Rapport Projet Fin de Semestre SI4

Groupe de Projet J
CASPAR-3
AL-2

Quentin CORNEVIN Hong JIN Didier MARTINI
Jérémy ROUMEGUE Garance VALLAT

23 janvier 2015

Sommaire

1 Problématique scientifique

1.1 Simulateur de positionnement

La carte que nous avons créée simule la position et les déplacements de plusieurs personnes, au sein d'un bâtiment que nous avons imaginé.
Dans la continuité du projet, ce que nous avons développé est une nouvelle fonctionnalité liée aux déplacements. Elle prend en compte le déplacement réel des personnes avec une localisation par satellite, système de localisation présent dans les smartphones.
Cependant, le réseau ne dépend pas de nous et peut de temps à autres être défaillant. De ce fait, cette méthode comporte des éventuelles coupures de réseau, et il est donc possible dávoir des pertes temporaires de localisation. La perte de localisation entraîne des incohérences, comme des téléportations ou des disparitions, et c'est là que notre système entre en jeu.
Le simulateur de positionnement va permettre aux utilisateurs de ne pas remarquer les éventuelles pertes de connexion et d'éviter les téléportations ou disparitions des personnes. On peut également lancer une simulation réaliste des étudiants et professeurs en paramétrant l'application.
Il faut donc mettre en place un système qui, dès qu'une perte de connexion est détectée, va remplacer et imaginer le chemin de plus cohérent pour l'utilisateur.

Comment pouvons-nous faire pour qu'un utilisateur ne remarque pas les pertes de connexion et que les positions trouvées soient cohérentes vis-à-vis du déplacement réel des personnes ?

1.2 Sécurisation du système

Le système est basé sur des échanges clients serveur. Il y a donc des données envoyées sur le serveur au travers du client, et si elles ne sont pas protégées elles peuvent, être interceptées et utilisées à des fins malhonnêtes.
Afin de répondre à ce danger, lápplication possède un système de connexion, et c'est seulement via cet accès qu'un utilisateur peut visualiser les différents déplacements des personnes et éventuellement de rejouer une simulation passée.
Il nous paraît donc important de protéger ce transfert autant dún point de vue du mot de passe, que dún point de vue utilisations des données de déplacement.
La sécurisation du système est donc un point capital de lápplication, car il va permettre d'assurer la sécurité sur les données présentes dans notre système, afin quélle ne puisse pas être interprétée ou décodée. Il permettra également de sécuriser l'accès au serveur, afin qu'une personne malveillante ne puisse pas bombarder le serveur de demandes de simulation, afin de le faire tomber ou de saturer notre système de stockage.
L'essentiel de la difficulté ici est donc de trouver un moyen qui soit le plus efficace avec un temps minimal.

Quel est le moyen le plus efficace, en rapport qualité de l'algorithme et temps pour protéger les données du serveur ainsi que la connexion entre le client et le serveur ?

2 Analyse critique de la solution

2.1 Notre solution

Notre solution devait répondre au problème du client ayant besoin d'un simulateur de positionnement protégé. Nous avons répondu à ce besoin grâce à une architecture sur trois niveaux. En effet, nous avons un client Java s'occupant de l'affichage, un serveur faisant les calculs et une base de données stockant les informations calculées par le serveur.

Notre client Java lit les informations de la simulation dans un fichier XML complété par l'utilisateur. Ce fichier peut contenir le nombre de personnes présentes dans la simulation, ainsi que l'endroit de la carte d'où ces personnes doivent démarrer, et enfin, la durée de la simulation.
Ces informations sont ensuite communiquées au serveur NodeJS par du JSON. Le serveur fait ensuite les calculs de déplacement. Il enregistre ensuite les informations dans la base de données et les renvoie au client.

2.2 Ses forces

Nous distinguons plusieurs forces dans notre application, à différents niveaux.

En terme d'architecture tout d'abord :
L'architecture du serveur est MVC(Model-View-Control), ce qui permet de pouvoir maintenir le code à jour facilement et aussi d'avoir une flexibilité de programmation importante : cela facilite le travail en équipe.
D'autre part, nous avons choisi de faire un serveur en JavaScript, pour profiter de la rapidité de calcul et d'exécution de ce langage. Cela nous permet de faire supporter à notre serveur une montée en charge importante, principalement bloquée par les capacités physique du serveur, qui est privé.

