APP
ROBOTIQUE DE SERVICE

BENHIMA Mehdi
DINDELLI Dylan-Anthony
JENNY Camille
LÉGLISE Cloé
MISON Jules
RUIZ Florian
SANGOUARD Marine

Table des matières

I. Introduction

Dans le cadre de notre semestre 7 dans la filière « Systèmes Numériques – Instrumentation », nous poursuivons notre apprentissage par projet centré sur la robotique de service. Notre projet consiste à utiliser les robots Pepper et Nao pour réaliser une visite guidée de Polytech. Le processus de notre projet comprend la conception jusqu'à la réalisation et la validation de celui-ci.

Le premier semestre nous a permis de comprendre et de s'habituer au fonctionnement global des logiciels et robots que nous utiliserons afin de les maîtriser au mieux tout au long du projet. Nous avions déjà commencé à réaliser quelques tests afin de connaître les capacités et limites des robots, notamment son déplacement intelligent, l'interaction avec son environnement et la communication avec un serveur et donc avec d'autres entités. Grâce à ce travail préparatoire, nous avons pu naturellement réaliser une ébauche de cahier des charges du projet.

Durant ce semestre, nous nous sommes alors concentrés sur la continuation et la finalisation de la rédaction du cahier des charges, ainsi que sur le début de la réalisation du projet. Ainsi, nous avons pu trier nos différents objectifs par niveau de difficulté et mieux poser la structure du projet. Nous avons aussi pu commencer à programmer les différentes parties du projet, en se basant sur les programmes du semestre précédent qui étaient fonctionnels.

II. Objectifs

Pour commencer ce projet, nous avons choisi de se baser sur le bâtiment de Polytech Annecy, et plus précisément de programmer Pepper pour qu'il puisse déjà se repérer dans le deuxième étage où notre salle d'APP se situe ; le but étant de parvenir à faire de lui un guide de l'école lors d'une journée portes ouvertes en indiquant les salles intéressantes à visiter pour les élèves.

Pour ce faire, notre but est tout d'abord d'identifier différents objectifs spécifiques aux robots. Ces objectifs seront les lignes directrices de notre projet, qui devront se coordonner les unes avec les autres pour une réalisation fonctionnelle. Nous avons donc choisi de construire notre cahier des charges à partir de ces objectifs et les classant par ordre de difficulté *(cf Annexe 1 - Cahier des charges)*. Ce cahier des charges prenait initialement la forme d'un tableau à deux entrées (les différentes parties de notre projet par rapport à des niveaux de difficultés). Ensuite, nous avançons sur l'élaboration de plusieurs points du projet pour compléter et préciser au fur et à mesure notre cahier des charges par écrit.

III. Répartition des rôles et organisation

Par rapport au semestre précédent, la composition de notre équipe est légèrement différente. Nous avons donc dû modifier légèrement notre organisation de travail, même si la plupart des rôles sont restés identiques.

Pour le semestre, c'est Cloé LÉGLISE qui a été désignée chef de projet. En plus de cette fonction, elle a travaillé principalement sur la réalisation de l'application web comprenant le plan de l'école que nous souhaitons afficher sur le robot Pepper.

Camille JENNY a conservé son poste sur le développement du code pour le déplacement de Pepper, tout comme au semestre précédent. Mehdi BENHIMA est également resté sur le même poste qu'avant, sur la partie serveur, et a été rejoint par Dylan-Anthony DINDELLI, nouveau dans notre groupe. Jules MISON, lui aussi, a gardé le même rôle qu'avant, en travaillant sur la communication orale avec Nao, et c'est également lui qui s'est occupé de l'algorithme de plus court chemin en priorité. Il a été épaulé par Florian RUIZ, qui a effectué la majorité des petites tâches nécessaires au projet telles que recenser l'ensemble des salles et leur fonction, par exemple, mais a également beaucoup travaillé sur le cahier des charges, tout comme Marine SANGOUARD. Cette dernière a également aidé plusieurs membres de l'équipe, principalement sur le déplacement de Pepper.

IV. Travail effectué durant le semestre

1. Le cahier des charges

Le cahier des charges étant le thème de ce semestre, il a bien entendu été au cœur de notre travail durant ce semestre. Nous avons concentré un temps assez important à la réflexion et la réalisation du cahier des charges, que nous avons décidé d'effectuer par niveau de difficultés. Ainsi, le premier niveau nous permet d'avoir des fonctionnalités de bases afin d'assembler un projet fonctionnel, et une fois que ces fonctionnalités-la sont implémentées, nous pouvons passer au niveau supérieur. Ainsi, nous devrions avoir un projet fonctionnel d'ici la fin du semestre, même si nous risquons de devoir abandonner certaines fonctionnalités nous ne devrions pas bloquer la totalité du projet.

Le cahier des charges a donc été rédigé de cette manière, et il se trouve en annexe de ce document. Une fois que cela a été fait, nous nous sommes concentrés sur la programmation des différentes fonctionnalités, et avons donc commencé à avancer sur le projet de manière plus concrète.

2. L'outil Git

Lors du semestre précédent, nous avions créé un repository Git sur lequel mettre tous nos codes. La plupart des personnes de notre équipe ne sachant pas se servir de l'outil Git, un des axes de travail de ce semestre a été de faire une réunion pour que chacun puisse utiliser cet outil de manière simple mais efficace pour l'ensemble de l'équipe, mais également de réorganiser le repository. En effet, celui-ci avait initialement été pensé pour que chaque partie du développement ait sa branche propre afin que personne ne se marche dessus, et également parce que pour le moment, toutes ces parties sont indépendantes (leur raccordement se fera lors du semestre prochain).

Suite à la remarque de l'un de nos professeurs encadrants, cependant, nous avons compris qu'il fallait que nous réorganisions le repository afin que toutes les parties du développement soient centralisées. Nous avons donc réfléchi à une structure temporaire pour nos différents dossiers – qui changera inévitablement lors de la connexion des différentes parties – afin d'obtenir un code général propre.

Nous avons donc pris le temps de trier toutes les parties du code qu'il nous fallait conserver et effectué un "merge" de toutes les branches dans le main. Aujourd'hui, le repository ne contient donc plus que la branche main, organisée selon notre architecture en deux axes principaux : le code consacré aux robots sur Chorégraphe (la communication, la tablette, le déplacement…) et celui consacré au serveur. Il reste également une branche que l'un d'entre nous a l'habitude d'utiliser pour faire des tests qui n'ont pas leur place sur le code principal.

Nous ferons désormais attention à mieux utiliser Git de la manière dont l'outil a été pensé, en ne créant des branches que lors d'essais sur des parties importantes et en travaillant plus souvent sur la branche main.

