Département de génie informatique et de génie logiciel

INF8970 – INF8980 – INF8985

Projet final en génie informatique

Rapport de mi-session

Développement d'un environnement OpenAi/Gym pour simulation de réseaux électriques intelligents et soutenables — Projet Mila groupe de recherche Lesage-Landry

Équipe No 1

Ghazi Ben Achour
Alex Hua
Johnny Khoury
Alison Nemr
Victor Vergeau
Mohamed Zakaria

24 février 2023

Table des matières

1. Propositions de solutions (Q2.2)

1.1 Descriptions de quelques solutions possibles

Le groupe de recherche Lesage-Landry qui fait partie du Mila, soit l'Institut québécois d'intelligence artificielle, a fait appel aux étudiants de 4e année de Polytechnique Montréal, afin de travailler sur un système de contrôle de fréquence de climatisation pour les bâtiments afin de faire l'équilibre entre la production et la consommation d'énergie en variant très légèrement la température des maisons avec des climatiseurs. Cette problématique d'actualité et reliée au développement durable a interpellé notre équipe de 6 étudiants pour répondre à cet appel. En effet, non seulement le projet nous intéresse grandement en raison de ce que le projet de recherche cherche à accomplir, mais également en raison des tâches décrites dans le document de vision. Effectivement, avant d'avoir répondu à leur appel, nous avons fait une analyse du projet qui était décrit dans le document de vision fourni sur la page Moodle du cours INF8970. Une grande partie du projet se concentre sur la refonte de l'interface utilisateur, mais une autre partie aussi est concentrée sur l'architecture du code et, si possible, l'implémentation de fonctions. Le but principal est de rendre ce projet de recherche académique extensible pour que d'autres chercheurs-contributeur puissent facilement et rapidement se l'approprier. Ce projet touchera donc à plusieurs connaissances que nous avons acquises durant notre parcours à l'École Polytechnique de Montréal, en plus d'en développer de nouvelles.

Le temps faisant partie des contraintes qui nous ont été imposées, le projet sera à remettre le 20 avril 2023. Cela inclut le code pour le client, les rapports ainsi que l'oral qui devra être fait devant l'enseignant et le client. Il y aura deux grandes parties au projet au niveau académique: une remise de mi-session, qui comportera également un oral, et une remise de fin de session. Dans les deux cas devra être remis un rapport écrit et présenté oralement devant l'enseignant et le client. Les attentes du client devront être atteintes pour nous assurer de la réussite du projet. Afin de déterminer les objectifs de notre équipe au niveau de ce projet, nous avons également demandé au client ses objectifs pour nous donner une meilleure idée de sa vision. Le client a spécifié que son logiciel est surtout destiné à des chercheurs en génie électrique et énergique, pour la gestion de la consommation électrique des bâtiments. Il veut que son logiciel devienne la source et le produit incontournable pour la gestion de climatisation des bâtiments. Ainsi, nous nous sommes donnés comme but, en tant qu'équipe, de répondre à ses attentes. Notre objectif est de rendre le client fier de son produit et de présenter un logiciel qui sera facile d'utilisation, donc avec une interface utilisateur élégante, mais également avec une architecture pour la logique et le serveur[3] bien définie.

Après avoir tenu la première rencontre avec le client, celui-ci nous a présenté différentes visions qu'il avait sur le projet. Il nous a en effet spécifié qu'il préférerait que tout se fasse en Python, incluant l'extension de l'interface existante qui utilise la librairie de rendu graphique Pyglet. Toutefois, après avoir analysé la situation en équipe, nous

avons suggéré d'autres solutions qui aideraient le client à atteindre ses objectifs. En effet, le client a spécifié qu'il voulait que le logiciel sur lequel nous travaillons devienne le modèle à suivre pour ce type de projet. Donc, pour tout ce qui est en lien avec le contrôle énergétique des bâtiments par le chauffage et la climatisation, le but du client est que le code sur lequel nous travaillons puisse devenir une référence. Ainsi, afin d'atteindre ce but, nous avons proposé au client l'utilisation d'outils modernes mis à notre disposition afin d'avoir de bonnes fonctionnalités présentables en utilisant des technologies modernes que l'équipe peut exploiter au maximum. Nous comprenons que le client a voulu que nous utilisions Pyglet pour l'interface en raison des chercheurs en génie électrique qui ont l'habitude de travailler avec du Python, mais en faisant cela, nous nous limitions beaucoup au niveau des possibilités que nous pouvons atteindre avec ce logiciel.

Ainsi, nous avons parlé avec le client pour lui présenter une meilleure solution, car notre rôle est de répondre aux souhaits du client, mais également de lui présenter toutes les solutions possibles et de le guider vers la meilleure. Ainsi, nous avons dit au client que faire une interface en se basant sur Pyglet, qui est une librairie disponible en Python, mais qui est très limitée au niveau des possibilités pour l'interface utilisateur et qui ne marche pas toujours sur Windows, ne serait pas la solution idéale. Nous lui avons dit qu'en utilisant les langages HTML, CSS, TypeScript et en utilisant également le cadre de travail[4] Angular Material nous serions en mesure d'atteindre les objectifs beaucoup plus rapidement et plus facilement. De plus, nous lui avons parlé des avantages que cela lui apporterait, puis nous lui avons laissé le choix. Il a accepté d'y aller avec la solution présentée pour l'interface utilisateur.

Ensuite, au niveau du serveur et de la logique, nous avions également différentes solutions en tête. Principalement, cela concernait la façon de séparer les fichiers de code et notre façon de travailler en équipe. La première solution était de faire deux projets différents, soit un pour l'interface utilisateur et un autre pour la logique et le serveur. En ayant un projet qui regroupe l'interface utilisateur et la logique et le serveur, cela permettrait à toute l'équipe de garder le fil sur l'état d'avancement du travail et sur les tâches de tout le monde. Ainsi, cela donne une meilleure idée sur le travail qui est fait. Toutefois, cela veut également dire qu'il faut s'assurer que deux personnes ne travaillent pas sur la même branche sans quoi il pourrait y avoir des conflits qui nous feraient perdre du temps. La seconde solution proposée permet d'éviter cela, sauf que lorsque nous travaillons dans une équipe décentralisée comme c'est le cas pour nous, il ne s'agit pas de la meilleure solution, car l'équipe est divisée en deux sous-groupes indépendants, soit un groupe pour l'interface utilisateur et un pour la logique et le serveur. Bien que cela puisse réduire le nombre de conflits sur Git que nous pouvons avoir, il ne s'agit pas de la meilleure approche pour le type d'équipe que nous sommes.

1.2 Explications du rejet de certaines solutions ou de facteurs justifiant le choix de celle retenue

Au niveau des solutions que nous avons présentées, le plus gros dilemme que nous avions en tant qu'équipe était le choix entre l'utilisation de Pyglet et Angular Material. Comme mentionné précédemment, nous avons fait le choix, avec l'accord du client, d'y aller avec Angular Material. Le choix d'utiliser Pyglet était considéré, puisque le client préférait, au départ, utiliser cette technologie au lieu d'Angular Material. Pyglet a été rejeté pour plusieurs raisons. Tout d'abord, analysons pourquoi Pyglet était dans les choix du client. Ce dernier est utile, car il permet d'avoir tout le code en écrit en Python, incluant l'interface utilisateur. Le client a indiqué que la majorité des chercheurs en génie électrique ne connaissent que le langage Python. Nous comprenions donc son désir de continuer avec Pyglet. Toutefois, plusieurs aspects ont été considérés par notre équipe avant de faire le choix entre HTML/CSS/TypeScript et Pyglet.

