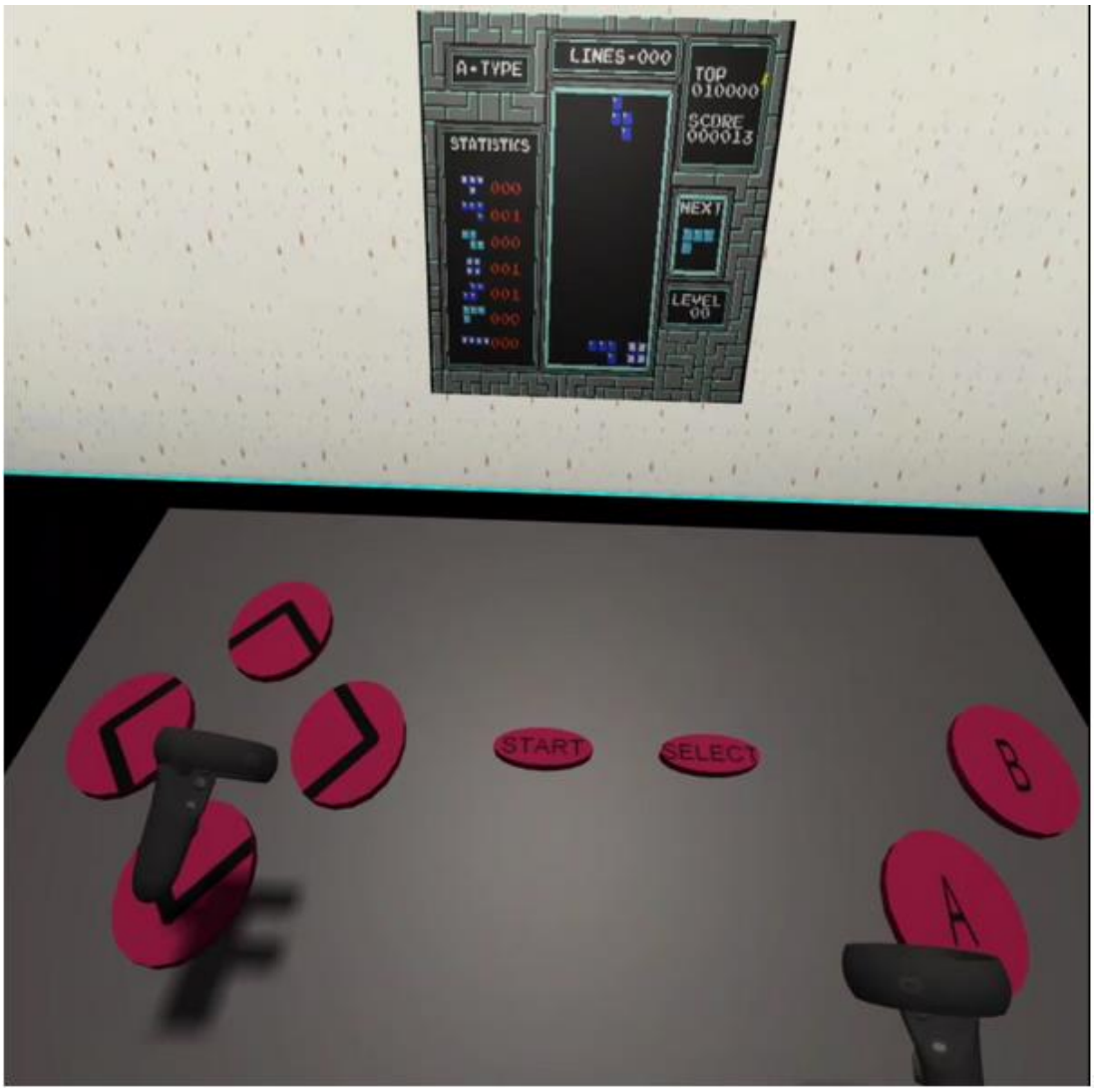


Figure 1 Vue de la scène virtuelle dans le casque

Dans l'objectif de présenter ce prototype à des thérapeutes, il est important de leur montrer toutes les possibilités qu'offre ce dispositif. Ainsi, mon objectif sur ce projet est de créer une interface web permettant au médecin d'avoir accès aux « performances » du patient. Concrètement, il pourra consulter les enregistrements des mouvements du joueur, afin d'en faire une analyse. Ma mission est donc de récupérer toutes les données utiles lors de la session de jeu (position et orientation du patient, position, orientation et déplacement de ses mains…), de les stocker et de les afficher pour consultation par le médecin. Ainsi, ce dernier aura un retour sur la session de jeu. Après concertation avec des professionnels, des analyses de ces données pourront être envisagées.

1.3. Cahier des charges

Mon projet se concentre ainsi sur la partie récupération et transmission des mouvements du joueur. L'objectif est de concevoir un système qui comporte un module d'enregistrement des gestes d'interaction. Une réflexion sera menée sur la manière optimale d'envoyer et de stocker les données utiles à la description des capacités et de la fatigue des participants.

Le médecin doit pouvoir avoir accès, via une interface web, à un enregistrement des mouvements d'interaction du patient qui a joué avec le casque de réalité virtuelle. Les mouvements d'interactions comprennent la position et l'orientation du joueur dans l'espace à minima, et si possible les mouvements des manettes. Il est important de choisir un mode de stockage des enregistrements optimal : il faut que chaque enregistrement soit associé au jeu, niveau de difficulté, date et durée de la session.

1.4. Choix matériels et logiciels

Le casque de réalité virtuelle utilisé est un Oculus Quest. La scène virtuelle est créée grâce au logiciel Unity 3D. Une Raspberry Pi sous le système d'exploitation RetroPie permet d'émuler des anciennes consoles et ainsi faire tourner des jeux 2D anciens. Le flux vidéo est envoyé au casque de réalité virtuelle. Un serveur TCP permet de recevoir l'état des manettes (les boutons appuyés). Enfin, un autre serveur TCP permet d'envoyer un changement de configuration. De plus, un serveur web offre la possibilité à un client de demander un changement de configuration et un second permet d'émuler une manette sur un téléphone afin de jouer en mode multijoueur (le patient sur le casque et le médecin sur son smartphone).