3. L'application web

Une partie de notre projet consiste à mettre en place un système de communication entre l'utilisateur et la machine. La partie orale, à l'aide du *speech reco*, sera développée dans une autre sous-partie.

La tablette tactile du robot Pepper fait partie des outils à notre disposition que nous avons exploités afin de mettre en place cette interaction. Une application a donc été développée dans les langages HTML, CSS et JavaScript. Elle contient une page d'accueil avec une explication sommaire pour le visiteur.

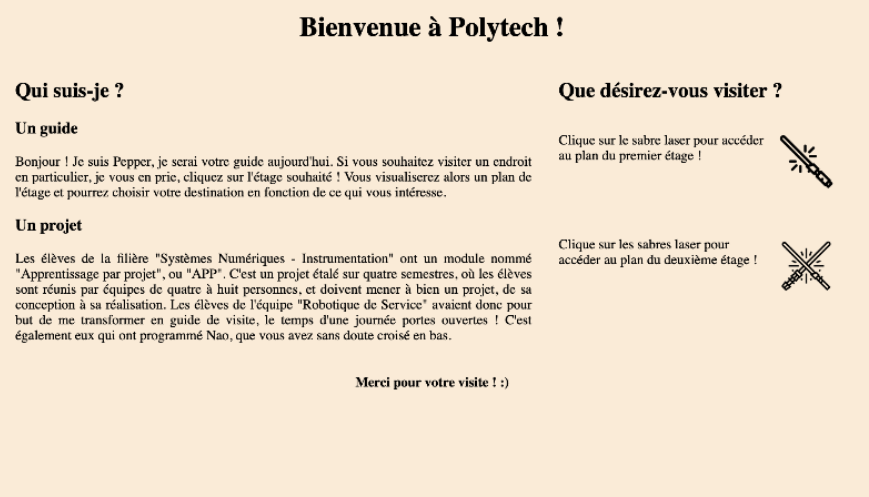

Figure 1 : Page d'accueil de notre application web

Depuis cette page d'accueil, l'utilisateur peut se déplacer au choix sur deux pages. Les deux pages sont extrêmement similaires et chacune contient un plan interactif d'un des deux étages du bâtiment de Polytech Annecy sur lesquels nous avons décidé de travailler : le premier et le deuxième.

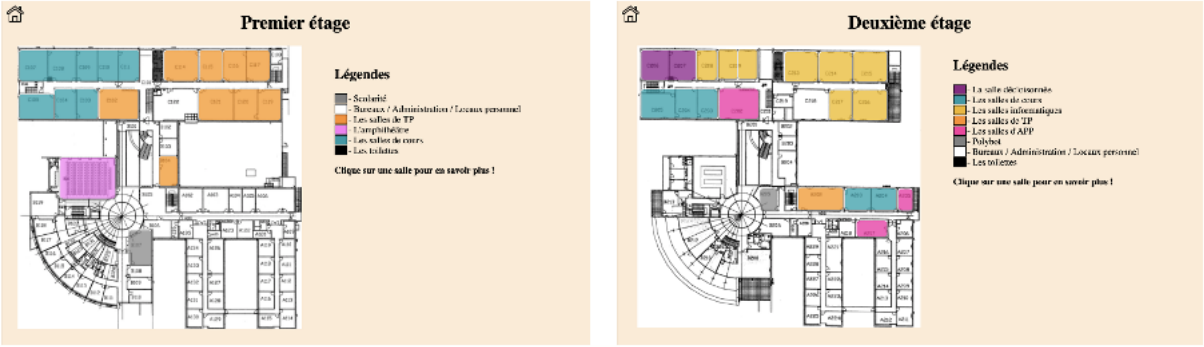

Figure 2 : Pages secondaires de notre application web

Ces plans contiennent des cases colorées et légendées selon leur catégorie (salles de TP, salles informatiques, amphithéâtres…) correspondant aux différentes salles que nous avons jugées les plus intéressantes à connaître pour un futur élève (nous avons omis les locaux administratifs, par exemple). Le fait de cliquer sur une case affiche une description rapide de la salle ainsi qu'un bouton.

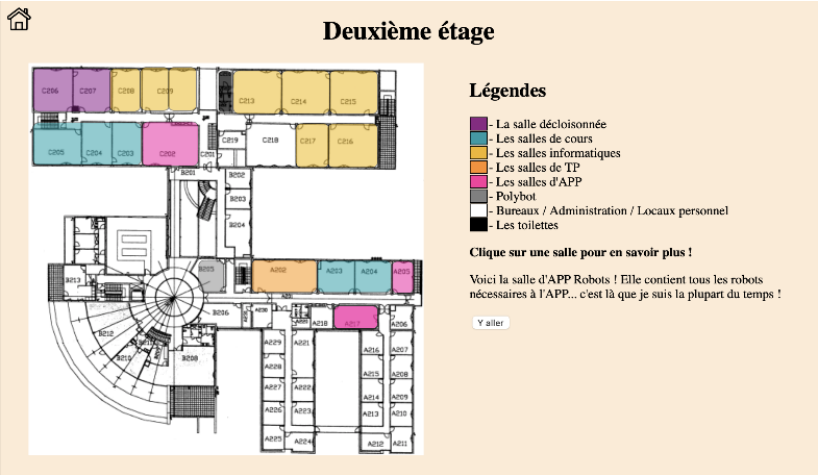

Figure 3 : Visuel de l'application lorsqu'une salle est sélectionnée

Ce dernier bouton n'est pas encore utile, mais lors du raccordement des différentes parties du projet au prochain semestre, le but sera, selon le niveau du cahier des charges développé, d'afficher un plan avec l'algorithme de plus court chemin en partant de la position initiale du robot, ou bien directement de mettre en marche Pepper pour qu'elle se déplace vers la salle désirée afin de montrer le chemin au visiteur.

En plus de la réalisation de cette application, nous avons réussi à l'afficher sur la tablette de Pepper à l'aide d'un bloc Chorégraphe. Pour la suite de la réalisation, lors de la mise en commun des différentes parties, il faudra incorporer ce bloc au reste du projet Chorégraphe principal. Afin que le bouton « Y aller » lance une action de Pepper, nous aurons besoin de lancer une action sur Chorégraphe, en utilisant les fonctions Javascript.