Tout d'abord, nous avons regardé une étude sur les langages de programmation les plus utilisés par les développeurs. Parmi ceux-ci, en ordre d'utilisation, il y a JavaScript, HTML/CSS, SQL, Python, TypeScript, Java, puis les autres sans tous les nommer [1]. Ainsi, nous voyons que le langage Python, via Pyglet, n'est pas le choix le plus populaire en matière de langage pour le développement de l'interface utilisateur. Les langages tels que HTML et CSS sont beaucoup plus populaires. En première position, nous voyons que JavaScript est le choix le plus populaire des répondants. Il faut comprendre que TypeScript, langage que nous allons utiliser, est un surensemble de JavaScript. Il permet de rendre le typage beaucoup plus fort, puisque JavaScript ne prend pas en compte cela. Ainsi, pour un développeur qui connaît JavaScript, le TypeScript sera facilement lisible, puisqu'il n'y a pas vraiment de difficulté ajoutée entre les deux langages. Au contraire, le TypeScript permet de faciliter la compréhension du code en spécifiant le typage.

Par ailleurs, nous avons également parlé du fait que nous voulions utiliser Angular Material pour le développement de l'interface utilisateur. Ce cadriciel[4] fait partie des plus utilisés par les développeurs web [1]. Puisque nous sommes déjà familiers avec ce dernier en raison du projet 2, cela nous encourage à l'utiliser.

Aussi, puisqu'il faut faire de la refonte de code, nous pensons donc qu'il est mieux de le refaire en utilisant les langages et les pratiques les plus utilisés aujourd'hui par les développeurs web. Faire le développement sur une interface web nous permet de mieux adapter le code au goût du jour. Nous ne pensons pas qu'il est idéal de continuer la programmation en utilisant ce qui est déjà existant, car plusieurs pratiques sont défaillantes. Par exemple, plusieurs fonctions font des dizaines de lignes sans vraie structure, puisque le code a été développé de façon itérative. Il manque également de la documentation et cela rend la tâche plus difficile pour une personne qui tenterait d'ajouter du code.

Pour poursuivre, la version de Pyglet qui a été utilisée n'est pas la plus récente. Dans le code fourni par le client, certains rendus graphiques utilisent OpenGL, mais il faut savoir que cette librairie n'est pas adaptée pour le produit que nous avons présenté au client. Il faudrait donc passer beaucoup de temps à coder plusieurs éléments un à la fois pour avoir ce que nous avons présenté, au lieu de pouvoir les utiliser directement avec Angular Material. Le travail prendrait donc facilement 2 ou 3 fois plus de temps.

Pour toutes ces raisons, qui ont également été présentées au client, nous avons choisi, avec son accord, d'y aller avec le développement sur l'interface web, soit en utilisant HTML, CSS et TypeScript et en intégrant des composantes provenant de la librairie Angular Material pour le côté interface utilisateur. Cela ne s'applique pas aux solutions développées pour le côté serveur.

De plus, comme mentionné dans la section 1.1, nous avions également deux solutions pour la gestion des fichiers et du projet. Nous avons hésité sur le fait d'utiliser un projet ou d'en faire 2, soit un pour l'interface utilisateur et un pour la partie serveur et logique. Finalement, nous avons choisi d'y aller avec un projet, car même si en ayant deux projets différents nous avions moins de conflits à gérer, notre but n'est pas d'isoler l'équipe en 2 parties différentes. Nous voulons toujours que tout le monde ait une idée de ce qu'il se passe dans les deux côtés du projet. Pour cela, en ayant un projet où tout le monde peut travailler semble la solution la plus idéale.

2. Description de l'architecture du système retenue (Q2.3 — Q4.2 – Q4.3)

2.1 Détails de la solution retenue

Comme mentionné précédemment, nous ferons le développement sur une interface web. Cela veut dire que nous utiliserons les langages HTML, CSS, TypeScript. Pour ce qui est du cadriciel, nous utiliserons Angular Material afin de faciliter l'intégration de certaines composantes dont nous allons avoir besoin. Côté logique, nous avons utilisé les cadriciels FastAPI pour la communication avec l'interface, ainsi que des librairies telles que pydantic pour faire du contrôle et de la validation sur les structures de données. Le code est testé d'abord sur nos machines avant de faire le déploiement. Il sera évidemment important de tester notre travail une fois que celui-ci aura été déployé. La section 1.2 présente bien les détails de la solution, tant au niveau serveur et logique que pour l'interface.

2.2 Quelques diagrammes de classes ou de modules

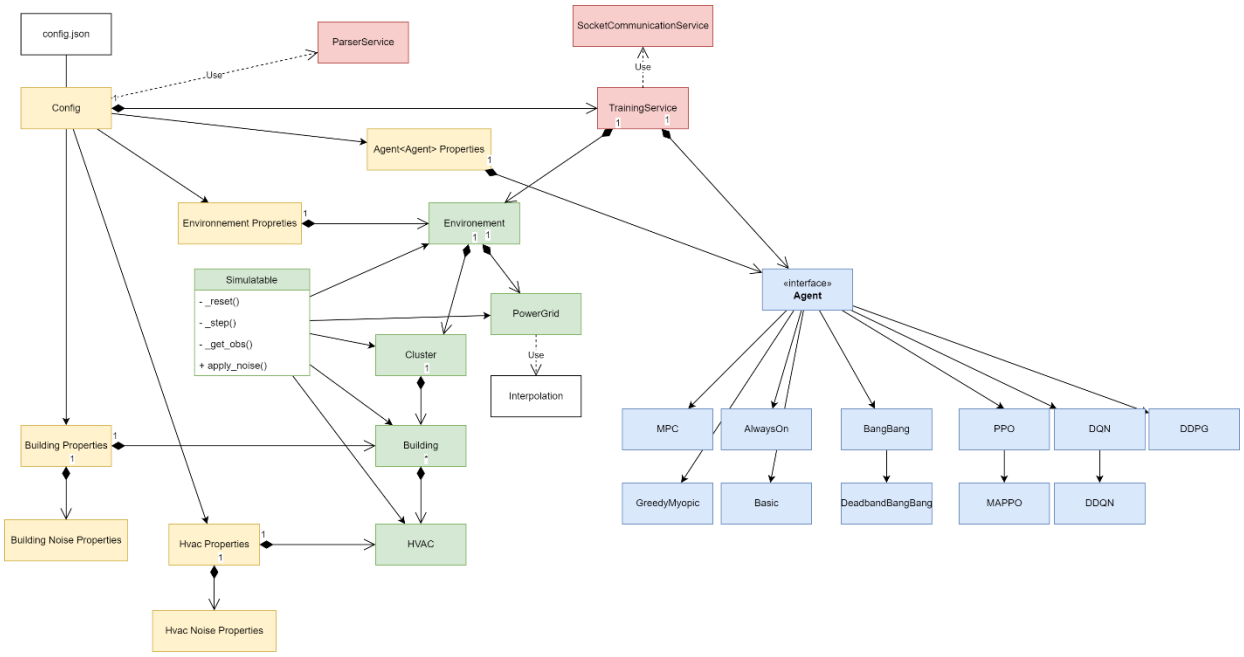

Figure 1: Diagramme de classe pour le côté serveur et logique

La figure 1 ci-dessus représente le diagramme de classe de la partie serveur et logique du projet. Tout d'abord, le côté serveur et logique comporte deux grosses composantes nécessaires pour son fonctionnement. Il y a l'environnement et l'agent (placés au centre du diagramme de classe de la figure 1). Il faut savoir que ces deux classes ne communiquent jamais ensemble de façon directe. Ils vont communiquer par le service « TrainingService ». Ainsi, la classe d'environnement générera des observations par un dictionnaire et l'agent a comme tâche de donner des actions à faire sur l'environnement en analysant le dictionnaire d'observation. À chaque saut de temps, l'environnement va générer de nouvelles observations. La communication entre ces deux classes est faite de façon à respecter la norme Gym qui était déjà implémentée lorsque le client nous a remis le projet en début de session. Les classes « Environnement » et « Agent » ne sont pas imbriqués l'une dans l'autre, elles sont complètement indépendantes. Les changements effectués dans la classe « Environnement » n'affectent pas l'implémentation de la classe « Agent »; c'est pour respecter les normes Gym que nous avons fait ce type de communication entre ces classes.