Les technologies utilisées sont le C# pour la partie réalité virtuelle sur Unity3D et le Javascript avec NodeJS pour les serveurs sur la Raspberry.

2. Présentation du dispositif

La réalité virtuelle est une technologie informatique qui simule la présence physique d'un utilisateur dans un environnement artificiellement généré par des logiciels. La réalité virtuelle crée un environnement avec lequel l'utilisateur peut interagir.

Le casque utilisé est un Oculus Quest. Il s'agit un casque de réalité virtuelle commercialisé par Oculus VR. L'appareil est entièrement autonome et sans fil avec deux contrôleurs (manettes) Oculus Touch. Il fonctionne sur un système Qualcomm Snapdragon 835 sur puce. L'OS utilisé est android 7.1.1.

Figure 2 Casque Oculus Quest

Pour ce projet, la scène virtuelle est créée grâce au logiciel Unity3D. Il s'agit d'un logiciel permettant de créer des jeux 2D et 3D pour toutes sortes de plateforme (mobile, PC…) ainsi que des jeux VR. Chaque scène est composée d'objets qui peuvent interagir entre eux et avec l'utilisateurs grâce à des scripts rédigés en C#. Dans ce projet, la scène VR se compose d'une pièce dans laquelle se trouve un écran et une ou plusieurs manettes de console de type NES. Sur l'écran est affiché un jeu en 2D de type Tetris ou Bomberman.

Le jeu 2D n'est pas exécuté dans le casque mais sur une Rapsberry Pi. Cette dernière tourne sous le système d'exploitation RetroPie, qui permet d'émuler des anciennes consoles. Elle crée un hotspot wifi sur lequel se connecte le casque. Le flux vidéo du jeu est envoyé grâce à un serveur TCP sur lequel le casque se connecte. Un autre serveur TCP permet de récupérer les boutons des manettes virtuelles sur lesquels le

joueur appuie. Il existe également un serveur Web sur lequel un autre joueur peut se connecter grâce à son téléphone ou un ordinateur. Il a ainsi accès à une manette virtuelle sur son appareil et il peut jouer en multijoueur.

Enfin un 3eme serveur TCP permet d'envoyer au casque des changements de configuration. En effet, le thérapeute peut se connecter à un serveur Web de la Raspberry pour choisir le nombre de manettes virtuelles dans la scène, la mise en mouvement des manettes…

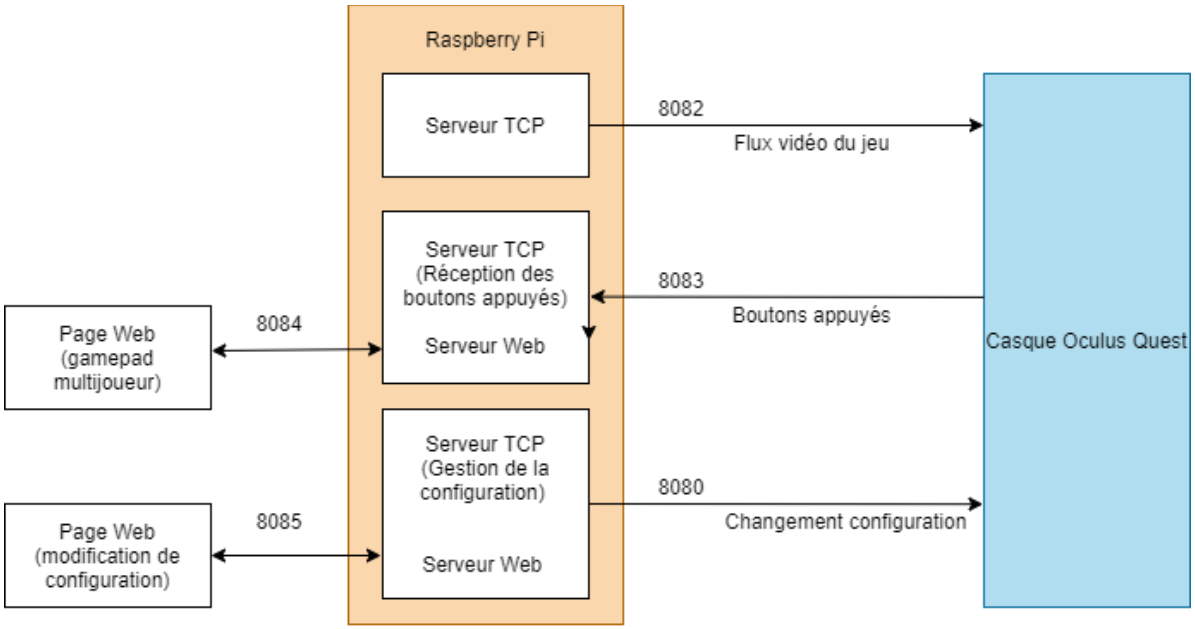


Figure 3 Architecture initiale du projet

Pour ma partie du projet, je ne dois modifier que le minimum de ce qui a déjà été réalisé. Ainsi, j'ai créé un nouveau serveur TCP sur le port 8091 et un nouveau serveur web sur le port 8090. Le serveur TCP permettra au casque d'envoyer les données des mouvements à la Raspberry qui stocke ces données dans des fichiers JSON. Le serveur Web offre une page web au thérapeute qui lui permet de lancer un enregistrement et de visualiser les enregistrements des mouvements du patient.