4. Communication orale en utilisant « speech reco »

Un deuxième système de communication que nous avons exploité est la communication orale. Nous avons voulu que nos robots Nao et Pepper soient en capacité de comprendre une question, y répondre et en ressortir une action si nécessaire. Pour cela, nous sommes partis d'un bloc que nous avions réalisé au semestre précédent, qui permet au bloc *speech reco* de prendre une « Word List » qui dans notre cas sera une liste de questions. Cette liste sera extraite d'un fichier texte qui contiendra les questions, les réponses, et les actions. Les questions ont un marqueur (“$”) que repérera le programme. Nous avons créé un bloc qui récupère toutes les questions de ce fichier texte, afin de les entrer dans la *wordList*. Le bloc *speech reco* fait son travail et une fois une des questions préenregistrées reconnue, le programme renvoit la réponse, et s'il y en a une l'action à effectuer.

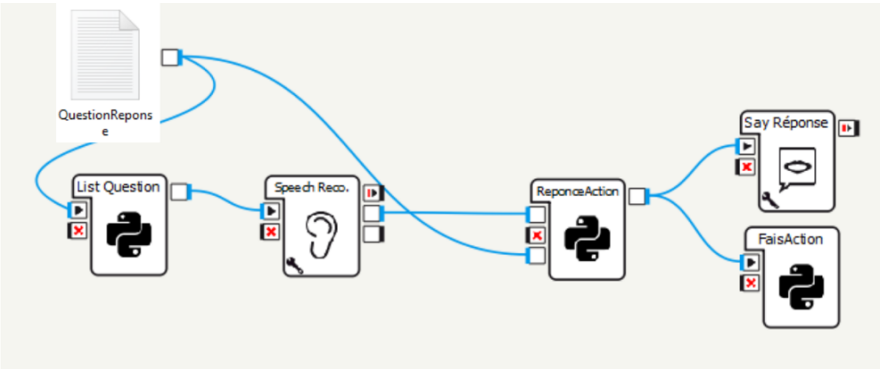

Figure 4 : Programme Chorégraphe correspondant à l'action du speech reco

L'avantage de cette méthode est qu'elle est très efficace pour reconnaître des phrases pré-écrites et donc peut donner une réponse. Elle a même la capacité de reconnaître plusieurs questions pour une même réponse. Le point négatif, elle n'a pas la capacité de comprendre des petits changements dans la phrase. Si la phrase est : « Où est la salle de maths » et que l'utilisateur demande « Où se trouve la salle de maths », le robot ne comprendra pas la question. Il faudrait pour ça enregistrer toutes les alternatives possibles à une même question pour une réponse. Par la suite, nous voulons créer un algorithme qui permet, à partir d'un fichier son qui contient une phrase, de comprendre le sens et d'en déduire une réponse adaptée.

5. Implémentation d'une librairie Chorégraphe

Nous avons créé une librairie Chorégraphe où on peut déposer tous les blocs que nous avons créés afin qu'ils soient réutilisés facilement par un membre de l'équipe.

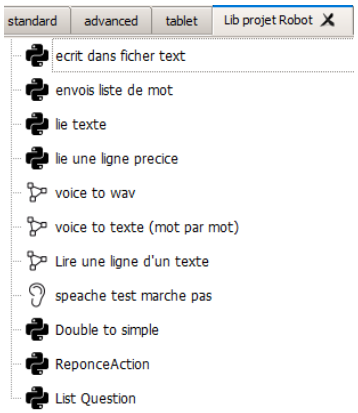

Figure 5 : Librairie Chorégraphe du projet

6. Algorithme de plus court chemin

Nous avons voulu créer un algorithme qui nous permet de trouver le plus court chemin permettant de donner a Pepper le chemin à suivre, mais aussi pour que des visiteurs puissent avoir le chemin directement affiché sur Pepper ou la possibilité de récupérer cette carte grâce à un QR code sur un smartphone. Pour atteindre cet objectif, il faut réaliser plusieurs étapes :

a. Cartographier Polytech et les points d'intérêt
b. Programmer l'algorithme
c. Sauvegarder les données sur un cloud
d. Générer un QR code

a. Cartographie de Polytech

Pour cartographier Polytech, nous somme partis d'un plan de l'école que nous avons quadrillé. Nous avons repéré les coordonnées des cases où nous voulions que le robots ou une personne puisse s'arrêter. Nous avons repéré les entrées des salles et certaines intersections qui sont nécessaires pour le fonctionnement du programme. Nous avons ensuite réalisé les liaisons entre ces points.

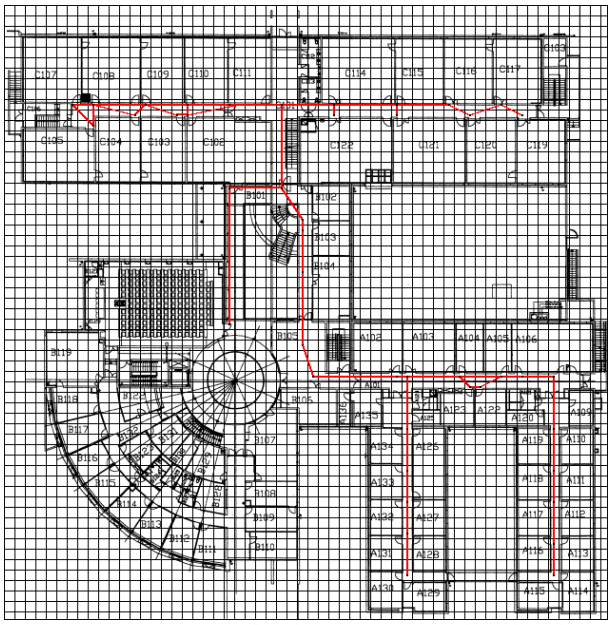

Figure 6 : Cartographie du premier étage de Polytech

b. Programmation de l'algorithme

Nous avons décidé de développer le programme de plus court chemin à partir de zéro. Nous avons aussi réalisé un protocole qui explique le fonctionnement du programme (*cf Annexe 3 – Protocole algorithme plus court chemin*). Il génère ensuite une image en PNG de Polytech avec le chemin demandé. Par exemple, si on veut aller de la salle A222 à la salle C204 :

```
PlusCours("A222","C204")
```

Le programme nous renvoie le chemin pour y accéder.