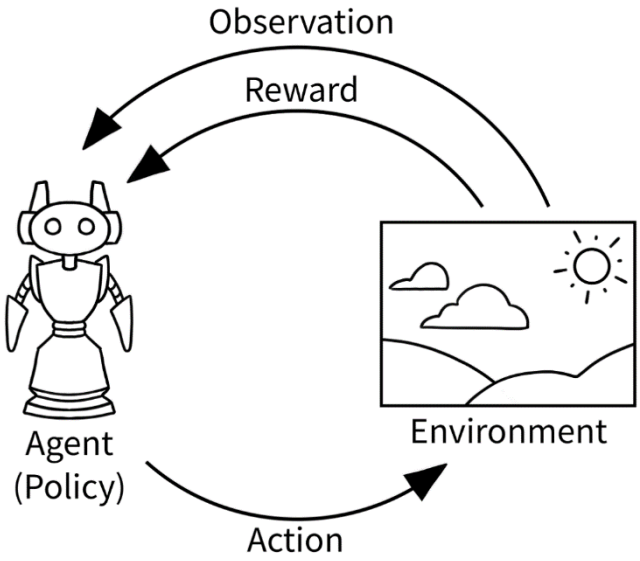

Figure 2: Norme Gym pour la communication entre un agent et un environnement [2]

La figure 2 ci-dessus explique l'interaction entre l'agent et l'environnement. Cette figure provient du site internet de la documentation de Gym et nous avons jugé pertinent de montrer cette norme afin de mieux visualiser la méthode d'échange d'informations entre ces classes. L'agent envoie donc des actions à l'environnement qui lui retourne des observations et des récompenses compte tenu des bonnes ou mauvaises actions que celui-ci a prises. Puis, de nouvelles actions sont envoyées à l'environnement et ainsi de suite jusqu'à la fin de la simulation.

Pour poursuivre, « Simulatable() » est une classe abstraite qui contient différentes fonctions propres aux normes Gym telles que « reset() », « step() » et « get_obs() ». Nous avons des classes et des fonctions nommées avec des termes anglais, car le code que le client nous a donné en début de session comportait des fonctions et des classes dans cette langue. Nous avons donc continué sur la même voie. Donc, pour chaque composante de l'environnement, soient « Cluster », « Building », « HVAC » et « PowerGrid », chacune d'entre elles implémente ces fonctions. La fonction « reset() » permet de réinitialiser la classe. La fonction « step() » permet de faire un pas dans le temps. Elle reçoit comme paramètre le dictionnaire d'action reçu par l'agent et retourne les dictionnaires d'observation et de récompense après avoir avancé d'un pas de temps. Quant à la fonction « get_obs() », elle permet de générer un dictionnaire d'observation puis le retourne. Cette méthode est par exemple appelée par le service « TrainingService » qui va par la suite fournir ce dictionnaire à l'agent, qui va à son tour donner les actions à faire à l'environnement. Chaque classe simule une partie de l'environnement, donc chaque classe hérite de la classe abstraite « Simulatable ». La classe « Environment », qui est composée de toutes les classes qui forment l'environnement, appelle les méthodes des classes sous-jacentes. Par exemple,

lorsqu'on appelle la méthode « step() » de la classe « Environment », celle-ci appelle les méthodes « step() » de la classe « Cluster » et « PowerGrid ».

Puis, nous avons aussi la classe Config. Cette classe ne contient pas de méthodes, mais seulement des attributs. En fait, elle s'occupe de lire un fichier de configuration « .json » qui contient toutes les informations de la simulation. Dans la figure 1, nous pouvons voir que cette classe contient les informations sur les agents (« Agent Properties »), les propriétés de l'environnement (« Environnement Properties » sur la figure 1) et les propriétés des bâtiments (« Building Properties » sur la figure 1). Celle-ci permet aussi de faire de la validation sur les données entrées dans le fichier de configuration. Le TrainingService reçoit donc la configuration de cette classe, puis initialise l'agent et l'environnement.

2.3 Diagrammes d'états et/ou d'interface usager

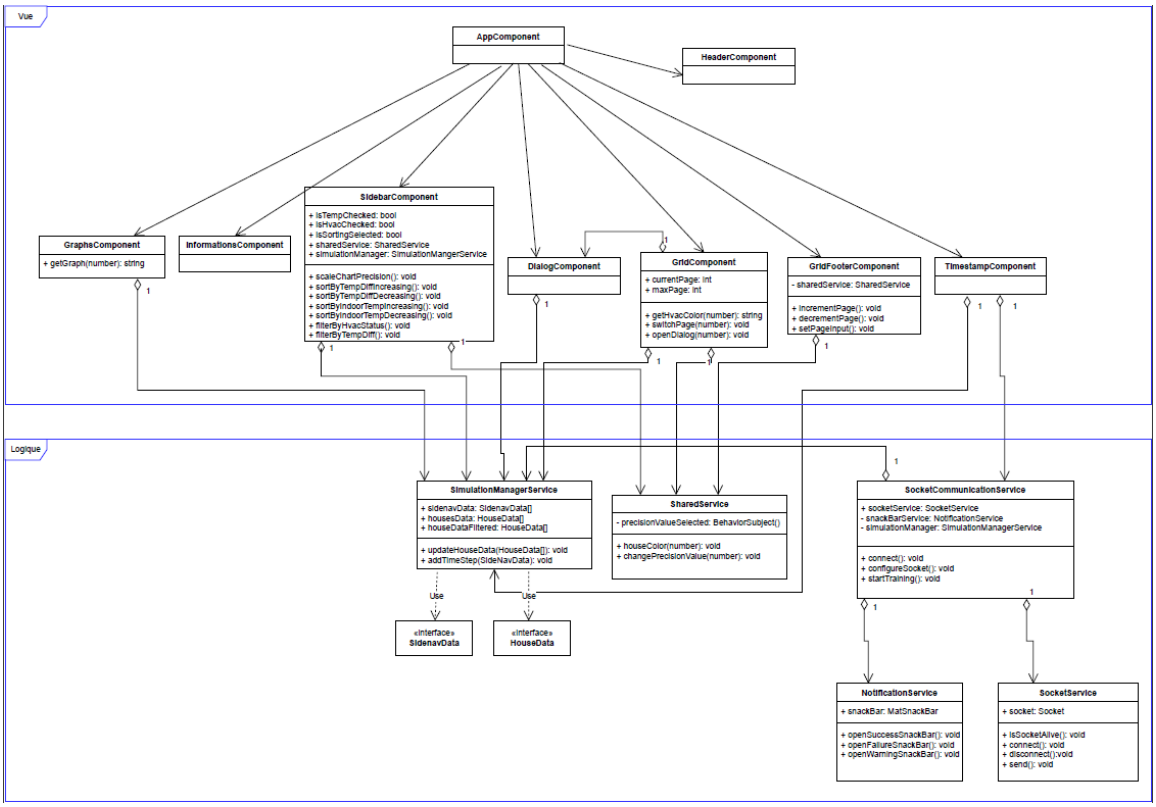

Figure 3: Diagramme de classe client interface usager

La figure 3 représente un diagramme de classe de notre interface usager. Nous pouvons voir qu'il y a 8 composantes pour l'interface utilisateur, soient « HeaderComponent », « GraphsComponent », « InformationsComponent », « SidebarComponent », « DialogComponent », « GridComponent », «

GridFooterComponent » et « TimestampComponent ». Toutes ces composantes ont été nommées en anglais pour assurer l'uniformité dans le code, car c'est cette langue qui a été utilisée, comme convenu avec le client. Ces 8 composantes se regroupent toutes sous la composante « AppComponent ». Chacune d'entre elles s'occupent d'effectuer une tâche bien précise que nous décrirons.