3. Réalisation du projet

3.1. Partie scène virtuelle Unity

Mon objectif est d'envoyer à intervalle de temps régulier, la position du patient dans l'espace. Il s'agit plus particulièrement de la position et de l'orientation de sa tête (donc du casque), de la position et de l'orientation de ses 2 mains (contrôleurs gauche et droit). Une version où l'on peut jouer sans les manettes (le casque détecte la position des mains) ayant été développée, les données envoyées ne sont pas les positions des contrôleurs mais des « Hand Anchor ». Il s'agit d'un objet situé au même endroit que les mains. Soit le casque positionne cet objet au niveau des manettes et affiche sur la scène virtuelle une image des manettes, soit le casque détecte la position des mains grâce aux multiples caméras dont il dispose. De ce fait, le module d'enregistrement des mouvements que je développe est compatible avec les deux versions du jeu.

Les positions sont représentées par un vecteur (x,y,z) représentant les coordonnées de l'objet dans l'espace, de manière relative à l'origine (centre de la scène virtuelle). L'orientation est représentée par un quaternion (x,y,z,w). Le w est un scalaire représentant la rotation autour du vecteur (x,y,z).

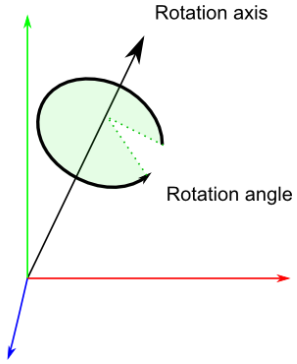


Figure 4 Schéma explicatif quaternion

Pour que je puisse envoyer les mouvements du patient, je crée un script « TCPMonitoring.cs ». Un script C# pour Unity déclare une classe qui possède des méthodes de base : Start() qui est exécutée lors de l'apparition de l'objet auquel est

rattachée la classe. Update() qui est exécutée à chaque rafraichissement de l'image sur le casque. Il est évidemment possible d'ajouter des nouvelles méthodes et variables. Ainsi, mon script déclare une classe *TCPMonitoring*. Au démarrage, il se connecte au serveur TCP de la Raspberry Pi sur le port 8091 (que j'ai choisi). De plus, il lance un thread pour recevoir des messages. Les messages qu'il reçoit sont « Run » pour le démarrage d'un enregistrement et « Stop » pour l'arrêt de l'enregistrement. L'intérêt d'utiliser un thread est de ne manquer aucun message. La réception des messages s'effectue donc en parallèle des autres actions de ce script.

Au début de mon projet, j'avais pensé enregistrer toute la session complète de jeu, du lancement du jeu jusqu'à ce que le patient quitte. Cependant, cela oblige de quitter le jeu et le relancer pour lancer une nouvelle session. Etant donné que l'on souhaite un système dans lequel le thérapeute n'intervient pas durant la session et n'utilise pas le casque virtuel, il n'était pas envisageable de contrôler le début et la fin de l'enregistrement depuis le casque. Ainsi, la solution pertinente a été d'offrir la possibilité au médecin de lancer et d'arrêter un enregistrement directement depuis une page web. Ainsi, les messages reçus par le casque sont des commandes de démarrage et de fin d'enregistrement.
Le framework .Net permet d'utiliser une classe *TCPClient* dans l'espace de noms System.Net.Sockets. Une instance de cette classe permet donc de se connecter facilement à un serveur TCP étant donné son adresse IP et son port. Cela permet ensuite l'envoi et la réception de données dans le flux.

Lorsqu'un enregistrement est lancé, le casque envoie la date, l'heure de début d'enregistrement et la configuration actuelle. De plus, il récupère la configuration actuelle et l'envoie également au serveur. Pour récupérer la configuration, on fait appel au Script qui gère le changement de configuration, *TCPConfig.* Quand une nouvelle configuration est envoyée, c'est ce script qui la récupère et la stocke dans une variable avant d'appliquer les changements correspondants. Je récupère donc cette variable et l'envoie à la Rapsberry avec le numéro 1 puisqu'il s'agit de la configuration initiale.

Ensuite, à intervalle de temps régulier, le script récupère l'ensemble des données de position, ainsi que le numéro de configuration actuel et l'heure. Au bout de 10 récupérations, il les envoie à la Raspberry Pi grâce au serveur TCP.
Quand *TCPConfig* reçoit une nouvelle configuration, il modifie une variable de mon script, ce qui me permet d'incrémenter mon compteur de configuration et d'envoyer la nouvelle configuration avec son numéro.

Il est à noter que les temps de récupération, nombre de récupérations avant envoi, adresse IP et port de la raspberry Pi sont des variables publiques. Ainsi, elles sont directement et facilement modifiables dans l'interface graphique de Unity, et ce sans devoir modifier le script. Des tests ont été réalisés et un temps de récupération de 400ms ne créée pas de latence dans le jeu.

3.2. Partie Serveurs Raspberry Pi

Sur la Raspberry, je crée un fichier serverMonitoring.js afin de créer les serveurs Web et TCP. Ce serveur, codé en javascript, utilise des paquets de Node.js. Node.js est une plateforme libre en JavaScript orienté serveur, qui permet entre autres de créer des applications en temps réel, où le serveur a la possibilité de transmettre de l'information au client.

3.2.1. Serveur TCP

Pour la création du serveur TCP, j'utilise le module Net. Le module Net fournit une API réseau asynchrone pour créer des serveurs TCP. La fonction Server() de ce module permet ainsi de créer un serveur TCP et de l'associer à un port donné grâce à listen().

Dès qu'un client se connecte sur le serveur, la socket est récupérée et permet de communiquer avec le client (en l'occurrence le casque de réalité virtuelle). Ainsi, le serveur peut recevoir plusieurs messages : des données *data* ou une fin de connexion (quand le casque se déconnecte du serveur lors de l'arrêt du jeu).

Lorsque le serveur reçoit des données, il reçoit le message *data* suivi d'une chaine de caractères comprenant les données, séparées par des points-virgules. Le premier champ des données indique le type de données reçues : Date, Config ou Log.

Lorsqu'il s'agit de Date, les champs suivants du message sont la date et l'heure. Lorsqu'il s'agit de Config, les champs suivants sont un numéro d'identification de la configuration et une chaine format JSON de la configuration. Enfin, quand il s'agit de Log, s'ensuivent l'heure et les 6 informations de position dans l'ordre suivant : position de la main gauche, rotation de la main gauche, position de la main droite, rotation de la main droite, position du casque, orientation du casque.