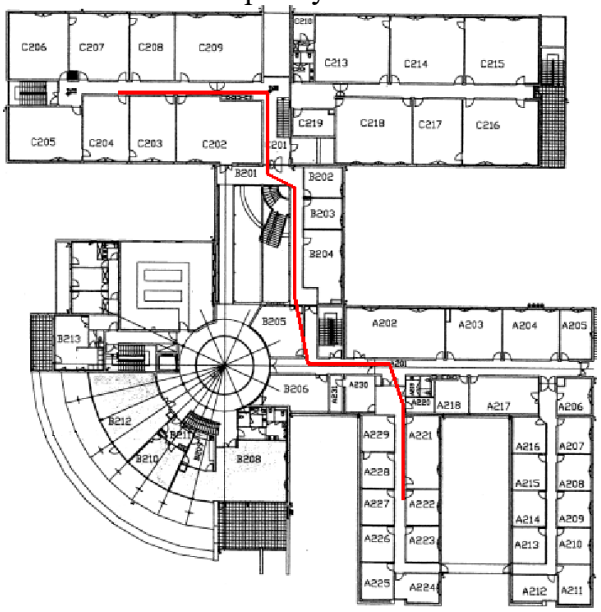

Figure 7 : Résultat de l'algorithme de plus court chemin pour aller de la salle A222 à la salle C204

c. Sauvegarde des données sur un cloud

Nous voulons que l'image générée soit disponible sur un smartphone. Nous devons donc héberger notre image sur un cloud afin qu'il soit disponible avec une URL. Pour cela, on utilise un service web nommé *imgbb* qui dispose d'une API permettant d'uploader une image sur leur serveur. Pour ce faire, on utilise une méthode par requête en lui précisant l'image et la clé API. La requête nous renvoie un fichier *json* qui contient toutes les informations. Il nous suffit de récupérer l'URL dans ce fichier.

d. Génération d'un QR code

Il nous reste plus qu'à générer un QR code grâce à la librairie python *qrcode*. `img = qrcode.make(url)` . On a donc comme résultat une image d'un QR code qui contient le lien vers le plan de Polytech avec le chemin pour aller à destination.

Figure 8 : QR code renvoyant au plan de l'exemple précédent

Difficultés rencontrées

Dans la création de cet algorithme, nous avons rencontré plusieurs problèmes. Un des premiers problèmes a été la création de la liste des points et des liaisons entre les points. Nous avons fait des fautes de frappe ou de lecture du plan dans la rédaction de la liste de coordonnées, ce qui nous a amenés à avoir des problèmes lors de nos tests. Pour régler ce problème, nous avons tracé un plan avec tous les points et les liaisons pour voir si elles étaient toutes correctement placées (*cf Figure 6 : Cartographie du premier étage de Polytech*).

Un autre problème est que notre programme, pour aller d'un point à un autre, nous faisait faire un détour. Ce problème est dû à la formule que nous utilisons pour calculer la distance du point suivant. Dans ce calcul, on ne prenait pas en compte la différence de distance des points possible et donc un point plus éloigné du point actuel était privilégié alors que ce n'était pas forcément le meilleur choix. Pour régler ce problème, dans la formule de calcul de distance, nous avons pris en compte ce facteur pour que le choix du point soit cohérent.

Dans la partie sauvegarde sur un cloud, nous avons recherché un service de sauvegarde disposant d'une API utilisable en python. Nous avons trouvé le service *imgbb* qui dispose de ces critères. Cependant, la documentation de l'API est très peu fournie. Il a été compliqué de comprendre comment elle fonctionnait.

7. Déplacement de Pepper

Pour le déplacement de Pepper, plusieurs niveaux de programmation ont été envisagés. Le premier serait tout simplement l'utilisation des blocs préprogrammés sur Chorégraphe prenant en entrée une distance et faisant avancer Pepper de cette distance. Le principe serait de recevoir les valeurs de distances à parcourir de l'algorithme de plus court chemin, et recevoir une information en cas de virage pour pouvoir tourner de 90°.

Au semestre précédent, nous avions réalisé un programme sur Chorégraphe permettant d'effectuer un suivi de mur. Le niveau deux consisterait à se baser sur ce programme. Le programme fonctionnant relativement bien, nous avons décidé de l'améliorer plutôt que d'en recommencer un autre. Le principal aspect à améliorer avec cette méthode était la distance parcourue. En effet, dû au déplacement latéral de Pepper pour rester à une distance constante du mur, nous pouvons apercevoir une perte de quelques dizaines de centimètres par rapport à la distance espérée. Pour essayer de corriger cette erreur sur la distance parcourue, nous avons réalisé un essai avec des distances de 2 mètres à 10 mètres en ligne droite et avec le moins d'obstacles possibles pour être dans des conditions idéales.

Distance	
Commande	Parcourue
2	2,1
2,5	2,67
3	3,21
3,5	3,68
4	4,19
4,5	4,68
5	5,18
6	6,22
7	7,23
8	8,15
9	9,17
10	9,96

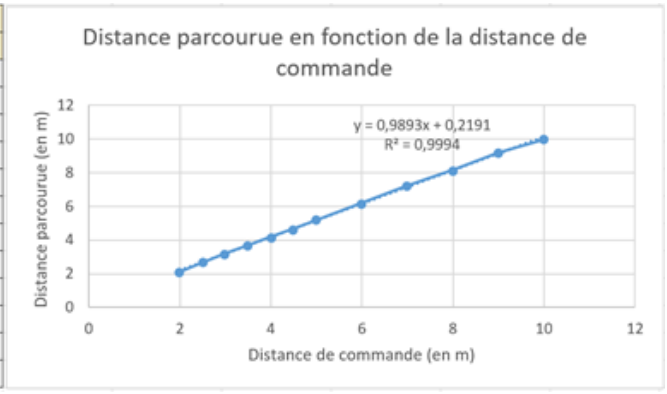

Figure 9 : Valeurs et courbe de tendance de la distance parcourue en fonction de la distance entrée en commande

Une fois les données récupérées, nous avons tracé le graphe correspondant aux résultats. Comme nous pouvons le voir ci-dessus, la courbe est assez linéaire, ce qui nous permet de tracer sa droite de régression linéaire et d'obtenir son coefficient.

Ainsi, nous pouvons corriger la distance entrée en commande à l'aide de la formule suivante : (distance_voulue – 0,21)*0,98 = distance_a_rentrer.

Pour plus de facilités, nous choisissons d'introduire cette correction directement dans le bloc de déplacement. Ainsi, nous pourrons tout simplement écrire en entrée la distance souhaitée. Après un test de cette correction, nous avons pu confirmer que la distance parcourue est maintenant beaucoup plus proche de notre objectif de distance.

La seconde étape était d'adapter notre bloc pour qu'il puisse être utilisé au sein du programme final. Pour cela, il a fallu modifier la manière d'entrer de la commande, pour pouvoir directement recevoir les résultats qui seront envoyés par le serveur après avoir fait tourner l'algorithme de plus court chemin.