Tout d'abord, pour le côté Vue de l'interface, la composante « AppComponent » s'occupe de regrouper toutes les composantes. Parmi ces responsabilités, il y a l'allure de la page. Donc, elle s'occupe de gérer la disposition des éléments et de gérer l'affichage de ceux-ci. Ensuite, le « HeaderComponent » s'occupe de l'en-tête de la page. Elle contient le texte que nous voulons afficher. Pour ce qui est du « GraphsComponent », celui-ci s'occupe d'afficher les graphiques à l'aide des données qu'il reçoit du serveur. La composante « InformationsComponent » permet d'afficher les informations sur l'ensemble des maisons ou des bâtiments. Si nous voulons avoir les informations pour un bâtiment en particulier, c'est plutôt le « DialogComponent » qui gère cela. Le « SidebarComponent » est une composante qui contient plusieurs fonctions et qui permet de filtrer les maisons et les bâtiments. L'utilisateur n'a donc qu'à sélectionner les informations qu'il veut afficher et cette composante s'occupera de filtrer cela. Par ailleurs, dans notre interface utilisateur, nous affichons 100 maisons au maximum sur une page. Chaque maison occupe une case dans une grille. Cette grille est gérée par la « GridComponent ». Elle s'occupe également d'afficher des couleurs en fonction de l'état des « HVAC ». Le « GridFooterComponent » permet de s'occuper des pages que nous pouvons changer. Par exemple, si nous avons 1000 maisons, nous devrons avoir 10 pages, puisque chaque page affiche 100 maisons. L'utilisateur doit être en mesure de changer de page et la « GridFooterComponent » s'occupe de lui donner cette possibilité. Finalement, le « TimestampComponent » permet à l'utilisateur de naviguer dans le temps pour voir l'évolution de l'état des maisons.

Ensuite, pour le côté logique de l'interface, nous avons la « SimulationManagerService ». Ce service permet de chercher les informations du serveur et de les placer dans les différentes composantes en fonction de leur rôle. Le service « SharedService » permet d'échanger des éléments entre les composantes et le « SocketCommunicationService » permet de gérer la communication entre le serveur et le client. L'interface « HouseData » permet d'organiser les données des maisons pour les afficher dans le « DialogComponent ». Nous avons en effet plusieurs services et composantes pour le côté interface utilisateur et ils sont presque tous en communication les uns avec les autres. Certaines composantes implémentent des méthodes qui seront utilisées par d'autres et c'est pour cela qu'il est important de s'assurer du bon type de relation entre les classes et de bien comprendre le fonctionnement de chacun. Cela nous permettra d'écrire du code optimisé et permettre son entretien facilement.

2.4 Interfaces logicielles et/ou matérielles importantes

Nous n'avons pas implémenté d'interfaces matérielles, car notre projet n'en nécessite pas. Nous n'utilisons pas de capteurs ou d'objets connectés que nous devons prendre

en compte. Tous les éléments que nous devons traiter ont été présentés dans les diagrammes de classes des sections 2.2 et 2.3.

2.5 Simulation, norme ou modèles importants

Pour ce qui est des simulations que nous faisons, elle concerne principalement l'état des HVAC, donc des climatiseurs, en fonction de la différence de température des bâtiments. Pour tester si les couleurs marchent en fonction des différences de température, nous avons envoyé les vraies valeurs à l'interface pour voir si l'interface utilisateur réagissait bien et que les changements de couleur se faisaient réellement. Pour ce qui est des informations pour les maisons, nous pouvons aussi envoyer des données pour s'assurer qu'elles s'affichent bien. De cette façon, nous pouvons régler toute erreur possible. Cela permettra de nous faire gagner du temps. Puisque cette méthode a bien fonctionné, nous pensons aussi faire cela pour les graphiques qu'il faut présenter sur l'interface utilisateur. Nous pouvons simuler les graphiques et ainsi régler les problèmes, si nécessaire.

2.6 Description d'outils, librairies ou cadres de travail (*frameworks*) utilisés (Q5.1)

Pour la réalisation du projet, il faudra avoir en notre disposition les éléments de base pour le développement de code. Cela inclut un ordinateur pour chaque membre de l'équipe, des caméras et des microphones pour les rencontres à distance. Une connexion à internet sera également nécessaire pour pouvoir communiquer. Par ailleurs, nous utiliserons plusieurs connaissances que nous avons acquises durant notre parcours scolaire. Parmi celles-ci, nous comptons utiliser les connaissances que nous avons apprises dans les cours INF3995, INF3405, INF2990, LOG1000, LOG2410 et bien d'autres, pour ne nommer que celles-ci. Ainsi, ce projet final vient regrouper plusieurs connaissances académiques que nous avons acquises durant notre parcours scolaire, en plus de nous permettre d'en développer de nouvelles. En effet, il faut savoir comment gérer une équipe ainsi qu'un client, sans avoir d'intermédiaire entre nous. Il faut développer chacune des étapes de conception sans que celles-ci soient établies préalablement. Il y a donc une grande responsabilité qui repose sur nos épaules et pour être en mesure de répondre aux attentes, nous avons également appliqué les connaissances acquises dans les cours de HPR durant tout notre parcours scolaire. Parmi celles-ci, nous mettons un effort particulier sur la communication, le respect, le travail d'équipe et tout ce qui a trait à la gestion d'un travail d'équipe.

Lors du développement du projet, nous avons utilisé différentes technologies pour être en mesure d'effectuer le travail que nous avons accompli jusqu'à présent. Tout d'abord, commençons par l'environnement technologique que nous avons utilisé. Tous les membres de l'équipe travaillent sur Visual studio Code afin d'écrire le code, que ce soit pour l'interface utilisateur ou pour le serveur et la logique. Bien qu'il ne soit pas nécessaire que tous travaillent sur le même environnement technologique, nous

sommes tous habitués à celui-ci. De plus, il est plus facile de s'entraider si jamais un problème survient, car nous sommes tous sur le même environnement, soit un environnement que nous connaissons tous bien.

Pour ce qui est des librairies, Pydantic est utilisé comme une alternative à la librairie de classe de données fournies dans la bibliothèque standard de Python. Elle permet la validation de données et la gestion des paramètres. Elle permet le typage des structures de données et l'importation rapide de données json.

Ensuite, nous avons utilisé la bibliothèque Socket.io. Celle-ci est utilisée pour la communication bidirectionnelle entre un client et un serveur. En effet, elle s'appuie sur des événements lors de la communication entre ceux-ci. Elle se base sur les protocoles de Websocket et est utilisable avec le langage Python. Par ailleurs, elle s'occupe également de faire la connexion pour la communication entre le client et le serveur.

Pour poursuivre, nous avons également utilisé la librairie FastApi. Cette librairie sert pour l'interface utilisateur lorsque nous voulons utiliser des API avec le langage Python. Elle s'appuie également sur Pydantic que nous utilisons déjà pour faire la validation de données sur le fichier de configuration fourni par l'utilisateur, ainsi que pour structurer les données envoyées à l'interface.

De plus, nous utilisons le générateur de documentation de Python nommé Sphynx. Il permet de faire de la documentation du code Python pour les classes et ses méthodes avec les commentaires présents.

Finalement, la librairie Angular Material utilisée pour l'interface utilisateur offre des composantes[5] qui peuvent être utilisées par les développeurs. Ces composantes sont déjà codées et testées, alors nous pouvons facilement les intégrer à notre interface utilisateur.