3.2.2. Stockage des sessions d'enregistrement

Pour stocker les données, j'avais d'abord pensé à MongoDB, qui est une base de données NoSQL orientée documents. MongoDB classe les documents dans différentes collections. Les données sont stockées au format JSON, un ensemble d'associations clé/valeurs, qui est cohérent avec celles de ce projet.
Après quelques tests, il s'est avéré que la version de MongoDB disponible sur RetroPie, est une version ancienne, qui n'est pas compatible avec le paquet de NodeJS « mongo », qui permet à un programme javascript de communiquer avec la base de données. Ainsi, il aurait fallu utiliser une version également plus ancienne de ce paquet pour que le tout fonctionne.
Après réflexion et discussion avec l'équipe qui travaille sur ce projet, il a été jugé peu utile d'avoir une base de données tournant en permanence sur la Raspberry Pi, étant donné la quantité relativement peu importante de données et la redondance des informations. Ainsi, j'ai préféré gérer des fichiers JSON directement dans un répertoire de la Raspberry. Nous avons donc à disposition directement des fichiers JSON, ce qui pourra être utile pour une future analyse des données, contrairement à MongoDB qui ne stocke pas les données dans des fichiers JSON directement accessibles.

Pour ce faire, j'ai utilisé un paquet NodeJS nommé « lowDB », qui permet de gérer une mini base de données sous forme de fichiers JSON dans un répertoire. Ainsi, on peut facilement récupérer le contenu d'un fichier JSON, de créer un fichier JSON, d'y ajouter du contenu…

Le répertoire de stockage des sessions d'enregistrement est stocké dans une variable DB_PATH, qui est facilement modifiable. Dans ce dossier, les fichiers sont enregistrés sous le nom :

AAAAMMJJ_HHMMSS_customName.json

où :

- AAAAMMJJ représente la date d'enregistrement au format inversé (Année,Mois,Jour). Ce format permet un classement des sessions dans l'ordre chronologique dans le répertoire.
- HHMMSS représente l'heure de début d'enregistrement (Heures, Minutes, Secondes).
- customName représente un nom de session donné par le thérapeute. Ainsi, il pourra nommer la session par rapport au nom du patient par exemple, ce qui lui permettra de la retrouver plus facilement.

Dans ces fichiers, les informations sont organisées comme suit :

```
{
  "Date":"...",
  "Start Time":"...",
  "Logs":[
     {
        "Time":"...",
        "Left":"...",
        "Left Rotation":"...",
        "Right":"...",
        "Right Rotation":"...",
        "Head":"...",
        "Head Orientation":"..."
        "Config":"...",
     },
     {
        "Time":"...",
        "Left":"...",
        "Left Rotation":"...",
        "Right":"...",
        "Right Rotation":"...",
        "Head":"...",
        "Head Orientation":"..."
        "Config":"...",
     },
       ...
  ]
}
```

Pour chaque log (positions à un instant donné) est associé un numéro de configuration. Ce numéro fait référence à une configuration bien précise. Au début de la session d'enregistrement, la configuration est récupérée et porte le numéro 1. A chaque changement de configuration durant la session, celle-ci est enregistrée et aura un numéro incrémenté de 1 par rapport au précédent.

La configuration, facilement modifiable par les médecins est stockée dans un fichier json de la raspberry. Au démarrage du jeu, le script TCPConfig charge le fichier JSON dans une variable et modifie les objets de la scène du jeu. Pour éviter de lire une deuxième fois le fichier JSON, je modifie le script TCPConfig afin de créer une fonction publique GetConfig() qui renvoie une chaîne au format json de la configuration. Dans mon script TCPMonitoring, j'appelle donc cette fonction et récupère la configuration. Lorsque le casque reçoit la commande RUN (quand le médecin demande le début de l'enregistrement sur la page web), le casque envoie en premier la date et l'heure. Ensuite il envoie donc la configuration récupérée précédemment et son numéro. Ensuite, à chaque log envoyé, sera associé le numéro de la configuration actuelle. Les configurations sont enregistrées dans un sous répertoire du dossier où sont stockés les sessions. La configuration est récupérée au format JSON et stockée sous ce même format. Le nom de ces fichiers est AAAAMMJJ_HHMMSS_customName_configX.json, où X correspond au numéro de la configuration.
Pour détecter un changement de configuration, le script TCPConfig modifie la variable newConfig de mon script à true lors du changement.

3.2.3. Serveur Web

Pour la création du serveur web, j'ai utilisé les modules express et socket.io. Express est une infrastructure d'applications Web Node.js minimaliste et flexible qui fournit un ensemble de fonctionnalités robustes pour les applications Web et mobiles. Socket.io est une bibliothèque Javascript pour les applications Web en temps réel. Il permet une communication bidirectionnelle en temps réel entre les clients Web et les serveurs.

Le serveur Web ainsi créé écoute sur un port choisi. Je redirige toute connexion sur serveur vers ma page HTML « indexmonitoring.html ». Ainsi, lorsqu'un médecin se

connectera au serveur en entrant uniquement l'adresse IP de la Raspberry et le port, il sera automatiquement dirigé vers la page web.

La page Web est composée de 2 parties : la partie gestion de l'enregistrement et la partie gestion des sessions enregistrées. Pour gérer l'enregistrement, 2 boutons sont affichés : Run et Stop. Si le casque n'est pas connecté au serveur TCP (le jeu n'est pas démarré), alors il n'est pas possible d'appuyer sur le bouton, qui est désactivé. Quand le casque est connecté, on peut appuyer sur le bouton Run mais le bouton stop est désactivé. Si un enregistrement est en cours, le bouton Run est désactivé jusqu'à l'appui sur le bouton stop. Ainsi, la page a besoin de savoir si un casque est connecté ou non. Le serveur Web envoie donc un message « connected » suite à la connexion du casque. De plus, un champ de saisie de texte permet de saisir un nom pour la session (facultatif).
Pour la gestion des sessions enregistrées, il s'agit d'une liste des sessions, avec pour chacune d'elle un bouton Select pour afficher la session et un bouton Delete pour la supprimer. Une fois la session choisie, s'affichent à l'écran tous les logs (heure et positions) dans un tableau, ainsi qu'un bouton retour pour retourner à la liste des sessions. Quand on clique sur le numéro de configuration d'un log, une pop-up s'ouvre et affiche la configuration désirée.
Pour supprimer une session, un appui sur le bouton Delete ouvre une pop-up qui demande une confirmation de suppression.