Difficultés à régler au prochain semestre

Pour le prochain semestre, l'objectif principal sera de relier les différentes parties du programme final, qui sont pour l'instant toutes préparées indépendamment les unes des autres. Ainsi, pour relier la partie de déplacement avec le reste, il faudra trouver un moyen pour envoyer les bonnes informations à Pepper. En effet, le bloc de déplacement est conçu pour avancer d'une certaine distance en ligne droite, en suivant le mur gauche ou droit. Lors d'un virage à droite, il faudra que l'algorithme de plus court chemin envoie l'information de suivre le mur droit au bloc de déplacement, et inversement dans le cas d'un virage à gauche, pour que Pepper suive un mur spécifique et tourne avec lui lors du virage. Il faudra aussi que le serveur garde en tête les données envoyées à Pepper et que Pepper annonce lorsqu'elle arrive à destination pour qu'une fois le parcours terminé, nous puissions la localiser dans l'école.

Un autre aspect important serait de trouver un moyen de savoir dans quel sens se trouve Pepper par rapport au couloir. En effet, certaines destinations dans certaines circonstances nécessitent à Pepper de réaliser un demi-tour pour pouvoir suivre les instructions qui lui seront envoyées. Nous pourrions par exemple pour cela utiliser les données du précédent trajet stockées sur le serveur pour savoir d'où Pepper vient et donc dans quel sens elle se trouve.

8. Partie serveur

Le semestre précédent, nous avons mis en place une première version de notre serveur. Lors de ce semestre, nous avons finalisé la création de notre serveur qui est codé en Java, et qui permet l'échange des messages avec ses différents clients. Pour la suite du projet, nous aurons besoin de coder les différents clients de notre projet pour bien assurer la connexion entre les

différents acteurs de notre projet. Pour cela, il fallait coder principalement le client Chorégraphe pour Pepper et Nao, et le client Web pour la tablette de Pepper.

a. Client Pepper

Le premier client que nous avons développé était le client associé à Pepper. Vu le caractère one-thread de Python, nous étions obligés de créer 2 clients pour Pepper, le premier pour l'envoi du message au serveur, et le deuxième pour la réception des messages du serveur ou d'autres clients.

Nous avons rencontré plusieurs problèmes lors de la création de notre client, le premier problème était lié à la syntaxe du programme mis sur Aldébaran qui est différente de celle du Spyder. En plus, Aldebaran ne gère pas les accents et les caractères spéciaux, de plus qu'il y avait au début un problème de connexion du serveur avec le robot Pepper. Finalement, nous avons réussi à programmer notre client qui permet l'envoi et la réception des messages entre notre serveur et le client Pepper. Le code utilisé sera mis en annexes.

Au lancement de notre serveur pour tester notre client Pepper, nous obtenons :

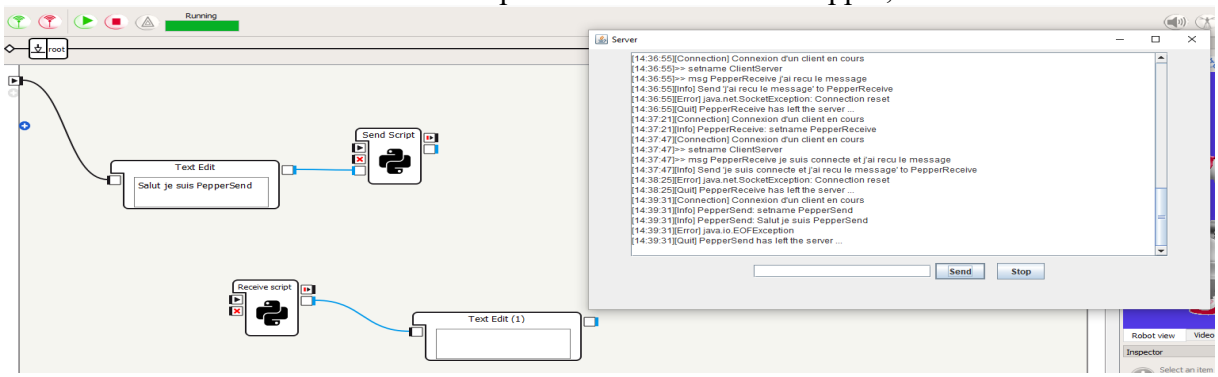

Figure 10 : Réponse du serveur pour l'envoi de données depuis Pepper

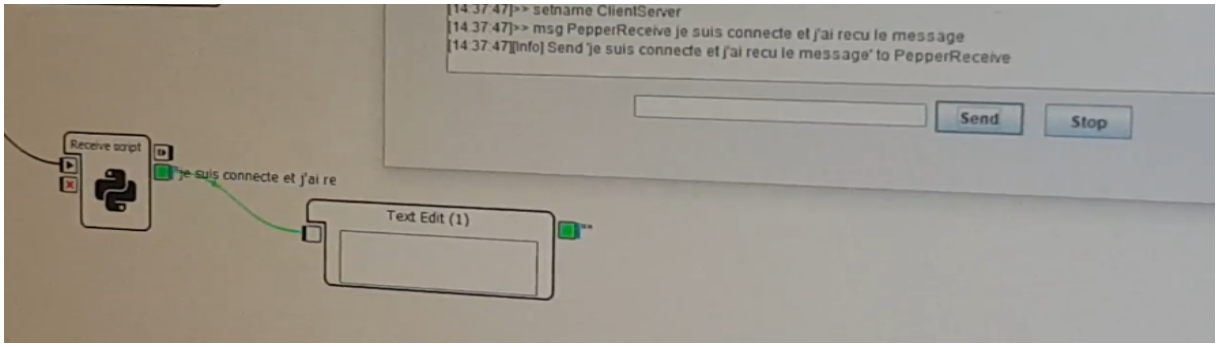

Figure 11 : Réponse du serveur pour la réception de données par Pepper

Notre client Pepper fonctionne dans les deux sens, et donc il peut échanger des messages avec le serveur.

b. Client Web

Le deuxième client que nous avons codé était le client Web, ce client est nécessaire pour notre projet pour assurer la connexion entre Pepper et sa tablette.

Le client Web est codé en PHP, Javascript et JQuery. C'est le client qui nous a pris le plus de temps à coder, le premier problème étant la connexion d'un client en Web avec le serveur, car même en testant sur une machine locale, le client n'arrivait pas à se connecter au serveur en localhost, il fallait donc obligatoirement changer l'adresse IP par l'adresse IP du serveur pour pouvoir tester notre programme. En revanche, le client Web est bien Multithread,

c'est-à-dire qu'il est capable de réaliser l'envoi et la réception des messages en même temps, donc il suffisait de coder un seul client Web qui réalise ces deux opérations.