Pour ce qui est des tests à effectuer, nous utiliserons différentes librairies pour les faire. Pour le côté serveur et la logique, nous utiliserons la librairie Pytest. Cette librairie nous servira, car le code est écrit en Python. Il s'agit donc d'une librairie qui répond à nos besoins en matière de tests à effectuer. Pour le côté interface utilisateur, nous utiliserons les librairies Jasmine, Chai et Mocha qui sont tous des librairies qui nous seront utiles pour tester, puisque nous utilisons Angular Material.

2.7 Quelques méthodes de test à appliquer pour valider la solution. (Q4.4)

Pour ce qui est des tests à effectuer pour vérifier le bon fonctionnement du système, nous testerons chacune des fonctions que nous avons écrites. Nous testerons autant le côté serveur et logique que l'interface utilisateur. Nous voulons nous assurer de fournir au client un produit complet et fiable sur lequel il pourra travailler dans les années à venir. Alors, le fait de tester chaque composante nous assurera de sa satisfaction et de l'atteinte des requis.

Ainsi, pour vérifier que le système fonctionne bien et que chaque fonction fait bien ce qu'elle a à faire, nous passerons le code sous différents processus de révision. Nous diviserons le projet en deux sections pour la révision et pour les tests. Tout d'abord, il y a les livrables à remettre que nous passerons sous révision ainsi que les fonctionnalités.

Pour les livrables, nous avons établi un plan de révision qui devra être respecté. Tout d'abord, nous nous assurerons de l'assurance qualité du code, et que les normes de codage soient cohérentes dans tout le code. Nous utiliserons un outil d'analyse de code nommé « linters » comme pylint, par exemple, afin de nous assurer que le code soit propre et lisible. Il faudra donc que tous les développeurs fassent de l'analyse en continu et écrivent des commentaires dans le code, lorsque cela est nécessaire. Comme mentionné précédemment, nous utilisons les « pull-requests » pour faire du contrôle lorsqu'une modification ou un ajout de code est fait. Lorsqu'un développeur demande à en faire un, il est de la responsabilité des autres développeurs de réviser le code et de fournir de la rétroaction sur le code écrit afin de l'améliorer si cela est nécessaire. Ce sera également le moment pour faire des suggestions. Une fois que tout cela a été fait, nous nommons quelques développeurs dans l'équipe, deux ou trois, comme responsables de la révision du code pour repasser à travers tout le code en le lisant et en s'assurant de la cohérence dans le code et de la qualité de celui-ci.

Du côté des fonctionnalités, tout comme le côté livrable, il faudra faire une analyse en continu. Une fois les fonctions écrites, nous ferons des tests unitaires pour chaque fonction écrite afin de nous assurer que chaque partie du code est fonctionnelle. En testant chaque fonction et chaque partie de code de manière individuelle, nous réduisons le risque des erreurs une fois tout le code mis ensemble. De plus, il est plus facile de détecter les erreurs dans les fonctions lorsqu'il est testé de manière incrémentale. Si jamais une fonction ne retourne pas les bons résultats, il est de la responsabilité de la personne qui a écrit la fonction de régler le problème. Nous utiliserons la librairie « Pytest » pour le côté serveur, puisque le code est écrit en Python. Pour le côté interface utilisateur, nous utiliserons des librairies comme Jasmine, Chai et Mocha afin de tester, puisque notre interface est implémentée en utilisant le cadriciel Angular Material. Nous créerons un fichier dans le projet qui sera spécifiquement dédié pour les tests. Nous surveillerons également la couverture des

tests avec les librairies de test et couverture fournie dans le projet par défaut Angular. Nous voulons également éviter les mauvaises surprises de dernière minute avant la remise. Nous souhaitons donc être en mesure de terminer les tâches requises au moins 1 journée, idéalement 2 jours avant les dates de remise. De cette façon, nous réduisons le risque de décevoir le client. La relation de confiance que nous développons avec le client est importante pour nous et sa satisfaction fait partie de nos objectifs à atteindre. Nous voulons être le plus professionnel possible et que le client aime son expérience de travail avec nous. En planifiant bien les tâches et les tests comme nous le faisons, nous nous assurons de sa satisfaction.

3. Gestion du projet

3.1 Identification des tâches principales du projet (Q3.3)

Après avoir discuté avec le client, nous avons identifié les principales tâches autour desquelles notre projet allait tourner. En effet, certaines tâches sont prioritaires et regroupent d'autres sous-tâches.

Tout d'abord, la première tâche importante sur laquelle nous travaillons est la refonte de l'interface. Le client avait déjà une interface présente lorsque le projet nous a été donné, mais il ne correspondait pas aux attentes et c'est une des raisons pourquoi le client a fait appel à nous. En effet, elle était très simple; c'est un peu une interface de base pour donner une idée générale de ce que le client recherche vraiment. À partir de cette interface de base, nous nous en sommes inspirés pour créer une interface beaucoup plus élégante et qui donne plus de possibilités à l'utilisateur. Le client avait spécifié au début de projet, lors de notre première rencontre, qu'il voulait que le logiciel sur lequel nous travaillons puisse devenir la référence à utiliser pour ce type de programme. Donc, il veut que, lorsque les chercheurs veulent utiliser un logiciel pour le contrôle de température de bâtiment et pour éviter le gaspillage énergétique, les gens se tournent vers le logiciel que nous développons. Donc, nous avons conclu avec lui que pour arriver à ce but, il faut que l'expérience utilisateur soit vraiment très bonne. Les principaux critères qui nous ont guidés pour le développement de l'interface sont: la simplicité, la facilité d'utilisation, la beauté/l'allure générale, la clarté et la façon dont les informations sont présentées.

Ensuite, une autre tâche générale sur laquelle nous avons travaillé est la refonte de l'architecture développée. Dans le code existant et qui nous a été fourni, il n'y avait pas d'architecture ou de logique présente. Le code écrit était incrémental, c'est-à-dire qu'il y avait des fonctions les unes après les autres, sans vraiment avoir de logique derrière. Notons par exemple l'absence de classe ou la présence de fonctions très grandes. Ces fonctions faisaient parfois des dizaines, voire des centaines de lignes de code. Nous avons donc repassé à travers tout le code et nous avons développé une architecture qui permettrait d'avoir une logique et une structure. Dans la section 2, nous avons présenté ces architectures. Elles permettent ainsi au code d'être plus lisible et il est plus agréable d'y travailler et d'ajouter des fonctions ou des fonctionnalités si voulues.

Par ailleurs, nous avons également écrit de la documentation. En effet, le client nous a mentionné qu'il aimerait que de la documentation soit présente afin de faciliter le travail des futurs chercheurs qui voudraient ajouter du code et pour permettre à quiconque d'utiliser le code, puisque le but de cette application est qu'il soit une source libre[1]. Nous avons donc fait cela. Nous avons décrit les grandes lignes de l'application et comment en faire utilisation. Nous avons également décrit le code de façon générale. La documentation est autant de bas niveau que de haut niveau. Donc, elle contient le rôle des fonctions, des classes, des fichiers et leurs types. Également, la documentation comportera des guides pour que les chercheurs puissent ajouter leurs propres agents, faire tourner la simulation et un guide de démarrage, entre autres.

3.2 Répartition des tâches et responsabilités dans l'équipe

Le projet sur lequel nous travaillons est assez gros et complexe. En effet, il comporte plusieurs parties et ce qui est existant a dû être revu et amélioré, car, comme mentionnée précédemment, la structure du code développée n'était pas correcte et ne respectait pas les normes de codage et de l'assurance qualité. Pour être en mesure d'accomplir les tâches requises, nous avons dû les séparer entre les membres de l'équipe. Pour nous y faire, nous avons regardé le projet de façon globale et nous avons établi qu'il y avait deux parties majeures au projet: l'interface utilisateur[2] et la logique/le serveur. Nous avons donc séparé notre équipe en deux en fonction des parties. Trois membres s'occupent de travailler sur la logique et le serveur et les trois autres membres s'occupent de travailler sur l'interface.