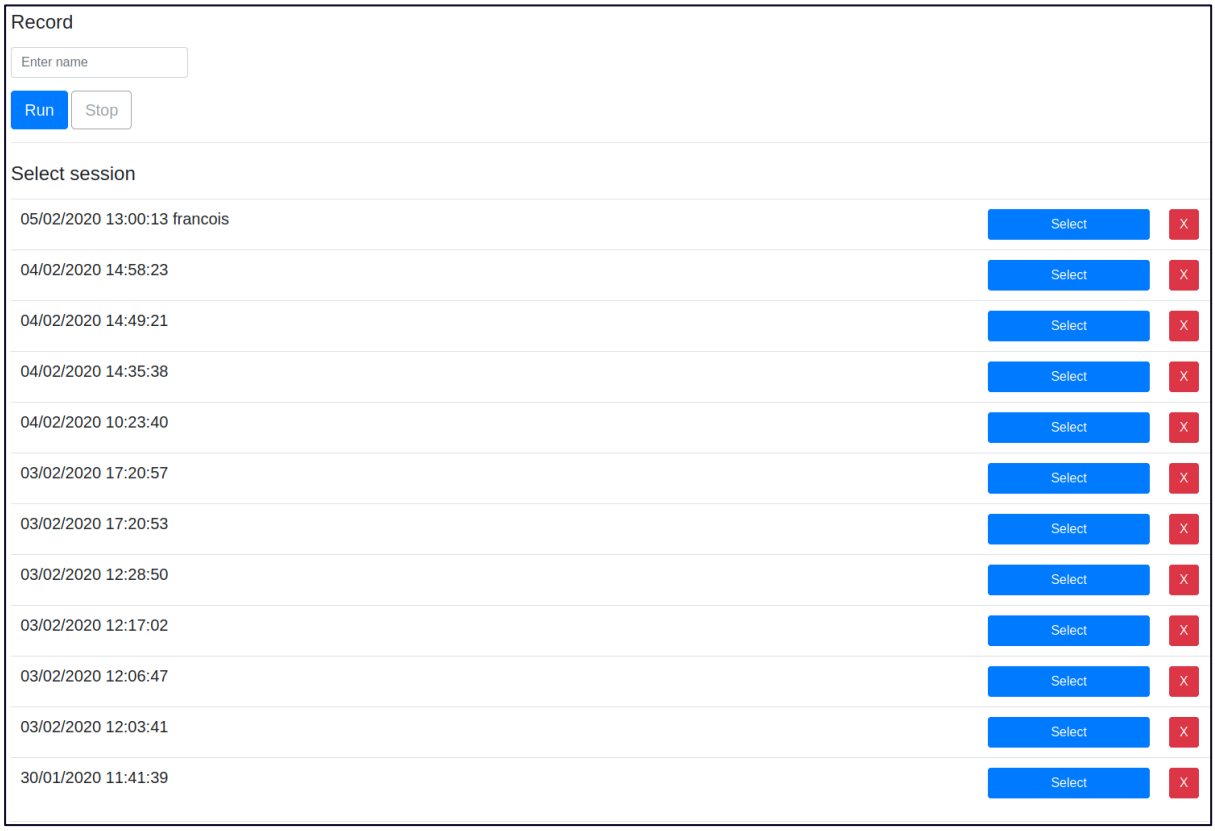
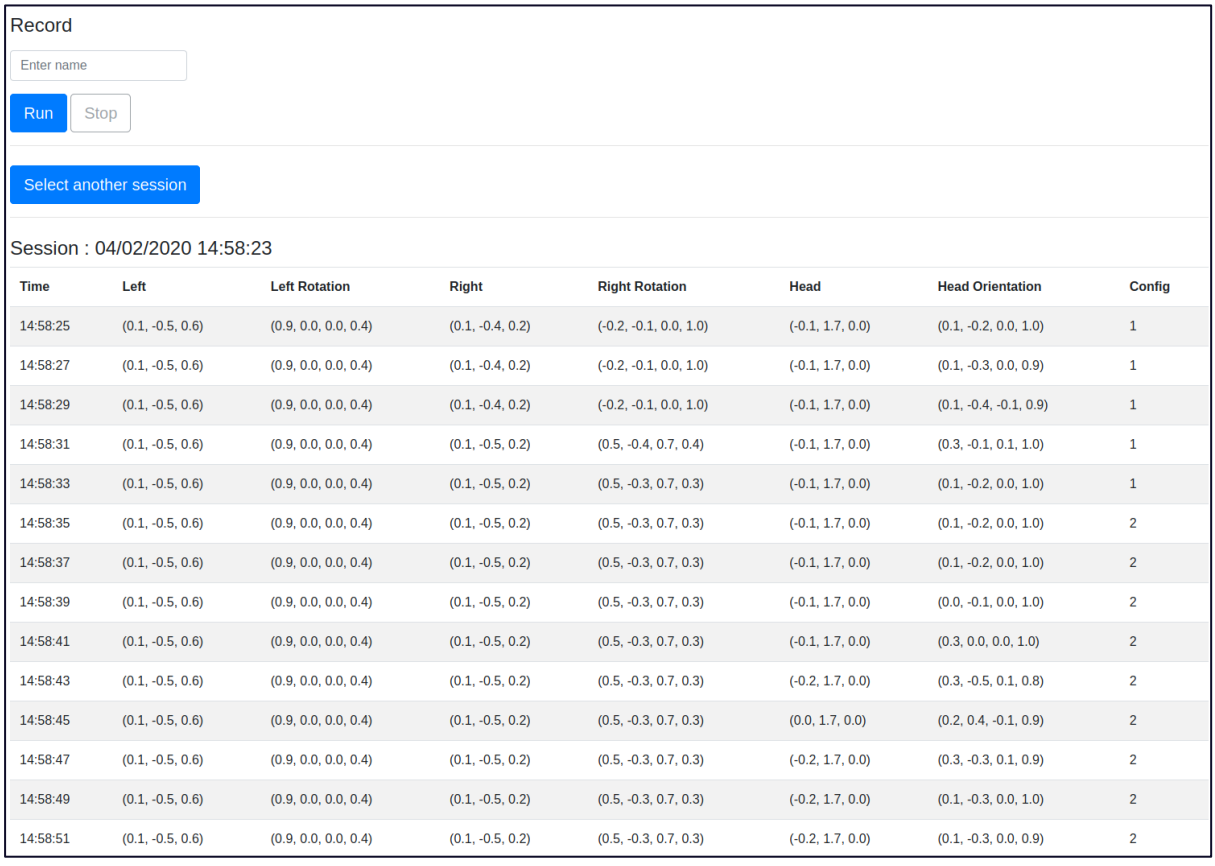