Le deuxième problème pour un client Web était que notre page web ne pouvait pas recevoir plusieurs messages, ce qui fait qu'il était capable uniquement de récupérer ou envoyer un seul message, et au moment où il en reçoit un autre, il écrase la page en créant une nouvelle page Web associée seulement au nouveau message, et per ainsi l'information du premier message. C'est d'ici que vient la nécessité d'utiliser le module AJAX de la bibliothèque JQuery de JavaScript. Ce module permet la synchronisation de notre page Web à chaque envoi ou réception d'un message en conservant les messages précédents, et sans recharger notre page à chaque message. Nous avons finalement réussi à faire fonctionner notre code, et notre client marche dans les deux sens. Nous avons mis le code utilisé en annexes.

On teste alors la connexion de notre client Web avec le serveur :

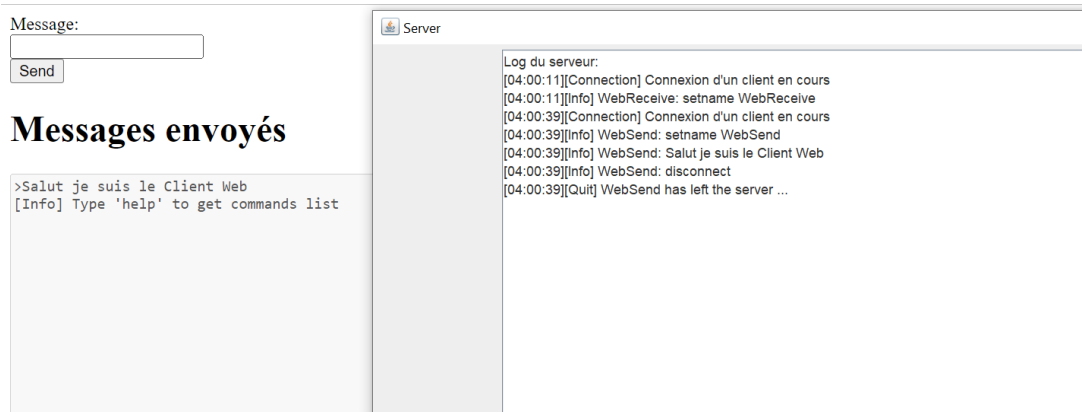

Figure 12 : Résultat pour l'envoi de message par le client web

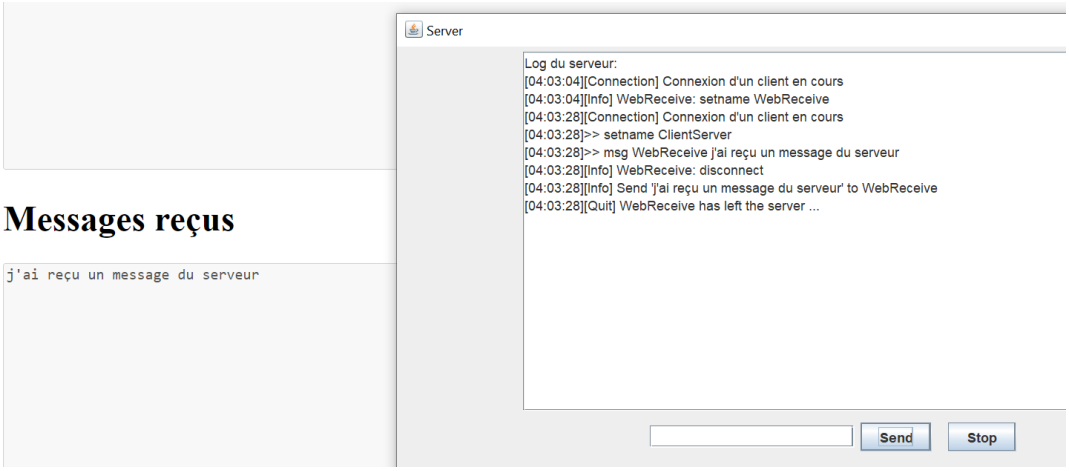

Figure 13 : Résultat pour la réception de messages par le client web

V. Conclusion

Nous avons rencontré plusieurs difficultés lors de ce semestre. La première difficulté était de se mettre d'accord sur un cahier des charges précis. En effet, il n'est pas toujours simple de rassembler des idées cohérentes pour un projet fonctionnel. D'un côté, Il faut avoir une liberté de pensée assez large pour agrémenter notre créativité afin de se fixer des objectifs ambitieux. Mais de l'autre côté, il faut pouvoir les résumer et les « canaliser » pour obtenir des objectifs réalisables. Nous avons également rencontré des difficultés mineures lors de nos avancements techniques. Toutefois, nous avons réussi à toutes les surmonter avec plus ou moins

d’aide et de temps. La prochaine difficulté que nous allons devoir relever est de mettre en commun les différents éléments de notre projet. En effet, nous devrons nous assurer que toutes les parties du code fonctionnent entre elles lorsque nous allons les regrouper.

Nous avons passé une bonne partie du semestre à réaliser un cahier des charges tenant en compte différents niveaux de difficultés de notre projet. En parallèle, nous avons également avancé la progression de nos différentes tâches commencées au semestre dernier afin de mieux se rendre compte des faisabilités et des limites de notre projet.

L’objectif principal pour la suite de notre projet est sa réalisation complète. Le but est que le robot guideur soit capable d’indiquer à un utilisateur son chemin et d’effectuer un parcours sur un étage. Pour cela, nous prévoyons de continuer nos avancées, comprendre nos erreurs et les rectifier pour avoir un projet fonctionnel d’ici la fin de l’année. Nous allons évidemment devoir s’appuyer sur le cahier des charges établi ce semestre.

VI. Annexes

1. Cahier des charges

Notre cahier des charges se découpe en différentes parties, qui constituent des objectifs classés par niveau de difficultés. Nous considérons ici que le niveau le plus bas (niveau 1) sont les objectifs les plus « faciles » à atteindre, très réalisables avec les capacités du groupe. Ces niveaux vont jusqu'au niveau le plus élevé (niveau 4) qui relèvent pour nous des objectifs les plus ambitieux. Nous souhaitons procéder d'une manière par échelon, c'est-à-dire que lorsqu'un objectif d'un niveau est atteint, il fonctionne parfaitement inclus avec les autres composantes. Nous devons nous concentrer sur le reste des objectifs du même niveau avant de se lancer dans l'élaboration d'une tâche qui relève d'un objectif contenu dans un niveau supérieur. Nous considérons alors un niveau fini ou atteint lorsque tous ses objectifs au complet sont fonctionnels. En procédant de cette manière nous seront plus aptes à avoir des résultats cohérents (et du même « niveau » d'attente). Par exemple, le premier niveau pour le repérage dans l'espace d'un utilisateur serait simplement d'afficher un plan à l'aide d'un algorithme de plus court chemin, ce qui est facilement réalisable, et le niveau le plus élevé serait que Pepper accompagne directement le visiteur à sa destination souhaitée.