Dans la partie sur la logique et le serveur, Victor, Ghazi et Alex s'occupent de développer une architecture pour le code. De plus, dans cette partie, on cherche à revoir toutes les fonctions, les classes et la façon dont sont séparés les fichiers. Plusieurs fonctions dans le code sont très grandes et font plusieurs dizaines ou des centaines de lignes de long. Il n'y a pas vraiment de structure existante, il fallait donc en développer une. Dans cette partie, des fonctions plus courtes et qui sont spécifiques pour une tâche ont dû être implémentées. Il fallait aussi être en mesure d'envoyer les données à l'interface utilisateur pour être en mesure d'en faire sortir des graphiques et d'avoir les détails de chaque maison. Puisque le programme sur lequel nous travaillons comporte de l'intelligence artificielle, le client nous a spécifié qu'il était possible de réutiliser ces parties. Il n'était pas de notre responsabilité de revoir la logique derrière les fonctions comportant de l'intelligence artificielle. Nous pouvions donc les réutiliser et c'est ce que nous avons fait en nous assurant qu'elles étaient bien intégrées au reste du code que nous avons développé. Les principaux langages de programmation utilisés pour cette partie sont le Python, comme convenu avec le client.

Ensuite, pour ce qui est en lien avec l'interface utilisateur, Mohamed, Johnny et Alison s'en sont occupés. Dans cette partie, il fallait revoir la façon dont on présentait l'information à l'utilisateur. Nous voulions présenter le plus d'éléments possible sans trop en mettre, car il ne faut pas non plus perdre l'utilisateur et mettre trop d'informations inutiles. La présentation de la page et la séparation des éléments sont

toutes des tâches que nous avons réalisées. Les principaux langages utilisés pour cette partie sont le HTML, CSS et TypeScript. Nous avons aussi fait usage de Angular Material pour plusieurs éléments que nous avons ajoutés dans la page. Une fois la présentation de la page développée sur une maquette, une réunion en équipe a permis d'être sûr que tous étaient d'accord avant de la présenter à l'enseignant et au client. Une fois que tout cela a été approuvé, nous avons développé le code pour l'interface. Dans cette partie, on cherche aussi à recevoir les données du serveur pour présenter les graphiques et les informations de chaque maison.

Voici un résumé des principaux rôles de chaque membre de l'équipe:

Ghazi Ben Achour

- Développeur logique et serveur;
- Développeur interface utilisateur;
- Assurance qualité;
- Tests.

Alex Hua

- Développeur interface utilisateur
- Développeur logique et serveur;
- Assurance qualité;
- Tests.

Johnny Khoury

- Développeur interface utilisateur;
- Assurance qualité;
- Tests;
- Secrétaire lors des rencontres.

Alison Nemr

- Développeur interface utilisateur;
- Assurance qualité;
- Tests.

Victor Vergeau

- Animateur lors des rencontres;
- Développeur logique et serveur;
- Assurance qualité;
- Tests.

Mohamed Zakaria

- Développeur interface utilisateur;
- Assurance qualité;
- Tests;
- Gestion du temps de l'équipe.

Outre les tâches sur le serveur et l'interface utilisateur, chaque membre de l'équipe avait également des tâches supplémentaires. Par exemple, la rédaction des rapports, les résumés des rencontres avec le client et la gestion du répertoire Git sont également des tâches qui ont été attribuées. Puisque nous tenions des rencontres en équipe deux fois par semaine environ, parfois plus au besoin, il fallait un animateur pour ces séances ainsi qu'une personne qui gère le temps et qui prend des notes. Tout cela a également été divisé entre les membres de l'équipe.

Du côté des responsabilités, nous nous sommes tous mis d'accord d'être disponibles durant les périodes du cours, soit les mardis et les vendredis. Il était donc de la responsabilité de tous les membres d'être présent et si jamais il y avait un imprévu, il fallait aviser. De plus, lors des rencontres SCRUM que nous tenions durant nos rencontres, chaque membre avait la responsabilité de parler de ses tâches et de l'état d'avancement. Lorsqu'une tâche était assignée à quelqu'un, il devait s'assurer de la compléter dans les temps prévus afin de ne pas retarder les autres membres de l'équipe. Si une tâche était terminée à l'avance, il fallait en aviser les membres afin d'aider une autre personne si cela est nécessaire. Nous avons également identifié des valeurs qui nous sont importantes au début du projet. Pour nous, c'était la communication, l'effort, le respect et la réussite. Ces valeurs font partie de notre équipe et nous voulons nous assurer qu'elles soient présentes du début du projet jusqu'à la fin de celui-ci. Finalement, la ponctualité est également un important dans notre équipe, que ce soit pour la remise des travaux ou pour les rencontres que nous tenions.

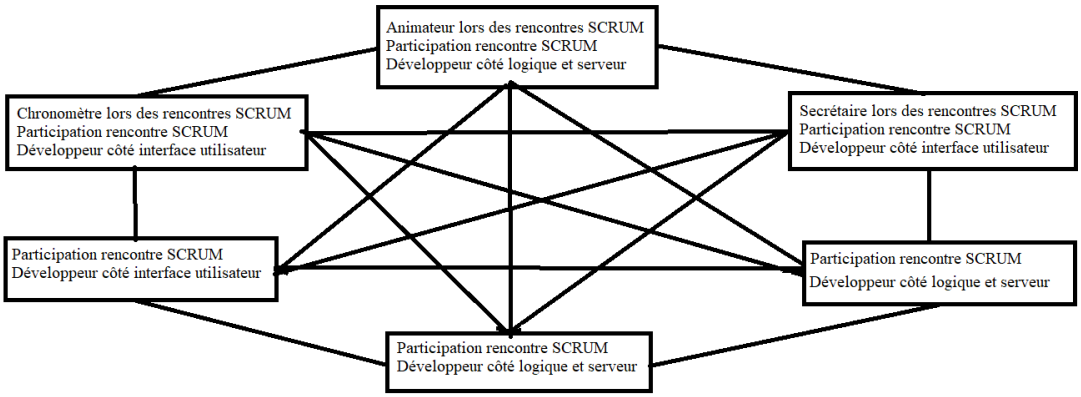

Figure 4: Rôle de chacun des membres de l'équipe

La figure 4 montre les différents rôles des membres de l'équipe. Nous travaillons avec le modèle décentralisé. Ce modèle a été favorisé, car le projet sur lequel nous travaillons est un projet complexe et très long. Il est préférable que tous les membres aient des tâches et que tous sachent l'état d'avancement du travail. Nous ne voulons pas qu'une personne se retrouve avec plus de responsabilités que les autres, alors ce modèle a été favorisé. De plus, cette façon de faire nous permettra d'atteindre nos objectifs ainsi de ceux du client. Par ailleurs, comme mentionnée précédemment, notre équipe est divisée en deux façons: développeur d'interface utilisateur et développeur de

serveur et logique. Chaque sous-groupe comporte 3 membres, puisque nous sommes une équipe de 6. Outre le code, chaque membre exerce également des responsabilités au niveau des rencontres que nous faisons dans l'équipe. Une personne s'occupe d'animer la rencontre, une autre de prendre des notes et de faire un résumé (secrétaire), un autre de chronométrer le temps de parole de chaque membre pour que les rencontres ne durent pas trop longtemps et aussi pour s'assurer que tous aient un droit de parole. De plus, tous les membres doivent s'assurer de prendre la parole pour dire l'état d'avancement de leurs tâches et de demander de l'aide, dans le cas où cela est requis. Cela est important, car, comme le montre la figure 4, tous les rôles des membres sont interreliés. Chaque membre est responsable de ses tâches, mais celles-ci peuvent influencer les tâches d'un autre membre de l'équipe, alors il est important que la communication dans l'équipe soit toujours présente.