Figure 5 Capture de la page Web - Menu Principal



Time	Left	Left Rotation	Right	Right Rotation	Head	Head Orientation	Config
14:58:25	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.4, 0.2)	(-0.2, -0.1, 0.0, 1.0)	(-0.1, 1.7, 0.0)	(0.1, -0.2, 0.0, 1.0)	1
14:58:27	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.4, 0.2)	(-0.2, -0.1, 0.0, 1.0)	(-0.1, 1.7, 0.0)	(0.1, -0.3, 0.0, 0.9)	1
14:58:29	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.4, 0.2)	(-0.2, -0.1, 0.0, 1.0)	(-0.1, 1.7, 0.0)	(0.1, -0.4, -0.1, 0.9)	1
14:58:31	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.4, 0.7, 0.4)	(-0.1, 1.7, 0.0)	(0.3, -0.1, 0.1, 1.0)	1
14:58:33	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.1, 1.7, 0.0)	(0.1, -0.2, 0.0, 1.0)	1
14:58:35	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.1, 1.7, 0.0)	(0.1, -0.2, 0.0, 1.0)	2
14:58:37	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.1, 1.7, 0.0)	(0.1, -0.2, 0.0, 1.0)	2
14:58:39	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.1, 1.7, 0.0)	(0.0, -0.1, 0.0, 1.0)	2
14:58:41	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.1, 1.7, 0.0)	(0.3, 0.0, 0.0, 1.0)	2
14:58:43	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.2, 1.7, 0.0)	(0.3, -0.5, 0.1, 0.8)	2
14:58:45	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(0.0, 1.7, 0.0)	(0.2, 0.4, -0.1, 0.9)	2
14:58:47	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.2, 1.7, 0.0)	(0.3, -0.3, 0.1, 0.9)	2
14:58:49	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.2, 1.7, 0.0)	(0.1, -0.3, 0.0, 1.0)	2
14:58:51	(0.1, -0.5, 0.6)	(0.9, 0.0, 0.0, 0.4)	(0.1, -0.5, 0.2)	(0.5, -0.3, 0.7, 0.3)	(-0.2, 1.7, 0.0)	(0.1, -0.3, 0.0, 0.9)	2