Niveau 1

Le niveau 1 représente l'état « minimum » de notre démonstration. Les objectifs sont donc assez basiques, qui demandent seulement une bonne compréhension, manipulation de nos logiciels et implémentation des programmes réalisés. Le but principal est d'avoir une version fonctionnelle de base de nos différents modules et qui puisse remplir les fonctions minimales d'un guide ou d'un réceptionniste.

Plus précisément, les objectifs sont les suivants

- Le robot n'est pas forcément apte à se déplacer. En tout cas, dans ce niveau, il n'influe pas sur l'avancée des autres parties du projet car le déplacement sera traité indépendamment.
- Un serveur sera créé principalement pour établir la connexion entre différents clients (le deuxième Pepper, sa tablette et pourquoi pas un robot Nao) pour recevoir, traiter et renvoyer des données.
- La tablette de Pepper affiche seulement une page web fonctionnelle sur laquelle les plans des différents étages du bâtiment ainsi que les descriptions des salles sont inscrites.
- Au niveau de l'interaction utilisateur-robot, Pepper aura un discours de bienvenue et sera capable de réagir à des mots-clés et phrases simples prédéfinis.

Pour que tous ces points puissent marcher ensemble, nous devons gérer que la liaison entre le site web et la tablette de Pepper fonctionne correctement. Nous nous basons sur le bon fonctionnement de tous ces objectifs pour la fin de notre projet afin d'être capables de présenter un projet fini.

Nous avons pensé à un objectif optionnel pour ce niveau :

- Ajouter un bouton « aide » pour que l'utilisateur puisse appeler un technicien (en l'occurrence, un membre de notre groupe ou un professeur) lorsque Pepper est en danger ou semble être perdu.

Niveau 2

Le niveau 2 représente un deuxième état de notre projet qui a des objectifs un peu plus élaborés que le premier. Ces objectifs relèvent d'un peu plus de recherches et de connaissances. Si nous réussissons à atteindre tous les objectifs de cette version, le robot sera capable d'indiquer à un utilisateur son chemin et d'effectuer un parcours en boucle sur un étage.

Plus précisément, les objectifs sont les suivants :

- Le robot sera capable de suivre un chemin prédéfini dans un couloir ou sur l'étage entier.
- Le serveur pourra répondre à la demande d'un client et lui renvoyer le bon plan demandé. Par exemple, l'utilisateur demande un chemin vers sa destination à Pepper à travers la tablette. La requête est envoyée du serveur interne de la tablette au serveur général qui serait capable de renvoyer le chemin du point A (sa position) et un point B (la destination demandée par l'utilisateur) au serveur de la tablette.
- La tablette de Pepper affichera le chemin entre un point A (sa position) et un point B (la destination demandée par l'utilisateur à travers la tablette). Une option avancée serait d'afficher un QR code contenant ce plan, que l'utilisateur scanne pour l'afficher sur son téléphone portable.
- Le robot aura des émotions en fonction de plusieurs situations. Par exemple, nous choisirons de changer la couleur de ses yeux de bleus à rouges lorsqu'il sera perdu et programmerons ses mouvements en conséquence.

Pour que tous ces points puissent marcher ensemble, nous devons gérer que les liaisons entre le serveur et le robot Pepper et entre le serveur et la tablette de Pepper fonctionnent correctement.

Nous avons pensé à des objectifs optionnels pour ce niveau :

- Une notification apparaîtra sur la tablette lors d'un niveau trop faible de la batterie (< 25 % par exemple) pour indiquer aux utilisateurs d'alerter un membre du groupe pour mettre en charge le robot.
- Une alternance de mise en marche entre les deux robots Pepper afin d'assurer un service en continu lors de visites guidées.

Niveau 3

Le niveau 3 représente déjà une version de notre projet assez élaborée, qui serait idéal pour la fin de notre prochain semestre. Le robot doit être capable d'évoluer dans son environnement de façon optimale.

Plus précisément, les objectifs sont les suivants :

- Le robot sera capable de suivre une personne jusqu'à sa destination. En effet, nous avons remarqué que cette fonctionnalité existe dans le programme autonome de Pepper. Il nous paraît donc plus facile de le réutiliser que de lui demander de parcourir un chemin pas à pas dirigé par le serveur. Comme notre

but est que le robot soit le guide et pas l'utilisateur (qui ne connaît pas le bâtiment), nous voulons qu'il indique soit en affichant seulement le plan sur sa tablette ou à haute voix les directions à prendre pour arriver à destination.

- En utilisant un algorithme du plus court chemin, le serveur doit être capable de renvoyer le plan en fonction de la destination demandée par l'utilisateur. L'algorithme du plus court chemin sera traité et envoyé au robot à travers le serveur directement par une liste d'instructions. Le serveur pourra recevoir des informations des robots sur leur position (indépendante) afin qu'ils sachent où ils se situent les uns les autres.
- Le robot sera capable de comprendre une phrase complète et de réagir correctement, c'est-à-dire en donnant à l'utilisateur des informations pertinentes.

Ces objectifs sont possibles car nous utiliserons principalement la relation serveur-client fonctionnelle (requis au niveau 1).

Nous avons pensé à des objectifs optionnels pour ce niveau :

- Le robot devrait être accessible pour tous. Nous avons donc réfléchi à ce que Pepper puisse guider des personnes muettes et/ou malentendantes/malvoyantes. Par exemple, l'utilisateur pourra cliquer sur une option « je suis malentendant.e » afin que Pepper retranscrive son texte sur la tablette au fur et à mesure et que l'interaction entière se fasse par tablette tactile. Il pourrait aussi taper un message sur la tablette à l'aide d'un clavier que Pepper comprend pour les muets ou lors d'un problème de prononciation et Pepper ne comprend pas ce que l'on veut. La description pourrait être plus précise pour les déplacements, ou être accompagnée de des petits bips fréquents pour que la personne suive bien Pepper pour les malvoyants.
- Pepper pourrait changer de langue pour des visiteurs anglophones. Pepper est programmé en Anglais et en Français, mais nous voudrions que les visiteurs puissent changer ce paramètre eux-mêmes.
- Le robot pourrait occuper les visiteurs en chantant ou dansant lorsqu'on ne le sollicite plus au bout d'un certain temps.
- Un mode opérateur pourrait être intégré. Le robot serait téléguidé par une « manette virtuelle » affichée sur la tablette. Les utilisateurs pourront donc déplacer le robot si cela est nécessaire (encombrement du passage dans un couloir, risque de se rapprocher des escaliers, …)

Niveau 4

Le niveau 4 représente un aspect plus poussé qui serait une finalité pour l'objectif global de notre projet. Ce sont plus des perspectives pour les groupes qui nous succéderont ou des ouvertures de notre projet car nous ne pensons pas avoir le temps de les atteindre. L'idéal serait donc que le robot soit capable de guider les visiteurs dans un bâtiment quelconque.