3.3 Mesure sommaire de la progression du projet

Au stade où nous sommes lors de la remise du rapport de mi-session, soit le 24 février 2023, nous devrions être à la moitié du projet. Donc, la moitié des tâches requises devraient avoir été accomplies. La structure du projet déjà établie, le choix des technologies ne devraient pas changer ainsi que les architectures développées. De plus, le fonctionnement dans l'équipe devrait être clair et chaque membre devrait connaître ses responsabilités. Nous pouvons affirmer que cela est le cas dans notre équipe. En effet, nous estimons que nous en sommes à la moitié des tâches qui ont été accomplies. Principalement, la structure de l'interface utilisateur est bien avancée et le code pour la logique et le serveur avance à un bon rythme. Bien sûr, certaines tâches sont bloquantes, il faut donc attendre la fin d'une tâche avant d'en commencer une nouvelle. C'est, par ailleurs, le cas pour certains éléments de l'interface utilisateur. En effet, certains éléments de l'interface utilisateur ont besoin de recevoir ce qu'ils doivent faire de la part du serveur et de la logique, alors il faut attendre la fin de cela avant de continuer. Il est normal que le serveur et la logique prennent plus de temps que l'interface utilisateur, car les tâches sont un peu plus lourdes. Effectivement, il fallait revoir toute la structure et l'architecture du code. Bien que cela ait été fait en équipe et que tous se sont mis d'accord sur cela, il fallait également revoir plusieurs fonctions dans le code et réécrire plusieurs centaines, voire plus, lignes de code, et cela a pris beaucoup de temps. Pour la suite du travail, si nous continuons d'avancer au même rythme que nous avançons présentement, nous devrions être en mesure de fournir au client un logiciel qui répondra à ses attentes d'ici la remise finale, soit le 20 avril 2023. Nous pouvons donc affirmer que le travail avance bien. Nous sommes dans les temps sur le plan que nous avions mis en place; nous sommes dans les temps.

3.4 Mesures de révision du travail réalisé par les autres membres de l'équipe

Le fonctionnement du travail dans notre équipe tourne autour la méthode Agile, aussi appelé Processus Agile. En effet, cette méthode permet de décomposer le projet en plusieurs sous-parties afin de s'assurer que chaque partie est bien faite avant d'en commencer une nouvelle.

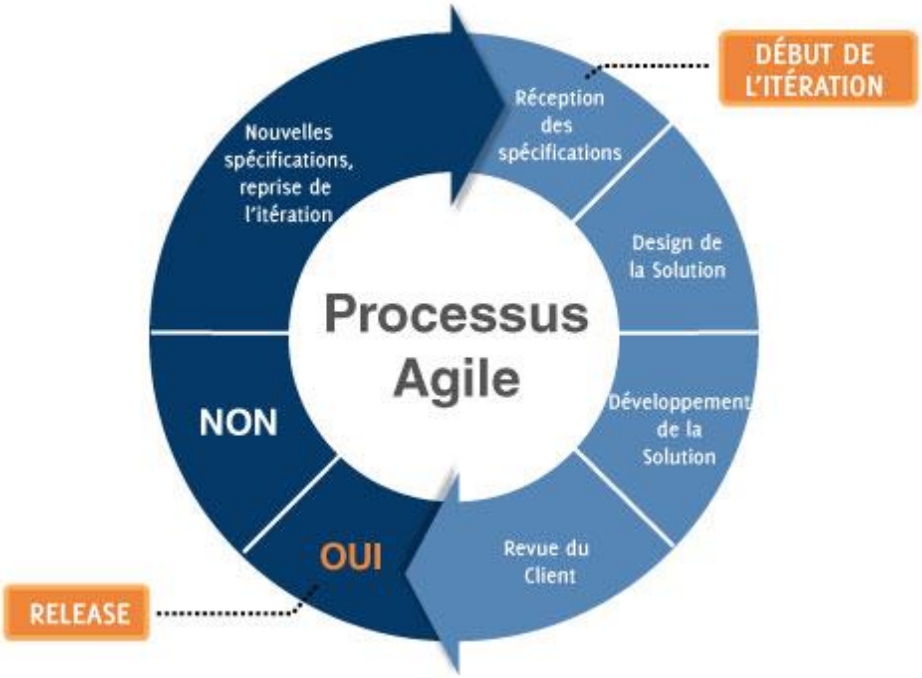

Figure 5: Processus agile utilisé dans l'équipe [3]

La méthode agile comporte différentes étapes. Tout d'abord, l'équipe reçoit les demandes du client et ce qu'il souhaite avoir. Ses attentes sont également spécifiées lors de cette étape. Une fois les demandes du client fournies à l'équipe, nous avons réfléchi sur une solution possible et développé une architecture qui conviendrait le plus au client en fonction des contraintes qu'il nous a données. Ensuite, lorsque toute l'équipe s'est mise d'accord sur cela, nous avons commencé à travailler sur nos différentes parties en nous séparant les tâches. Nous faisions cela chaque semaine. Donc, chaque semaine, nous tenons des rencontres en équipe et nous parlons de l'état d'avancement des tâches. Ensuite, si une tâche est terminée, nous nous en attribuons de nouvelles. Nos rencontres avec le client sont hebdomadaires; nous avons donc des remises[6] chaque semaine. Lors des rencontres avec le client, nous parlons de l'état d'avancement des tâches et il nous donne son avis sur le travail effectué en plus de parler de ses attentes. Si jamais quelque chose doit être amélioré, nous travaillons dessus pour le prochain sprint. Ces étapes recommencent à toutes les semaines pour chaque sprint jusqu'à la remise finale.

La révision du travail par les autres membres de l'équipe fait également partie des tâches et des responsabilités de tous les membres. En effet, tout ce qui a rapport avec le répertoire Git et la révision du travail doit être accomplie par tous. Par exemple, lors de « pull-request », au moins un membre de l'équipe doit faire la révision de code, idéalement tous doivent la faire. On donne ainsi notre avis sur le travail effectué avant

d'accepter de mettre tout le code ensemble. Par ailleurs, les tâches de base des développeurs devront être assurées par tous les membres de l'équipe. Cela concerne, par exemple, l'assurance qualité dans le code et l'écriture de quelques commentaires aux endroits où c'est nécessaire.

4. Progression jusqu'à la fin du projet et conclusion

4.1 Retour sommaire sur le travail déjà réalisé

Voici un bref résumé du travail accompli par l'équipe du début de la session jusqu'à la mi-session.

- Développement d'une maquette pour le modèle de l'interface utilisateur approuvée par le client;
- Implémentation de l'architecture de l'interface utilisateur;
- Commencer la documentation interne et externe du code et du projet;
- Commencer la refonte du code serveur et logique;
- Développer une architecture de code robuste et résiliente à l'ajout de nouvelles dynamiques et fonctionnalités;
- Être en mesure d'envoyer les vraies données du serveur à l'interface utilisateur en temps réel;
- Commencer le fichier de configuration[7];
- Être en mesure d'afficher les couleurs des maisons en fonction des différences de température.

Toutes ces tâches effectuées représentent environ la moitié du travail demandé. Comme mentionné précédemment, nous estimons que nous sommes dans les temps sur le plan que nous avons élaboré.