Figure 6 Capture de la page Web - Affichage d'une session

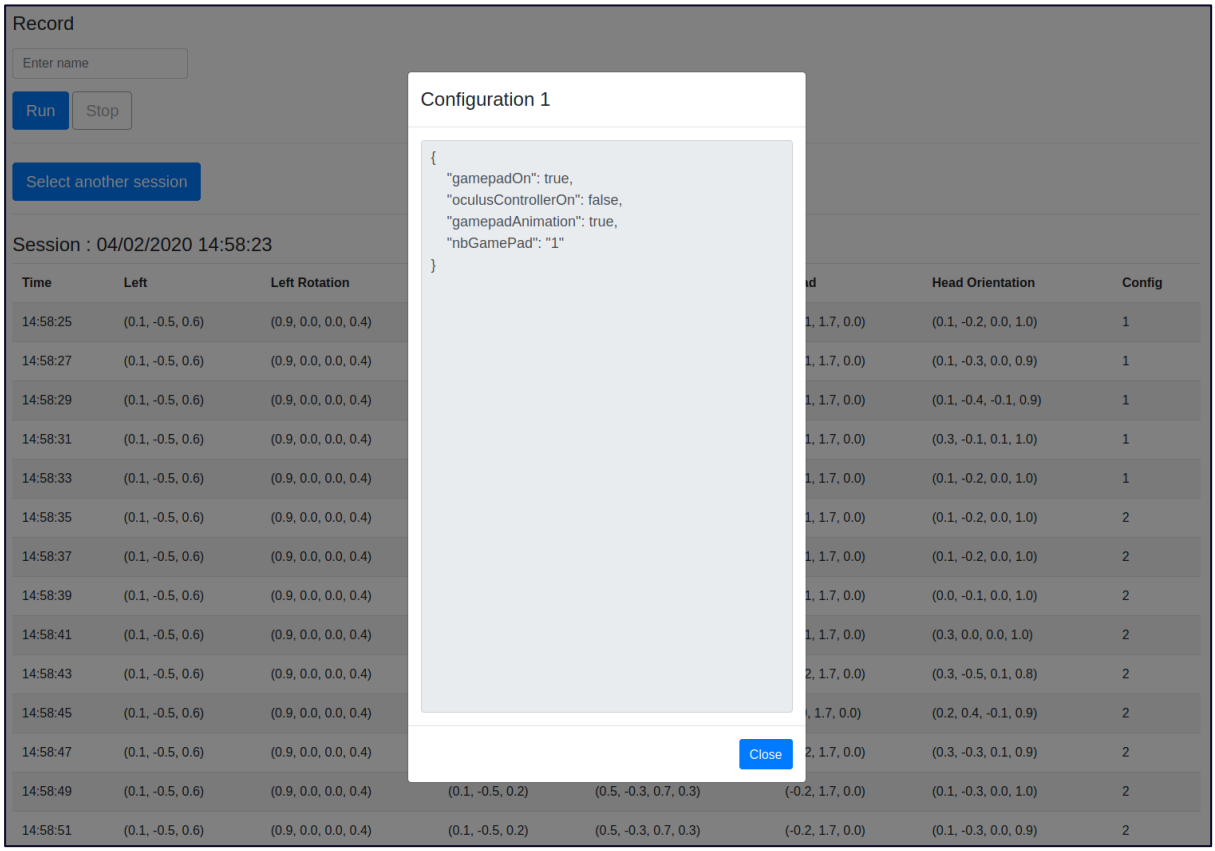


Figure 7 Capture de la page Web - Affichage d'une configuration

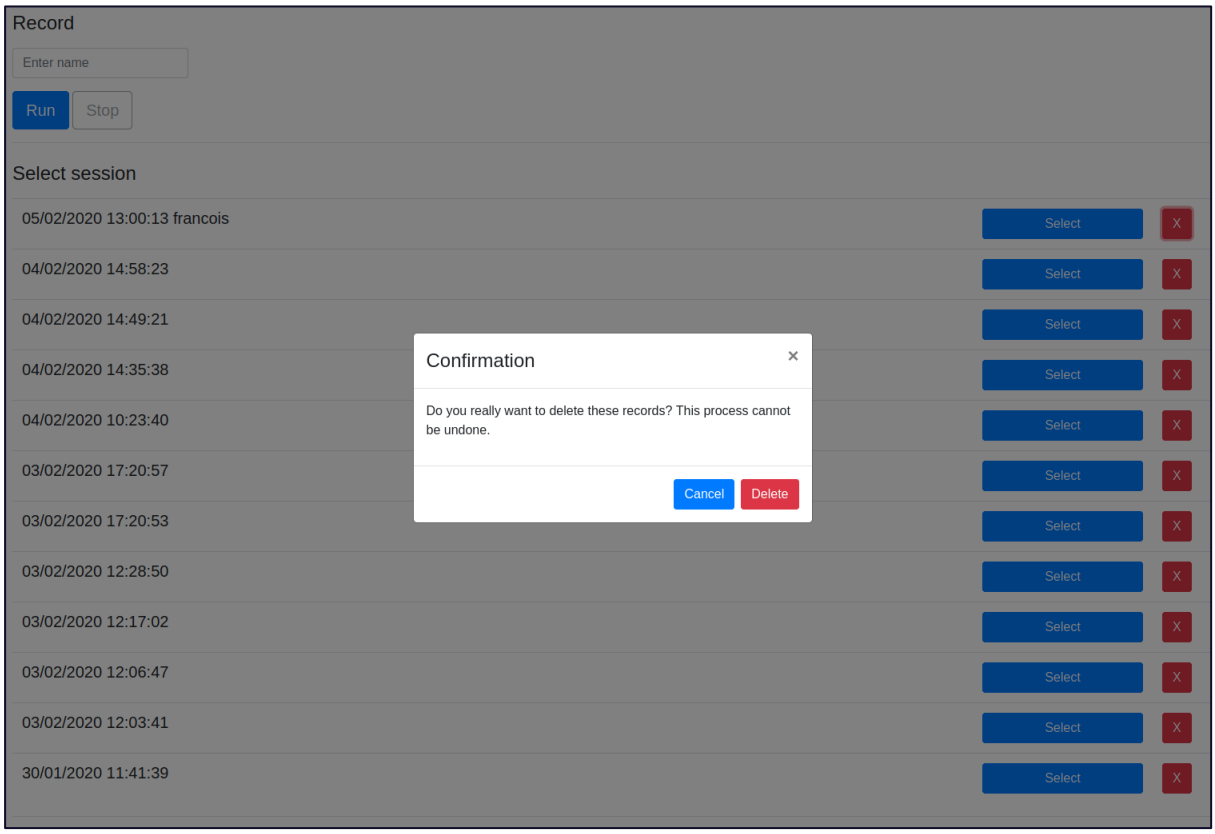


Figure 8 Capture de la page Web - Suppression d'une session

La page web utilise Bootstrap pour avoir un rendu esthétique et pour les pop-up. Bootstrap est un framework utile à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. Il regroupe une collection d'outils fournis sous la forme de classes CSS et de librairies JavaScript. Ainsi, c'est Bootstrap qui permet d'afficher une boite de dialogue « Modal » (pour la configuration et la confirmation de suppression). Les boites de dialogues peuvent être facilement ouvertes et fermées grâce à un appel de fonction du script javascript de Bootstrap.

Le serveur Web fonctionne avec un système de signaux. Le serveur peut recevoir un signal et effectuer des actions en fonction de ce signal. Il peut également envoyer un signal aux clients connectés. Les signaux peuvent être accompagnés de données. Il existe des signaux de base (connexion, déconnexion...) mais des signaux peuvent être créés et implémentés.