Plus précisément, les objectifs sont les suivants :

- Pour un déplacement plus fluide, le robot guidera les visiteurs en suivi de mur. Le mieux serait qu'il suive les deux murs pour gérer au mieux sa trajectoire dans les couloirs, et qu'il se situe au milieu du couloir pour un déplacement plus naturel.
- Le serveur serait capable de récupérer les plans du bâtiment dans lequel les robots sont installés (en réserve que ces informations soient accessibles et mises

à jour par le bâtiment). De la même manière, la base de données des événements qui se déroulent dans les salles sera récupérée.
    - Le site web pour la tablette serait responsive design. Il pourrait également se mettre à jour en actualisant le plan affiché lors du déplacement du robot et la position des différents robots.

- Tout le projet serait adapté et donc programmé en boucle fermée.

Nous avons pensé à des objectifs optionnels pour ce niveau :

- Pepper pourrait changer de langue pour des visiteurs étrangers. Pepper possède plusieurs langues programmées de base, mais nous voudrions que des visiteurs du monde entier puissent l'utiliser.

2. Lien du repository Git

https://github.com/PolytechAnnecy-RobotiqueDeService/APP_RobotiqueDeService

3. Protocole algorithme de plus court chemin

V = Liste des points (vertex)
E = Liste des liaisons (arc) composer des coordonnées des 2 point à lier
G = (V,E)
C = Point courant à partir duquel on cherche la suite du chemin
D = Point de départ
A = Point d'arrivé
H = Liste des points par lesquels on est passé et leurs voisins
L = Liste des points voisins de C

On crée la Liste des liaisons("E") et Liste des points ("V").
On définit le point de départ ("D") et le point d'arrivée ("A").
On ajoute D à notre liste des points par lesquels on est passé ("H").
On regarde les points voisins du point courant ("C") en enlevant le point précédent et on en ressort une liste des points voisins("L").
Pour chacun des points de la liste L, on calcule la distance de ce point à A.
Le point le plus proche de A est choisi comme prochain point.
On se retrouve dans un cul-de-sac, s'il n'y a plus de points voisins non explorés.
Dans ce cas, on retourne au dernier croisement disponible et on recalcule les points voisins et la distance à A pour en ressortir le point suivant.
On ajoute ce point dans l'historique H.
On répète ces étapes (les 6 dernières étapes) jusqu'à arriver à A.
On renvoie la liste des points par lesquels on doit passer pour arriver a A.
On dessine sur la carte le chemin.

Pour la création des liste E et V :

- On quadrille un plan de Polytech (44 cases par 44 cases) pour avoir des coordonnées.
- On définit manuellement V (salle de cours, de TP, intersection…) et on en déduit les E.

Pour repérer le point voisin :

- On lit la liste E pour trouver le ou les arcs dans lequel C apparaît.
- On prend le ou les points associés et on les met dans une liste L.

Pour calculer la distance à A :

- On commence par récupérer la liste L, pour chaque point (appelé P) de L, on calcule : Distance = P2+A2
- On garde le point P avec la plus petite distance à A.

Pour trouver les croisements disponibles :

- On regarde dans H le dernier point par lequel on est passé et à qui il reste des voisins non explorés.
- On change notre point actuel par ce point de croisement.

Pour l'historique :

- On prend C pour le mettre en historique et on l'ajoute a H.
- On ajoute aussi les points voisins non explorés de C (tous les voisins sauf le point d'avant, qui est le précédent point de H)
- On enlève aussi, dans la liste des voisins du point précédent, le point A.

Pour afficher le plan :

- Utilisation de la librairie PIL pour Image et ImageDraw.
- On ouvre le plan de Polytech.
- On définit le quadrillage.
- On trace des traits entre les différents points de H.
- On sauvegarde le plan.

4. Code du client Pepper

```
    def onInput_input(self, msgIn):
        #Importation dépendance
        import socket
        import sys
        import os
        import struct
        #Connexion au serveur
        host = "localhost"
        port = 39039
        try:
            #Initialisation
            main_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            main_socket.connect((host, port))
            print("Connected to {} using port
{}".format(host,port))
            msg= b"setname PepperSend"
            size = len(msg)

            main_socket.send(struct.pack("!H", size))
            main_socket.send(msg)
            msg_in = main_socket.recv(1024).decode('utf-8')
            print(msg_in[2:])
        except OSError:
            print("[Erreur] Timeout\nVérifier que le serveur
est bien démarré et réessayer !")
            os.system("pause")
            sys.exit()


        message = b""
        msg_out = ""
        message = msgIn #Récupération du message à l'entré du
bloc
        if(message == ''):
            message = "None"

        message = message.encode()
        size = len(message)
        main_socket.send(struct.pack("!H", size))
        main_socket.send(message) #Envoie du message au serveur

        msg_in = main_socket.recv(1024).decode('utf-8')
#Réception de la réponse du serveur

        self.output(str(msg_in)[2:]) #Envoie de ma réponse sur
la sortie "output" du bloc
```

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
    def onLoad(self):
        pass
    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        #Importation des dépendances
        import socket
        import sys
        import os
        import struct
        #Connexion au serveur
        host = "localhost"
        port = 39039
        try:
            #Initialisation
            main_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            main_socket.connect((host, port))
            print("Connected to {} using port
{}".format(host,port))
            msg= b"setname PepperReceive"
            size = len(msg)
            main_socket.send(struct.pack("!H", size))
            main_socket.send(msg)
            msg_in = main_socket.recv(1024).decode('utf-8')
            print(msg_in[2:])
        except OSError:
            print("[Erreur] Timeout\nVérifier que le serveur est
bien démarré et réessayer !")
            os.system("pause")
            sys.exit()

        #Boucle qui laisse pepper en mode réception
        msg_in = b""
        while(msg_in != "stop"):
            msg_in = main_socket.recv(1024).decode('utf8')
            self.output(str(msg_in)[2:]) #On envoie le message sur
la sortie "output" du bloc

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up
as the box is stopped
        self.onStopped() #activate the output of the box
```