D'autre part, en terme de complexité, l'algorithme de notre simulateur (celui qui permet de relier deux points distants) est en $\mathbf{O}(n^2 * \log(n))$, ce qui n'handicape que peu la montée en charge, afin de pouvoir répondre au souhaite de notre client, de gérer les flux.

Nous tirons les autres forces de notre projet de son aspect réseau :
Par exemple, pour faire transiter les données nous utilisons le standard JSON, qui est un format facile d'utilisation, particulièrement dans notre cas où le client et le serveur ne sont pas programmés dans le même langage.
Pour faire communiquer ce client et ce serveur en tout sécurité, nous avons mis en place un protocole HTTPS qui chiffre les données échangées. Ainsi, des données volées sont inexploitables, et ce que reçoit le client ne peut pas être corrompu. Enfin, un utilisateur peut se connecter en toute sécurité sur notre application grâce à notre système de connexion chiffré localement en SHA-512. Le mot de passe ne peut donc pas être lu directement, par récupération de requête.

2.3 Ses faiblesses

La solution que nous proposons n'est cependant pas parfaite. En effet, bien que le format JSON soit un format standard, il est plus lourd qu'un format personnalisé pour l'application. Il va donc ralentir l'application et diminuer la capacité en charge de notre système. Il aurait été préférable de créer notre propre format.

Une de nos plus grande faiblesse est du coté client. En effet, nous avons décidé de programmer cette partie en Java. Ce choix de technologie nous a grandement limité aussi bien d'un point de vue esthétique que d'un point de vue performance. En effet, le client n'est pas très agréable et lent pour un grand nombre de personnes à afficher. De plus, cette partie du code a été seulement partiellement faite en MVC, l'entretien à long terme en est donc difficile.

Du coté du serveur, ce dernier est un serveur privé, donc limité en puissance pour des simulations trop complexes, aussi bien pour la communication avec le client que pour la communication avec la base de données. De plus, cette dernière n'a pas été orientée "Big Data", ce qui ne correspond pas pour la mise à l'échelle pour des simulations d'un grand nombre de personnes.

Enfin, notre simulation intelligente, avec l'algorithme A* est lourde au niveau calcul, elle prend de plus en plus de temps en fonction des paramètres choisis, malgré sa puissance.

3 Positionnement par matières

3.1 Complexité

3.1.1 Problématique Scientifique

Afin d'avoir un simulateur de position à la fois intelligent et performant, nous avons dû à de nombreuses reprises revoir la complexité de nos algorithmes. En effet, notre client ayant pour objectif de pouvoir simuler un grand nombre de personnes, nous ne pouvions nous permettre d'avoir des temps de calculs trop long.

Une autre problématique reliée à la complexité dans notre projet a été pour la sécurité, et le chiffrement des données à protéger. Comment protéger des données efficacement, tout en évitant de perdre trop de temps d'exécution ?

3.1.2 Réalisation

Pour notre simulateur intelligent, nous avons voulu faire des calculs de meilleur chemin, ce qui nous a conduit à utiliser l'algorithme A*. L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. Il est aussi la méthode la plus rapide pour trouver le meilleur chemin.

En terme de sécurité, nosu avons choisi de chiffrer les mots de passe des utilisateurs en les hachant avec la méthode SHA-512. Cette méthode transforme les informations qu'on lui envoie en un mot unique de 64 bits. Cette méthode a été introduite comme standard par l'administration américaine en 2002, et aucune faille majeure de sécurité n'a été démontrée à ce jour. Nous effectuons cette transformation localement dans le client, avant d'envoyer le résultat haché au serveur pour comparaison avec le contenu de la base de données.