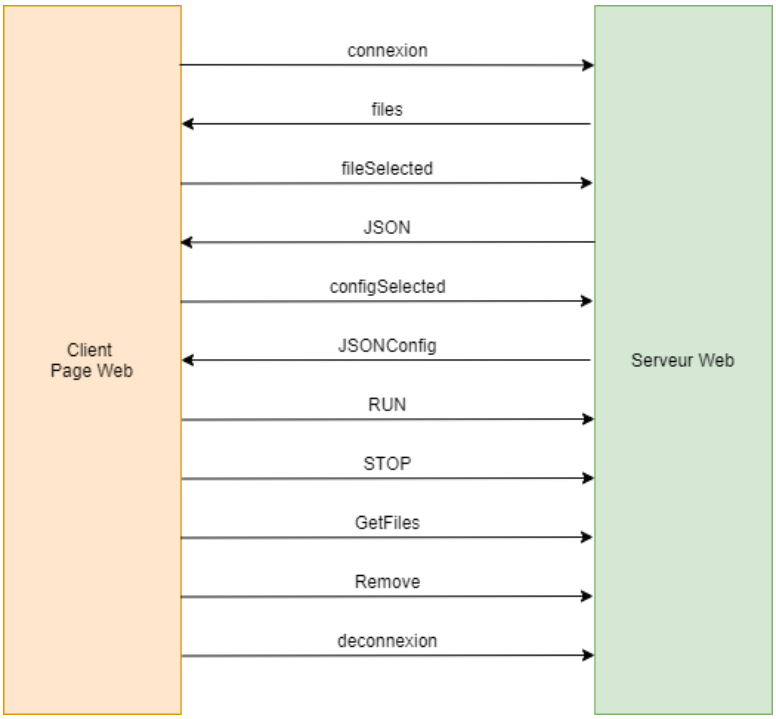


Figure 9 Schéma des communications entre le serveur web et le client

Les principaux signaux utilisés sont les suivants :

- Files

Lors de la connexion, le serveur envoie le signal Files, accompagné de la liste des fichiers JSON de session. Pour ce faire, la fonction *fs.readdir(<chemin>)* renvoie la liste

de tout ce qui se trouve dans un répertoire. Les fichiers dont l'extension n'est pas « .json » sont retirés. La liste est ensuite inversée pour l'avoir dans un ordre anti chronologique, afin d'avoir la session la plus récente en premier. Cela permet également de retirer les éventuels sous-répertoires.
Ainsi, lors de la connexion, le client reçoit directement les sessions disponibles qui sont affichées. Pour chaque session, un bouton Select et Delete sont associés. L'attribut « name » de ces éléments correspond au nom du fichier. Lors de l'appui sur le bouton Select, le signal FileSelected est émis avec le nom du fichier. Lors de l'appui sur le bouton Delete, la boite de dialogue de confirmation de suppression s'ouvre.

- FileSelected et JSON, ConfigSelected et JSONConfig

Quand le serveur reçoit le fichier de session sélectionné, celui-ci est ouvert, converti du format JSON à un objet Javascript et envoyé au client avec le signal « JSON ». Une fois les données de sessions reçues, le client affiche chaque log dans une ligne d'un tableau. Lors du clic sur un numéro de configuration, le signal ConfigSelected est envoyé avec le nom du fichier de configuration désiré (concaténation du nom du fichier de session avec « _configX.json »). Le serveur reçoit ce message, ouvre le fichier JSON, le convertit en objet JavaScript et l'envoie à la page web avec le signal JSONConfig. Une fois reçue, cette dernière ouvre une boite de dialogue afin d'y afficher la configuration.

- Connected et disconnected

Quand le casque se connecte au serveur TCP, le signal connected est envoyé au client du serveur web. De même quand il se déconnecte avec le signal « disconnected ». Cela permet d'autoriser ou non l'appui sur le bouton Start afin d'empêcher d'envoyer la commande de début d'enregistrement si aucun casque n'est connecté.

- Run et Stop, GetFiles

Lors de l'appui sur le bouton Run, le bouton Stop devient cliquable et le message « Run » est transmis au serveur Web, puis au casque par l'intermédiaire de la socket TCP. Cela fait passer la variable recording du script TCPMonitorign a true et démarre l'enregistrement.
De même, lorsque le bouton Stop est pressé, envoi du signal « Stop » transmis au casque qui arrête l'enregistrement et envoie les derniers logs non envoyés. Le signal GetFiles est alors envoyé de la page au serveur web afin de récupérer la liste des

fichiers de configuration et permettre de visualiser directement la session qui vient d'être créée.

- Remove

Lorsque le thérapeute confirme la suppression d'une session, le signal remove est envoyé avec le nom du fichier de session à supprimer. Le fichier est alors supprimé, grâce à la fonciton *fs.unlinkSync(<file>).* Il faut également supprimer les fichiers de configuration correspondant à cette session. Pour ce faire, je récupère la liste des fichiers contenus dans le dossier de configuration. Une expression régulière me permet de trouver les bons fichiers et de les supprimer.

3.2.4. Utilisation de PM2

Pour la stabilité du système, PM2 est utilisé. PM2 est un gestionnaire de processus pour le moteur d'exécution JavaScript Node.js. Il permet de garder les applications en vie pour toujours et de les recharger sans interruption. PM2 permet également de gérer la journalisation et la surveillance des applications.

Conclusion

Le cahier des charges initial est respecté. Le module d'enregistrement des mouvements d'interaction est fonctionnel. Il a été intégré sans problème au projet existant. Des démonstrations ont été faites aux membres de l'équipe qui sont satisfaits.

Je suis personnellement fier de ce projet que je suis parvenu à réaliser en un temps limité d'environ 6 semaines. J'ai pu découvrir le fonctionnement d'un laboratoire de recherche et participer à des expériences dans le cadre d'autres projets de recherche. Cette expérience m'a également permis de me former sur de nouveaux langages : Javascript et NodeJS, C#. De plus, j'ai pu me familiariser avec la réalité virtuelle.

Grâce à ces quelques semaines passées à travailler sur ce projet, mon champ de connaissances s'est encore élargi.

Sources

Site web du projet VR4REHAB : https://www.nweurope.eu/projects/project-search/vr4rehab-virtual-reality-for-rehabilitation/

Documentation lowDB : https://www.npmjs.com/package/lowdb

Documentation Node JS : https://nodejs.org/api/

Documentation Bootstrap : https://getbootstrap.com/docs/4.4/getting-started/introduction/

Code du projet

Le code source de ma partie de ce projet est disponible sur mon wiki : https://projets-ima.plil.fr/mediawiki/index.php/IMA5_2019/2020_P17#Documents_Rendus