4.2 Identifications des facteurs encore mal connus qui pourraient poser problème d'ici la fin du projet (Q2.6)

Pour le moment, nous n'avons pas rencontré de problèmes majeurs. Que ce soit en lien avec le projet ou avec l'équipe, tout se déroule bien. Les objectifs et les valeurs que nous nous sommes fixés en début de session nous ont aidé à tous rester sur la même page. Tous les membres de l'équipe sont impliqués et investissent beaucoup de temps pour la réussite du projet. De plus, la communication est très bonne et le travail avance bien.

Bien sûr, un gros projet comme celui-ci nous donnera évidemment quelques difficultés. Par exemple, un des plus gros facteurs qui pourrait nous causer problème d'ici la fin du projet est le code déjà écrit sur le côté serveur et logique. En effet, comme mentionné précédemment, le code déjà écrit est très long et fait de façon incrémentale. Les normes de codage ne sont pas vraiment respectées et l'on peut facilement s'y perdre, surtout pour des personnes qui n'avaient pas d'idées sur le travail effectué. Bien que nous ayons tenu une rencontre avec le client pour nous expliquer le code et son

fonctionnement, la refonte du code côté serveur et logique, et l'implémentation de la nouvelle architecture développée sont de grosses tâches qui demanderont beaucoup de temps.

Bien que le projet puisse présenter quelques difficultés, nous sommes convaincus d'arriver à atteindre nos objectifs et ceux du client. Nous avons établi un plan à suivre lorsque nous faisons face à une difficulté afin de ne pas perdre beaucoup de temps. En effet, nous tentons de prévoir les différentes situations qui peuvent arriver afin de ne pas avoir de mauvaises surprises. Nous préférons nous préparer à faire face à ces situations. Nous avons, par ailleurs, fait plusieurs rencontres avec le client afin d'être sûrs de bien comprendre ce qu'il a déjà développé et d'éviter d'être bloqués. Le plan élaboré est présenté ci-dessous.

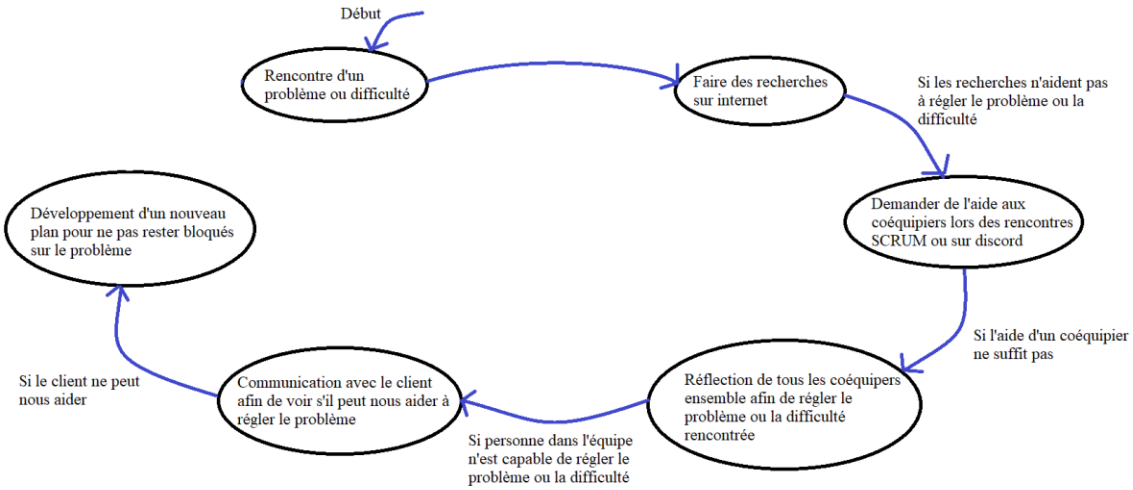

Figure 6: Plan élaboré par l'équipe afin de surmonter les difficultés

Nous estimons qu'il est rare, voire impossible, de se rendre à la dernière étape du plan, mais nous préférons prévoir les difficultés possibles. De plus, chaque étape du plan (chaque bulle) doit durer un temps raisonnable. Il n'est pas acceptable de rester bloqué sur un problème pendant plus d'une semaine. C'est pour cela que nous misons beaucoup sur la communication dans l'équipe.

4.3 Principales tâches à réaliser pour assurer le succès du projet et que l'équipe estime pouvoir compléter avant la fin (Q3.1)

D'ici la remise de fin de session, nous avons identifié quelques tâches restantes qu'il faudra être en mesure de terminer pour assurer le succès du projet et la satisfaction du client. Ces tâches sont:

- Terminer la refonte du code;
- Faire l'intégration de « Weights and Biases (WandB) »;
- Finir d'implémenter le fichier de configuration;
- Faire en sorte que le programme fonctionne avec plus d'un agent;

- Implémenter la fonctionnalité de navigation sur les différents pas de temps de la simulation;
- Terminer l'implémentation de l'architecture développée;
- Afficher les graphiques globaux sur l'interface utilisateur;
- Afficher des graphiques individuels de chaque maison sur l'interface utilisateur;
- Être en mesure d'afficher les détails de chaque maison dans les modales;
- Faire des tests unitaires et d'intégration plus exhaustifs.

Ces tâches restantes représentent la seconde moitié du travail demandé. Toutes ces tâches nous ont été demandées d'accomplir. Certaines d'entre elles sont sous-entendues avec les tâches qui ont été demandées. D'autres sont des tâches qui ajouteront une valeur supplémentaire au travail. Nous estimons être en mesure de toutes les accomplir d'ici le 20 avril 2023, soit la date de remise finale.

5. Références utilisées (Q3.2)

[1] Stack Overflow. (2022) Developer Survey. [En ligne]. Disponible: https://survey.stackoverflow.co/2022/

[2] Gymnasium. Gymnasium Documentation. [En ligne]. Disponible: https://gymnasium.farama.org/content/basic_usage/

[3] Siimec. (2023) Processus-agile. [En ligne]. Disponible: https://www.siimec.com/presentation/notre-approche/processus-agile/

[4] FastApi. FastApi Documentation. [En ligne]. Disponible: https://fastapi.tiangolo.com/

[5] Angular Material. (2010-2023) Guides. [En ligne]. Disponible: https://Material.angular.io/

[6] Pydantic. Overview. [En ligne]. Disponible: https://docs.pydantic.dev/

[7] Socket.io. (2023) Documentation. [En ligne]. Disponible: https://socket.io/

[8] Farama. (2022) Announcing The Farama Foundation. [En ligne]. Disponible: https://farama.org/Announcing-The-Farama-Foundation

[9] Polytechnique Montréal. (2021) Citer selon le style IEEE. [En ligne]. Disponible: https://guides.biblio.polymtl.ca/citer_ieee

6. Dictionnaire

1. Source libre ou logiciel libre (section 3.1): **Terme anglais:** Open source. **Définition**: Code qui est libre et disponible pour tout le monde.

2. Interface utilisateur (section 3.2): **Terme anglais:** Front end. **Définition:** Partie visible de la page web.

3. Logique/serveur (section 1.1): **Terme anglais:** Back end. **Définition:** Partie non visible à l'utilisateur, celle qui contient la logique des fonctions du code.

4. Cadre de travail (section 1.1) ou cadriciel (section 1.2): **Terme anglais:** Framework. **Définition:** Outil qui présente des éléments ou des composantes qui peuvent être utilisés directement.

5. Composantes (section 2.6): **Terme anglais:** Components. **Définition:** Partie du code qui permet d'effectuer une fonction précise.

6. Remises (section 3.4): **Terme anglais:** Sprint. **Définition:** Période où la fin de celle-ci correspond à une échéance, se terminant souvent par une rencontre avec le client.

7. Fichier de configuration (section 4.1): **Terme anglais:** Config file. **Définition:** Fichier servant à la configuration d'un programme.