3.1.3 Analyse critique de notre réalisation

On a une fonction heuristique

$$f(n) = g(n) + h(n)$$

et deux listes OpenList et CloseList. Si $h(n) = 0$, on effectue un parcours en largeur, parce qu'on peut utiliser $\mathbf{g(}n\mathbf{)}$ comme hauteur de nœud. La complexité est donc identique à celle d'un parcours en largeur.
Pour la complexité de l'algorithme A*, dans la boucle de parcours, on récupère un nœud de la OpenList, n fois. On commence donc nécessairement par $\mathbf{O(}n\mathbf{)}$. On a aussi besoin de trier ces données, et si on utilise le tri rapide, la complexité est $\mathbf{O(}n\log(n)\mathbf{)}$. Donc, pour la boucle, on a la complexité de $\mathbf{O(}n^2 * \log(n)\mathbf{)}$.
On peut aussi ne pas effectuer de tri, et au contraire, chaque fois, calculer directement la plus petite valeur de la OpenList. Donc la complexité est $\mathbf{O(}n^2\mathbf{)}$. C'est l'algorithme Dijkstra.
Mais l'algorithme Dijkstra est plus efficace seulement pour calculer la plus petite longeur du chemin entre le nœud initial et tous les autre nœud, il s'agit d'un parcours en profondeur. L'algorithme A* est pour trouver le meilleur chemin entre deux nœud ; il est un algorithme heuristique. Pour notre produit, on lui préfère donc l'algorithme A*.

3.2 Algorithmique et Structure de données

3.2.1 Problématique

Pour notre système de recherche de chemin, nous avons mis en place un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final, l'algorithme A*.

3.2.2 Réalisation

Il utilise une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin passant par ce dernier, et visite ensuite les nœuds par ordre de cette évaluation heuristique.

Dans un algorithme heuristique, l'évaluation de chaque nœud est très importante, différentes évaluations vont donner les résultats différents. L'évaluation heuristique est présentée par une fonction heuristique. Ici, on utilise

$$f(n) = g(n) + h(n)$$

f(n) est la fonction heuristique de nœud n. ***g(n)*** est la valeur réele pour le chemin entre le nœud initial et le nœud n. ***h(n)*** est la valeur heuristique pour le meilleur chemin entre le nœud initial et le nœud n. Pour la valeur heuristique, on utilise toujours ces fonctions :

— **Distance de Manhattan**(fonction heuristique standard)

$$h(x) = abs(xf - x) + abs(yf - y)$$

— **Ligne diagonale**
— **Distance Euclidienne**

Pour l'algorithme A*, on a besoin de deux listes.
OpenList : Au début, on met le nœud initial dans la OpenList et elle contient les nœud qui ne sont pas encore traités.
CloseList : Au moment du lancement de l'algorithme, la CloseList est vide et elle contient les nœud déjà traités.
Le déroulement de l'algorithme A* :
On met le nœud initial S dans la OpenList, on vide la CloseList, on lance l'algorithme :

1. Si la OpenList n'est pas vide, on prend le premier nœud n de la OpenList, si non, on termine l'algorithme.
2. Si le nœud n est le nœud final F, on a trouvé un résultat (on continue ou on termine l'algorithme). Si non, on va en étape 3.
3. On cherche tous les nœuds voisin du nœud n. S'ils ne sont pas dans la CloseList, on les met dans la OpenList, et puis on met le nœud S dans la CloseList et on calcule toutes les valeurs $\mathbf{f}(n)$ de nœud de la OpenList et les trie. On revient à l'étape 1.

Obtention du chemin : Dans la CloseList, on commence du premier noeud jusqu'au au nœud final F, si le nœud F-1 est le nœud père de F, il est dans le chemin. Quand on a terminé le parcours de la CloseList, on a trouvé le chemin entre le nœud initial S et le nœud final F.

3.2.3 Analyse Critique

C'est un algorithme simple, ne nécessitant pas de prétraitement, et ne consommant que peu de mémoire.

Dans notre produit, à cause de l'absence de la position réelle de l'utilisateur, on a besoin de l'algorithme pour simuler la trace entre deux positions sans que les utilisateurs se téléportent.

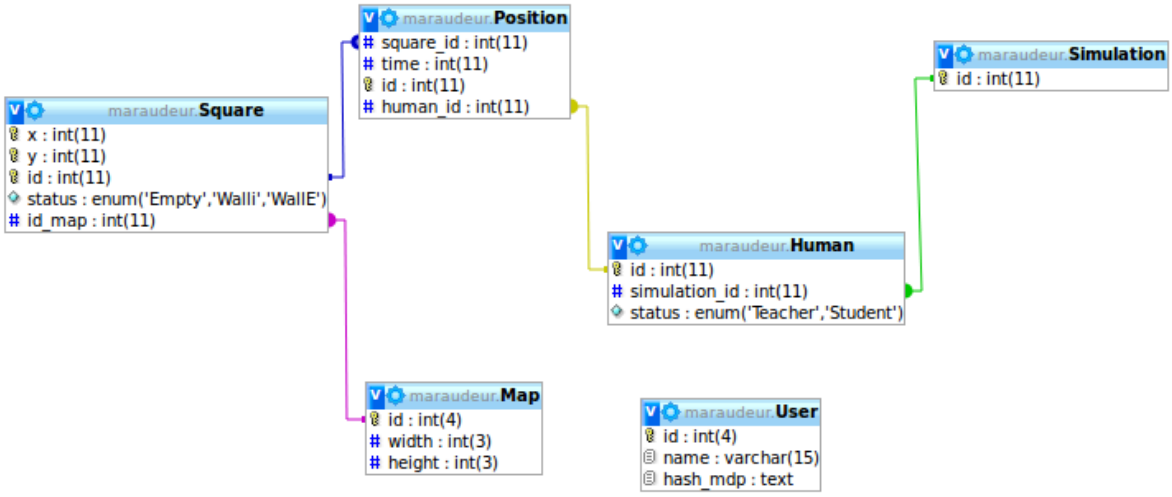
3.3 Bases de données

3.3.1 Problématique

Nous voulions pouvoir enregistrer nos simulations, afin de pouvoir les rejouer, en choisissant des paramètres précis, et pas tout ce qui a pu se produire. Il nous est donc apparu nécessaire d'avoir une base de données, afin d'avoir un vrai moyen de stockage des informations.
Nous avons choisi d'utiliser une base de données pour stocker les informations nécessaires à toute simulation : utilisateurs de l'application, personnes présentes dans une simulation, positions des personnes de la simulation et la carte.
Nous avons principalement fait ce choix pour sa simplicité d'utilisation. Nous devions également enregistrer les traces des différentes personnes, nous avons donc jugé que le choix d'une base de données "mySQL" était judicieux, car on peut récuperer simplement les données d'une personne précise, ainsi que le moment auquel un certain déplacement a pu être fait.
L'utilisation du mySQL a également eu l'avantage de simplifier l'accès à la durée d'une simulation, ou bien aux différentes personnes qui la composent, grâce à une requête SQL.

3.3.2 Réalisation

FIGURE 1 – Schéma base de données



Carte La carte du maraudeur est modelisée comme un ensemble de cases. Chaque carte possède un identifiant unique pour pouvoir permettre de stocker plusieurs cartes. Les tables "Square" et "Map" servent à cela.

Déplacement Les déplacements sont modelisés par la table "Position", où on associe à chaque ensemble de coordonnées un temps sous la forme d'un entier ($N°$ du pas de la simulation)

Simulation Chaque simulation possede un identifiant unique pour pouvoir les différencier. La table "Simulation" sert à cela.

Authentification La table "User" sert à l'authentification des utilisateurs. Le champ "hash_mdp" contient le hash du mot de passe de l'utilisateur.

3.3.3 Analyse critique

La base de données MySQL est adaptée pour faire les requêtes précises dont notre simulateur a besoin, avec toute la personnalisation que nous offrons.
Cependant, ce format nous limiterait dans les cas de simulation de taille très importante, où à chaque pas, on exigerait des insertions massive de lignes. Dans ce cas-là, une base de données NoSQL serait sans doute plus adaptée, en raison de ses performances pour gérer les données massives.
D'autre part, nosu aurions également pu faire le choix d'utiliser une base de données PostgreSQL qui respecte plus les normes SQL que MySQL, mais qui est également nettement plus longue et difficile à configurer. Dans le cadre de ce projet court, organisé en sprint, nous n'étions pas prêts à fournir cet effort de configuration supplémentaire.

3.4 Internet et Réseaux

3.4.1 Problématique

Nous avons fait le choix d'une architecture avec un serveur en ligne, distant du client. Avec ce choix, la question du mode de communication, ainsi que du protocole d'application, est apparue importante dès le début du développement. Avec notre problématique de sécurisation de l'ensemble de l'application, le choix d'une communication permettant à la fois fluidité et sécurité de l'application est devenue cruciale. Comment coupler une communication performante à un échange sécurisé, tout en restant compréhensibles ?

3.4.2 Réalisation

En lien avec notre client dédié pour le parcours CASPAR, nous avons avons choisi d'établir la communication entre notre client et notre serveur grâce au protocole HTTPS. Ce protocole, au dessus de TCP, nous a permis d'avoir une grande souplesse : en effet, dans la partie client il est implémenté comme s'il s'agissait d'un protocole HTTP classique, avec la seule différence que le certificiat SSL de notre serveur est intégré au .jar de l'application.
Une fois le certificat intégré à la liste des clés en lesquelles Java peut avoir confiance (dans le *truststore*), la communication s'établit selon le modèle *three way handshake* de manière classique, avec l'échange des clefs de sécurité en plus, comme on peut le voir dans le tableau 1.
Dès le début du développement du projet, et tout le long de notre avancement, nous avons créé notre protocole d'application. Nous avons fait le choix d'échanger toutes les données au format JSON, dont la structuration précise a permis d'éviter tout problème de compréhension d'une partie de l'application à l'autre.

Enfin, nous avons choisi de réaliser toutes nos requêtes par la méthode POST, et pas GET, pour plusieurs raisons :
Outre la complexité de nos requêtes, un paramètre important est la taille des données qui transitent : en effet, lors de la création des utilisateurs par exemple au tout début d'une simulation, le client envoie au serveur la quantité de personnes à simuler, leur statut (professeur - étudiant), ainsi que leur position de départ : cette quantité de données aurait été trop longue pour une requête GET.
De plus, cette façon de faire nous apporte une légère sécurité supplémentaire : des requêtes POST nous permettent d'éviter de faire transiter les paramètres en même temps que l'adresse d'envoi, et donc d'éviter des injections frauduleuses de requêtes, qui risque de surcharger notre serveur.

3.4.3 Analyse critique

Définir une première version du protocole d'application dès le début du projet, puis l'améliorer au fur et à mesure a eu plusieurs conséquences : d'une part, cela a permis de développer séparément les différentes fonctionnalités, et il était possible d'avoir tous les méthodes d'un seul côté, client ou serveur, prêtes à être utilisées en étant certains qu'une fois l'autre moitié implémentée, la communication fonctionnerait.
D'autre part, cela nous a forcé à précisément définir ce dont nous avions besoin pour chaque action à réaliser, et seulement cela : nous avons appris à éviter à

Source	Destination	Protocole	Type de message
Client	Serveur	TCP	SYN
Serveur	Client	TCP	SYN, ACK
Client	Serveur	TCP	ACK
Client	Serveur	TLS v1.2	Client Hello
Serveur	Client	TCP	ACK
Serveur	Client	TLS v1.2	Server Hello
Client	Serveur	TCP	ACK
Serveur	Client	TLS v1.2	Certificate
Client	Serveur	TLS v1.2	Client Key Exchange
Client	Serveur	TLS v1.2	Change Cypher Spec
Client	Serveur	TLS v1.2	Encrypted Handshake Message
Serveur	Client	TCP	ACK

TABLE 1 – Le modèle de communication entre client et serveur.

transférer des informations inutiles, qui auraient alourdi le protocole et la communication en même temps.
L'utilisation du format JSON, qui nous semblait plus simple aussi bien à l'encodage qu'au décodage, a cependant eu l'inconvénient de se montrer plus contraignante qu'avantageuse pour un certain nombre de requêtes simples. Nous aurions pu définir notre propre standard, de façon arbitraire, où nous aurions pu mettre seulement les formats dont nous avions besoin.

3.5 Analyse et Conception Objet

3.5.1 Prolématique

L'architecture d'un projet permet d'avoir un plan de route durant tout le long du développement. Il va nous guider dans nos choix, éventuellement être légèrement modifié, mais il va globalement donner une idée de ce à quoi ressemblera l'application à la fin.
Le choix de l'architecture dans un projet est très importante, c'est pour cela qu'il nous a fallu quelques jours pour décider de l'architecture que nous allions implémenter. Il est donc crucial pour le projet de choisir une bonne architecture, afin de ne pas recommencer tout ce travail au milieu du projet.
Quelle est la meilleure architecture pour une application client-serveur ?

3.5.2 Réalisation

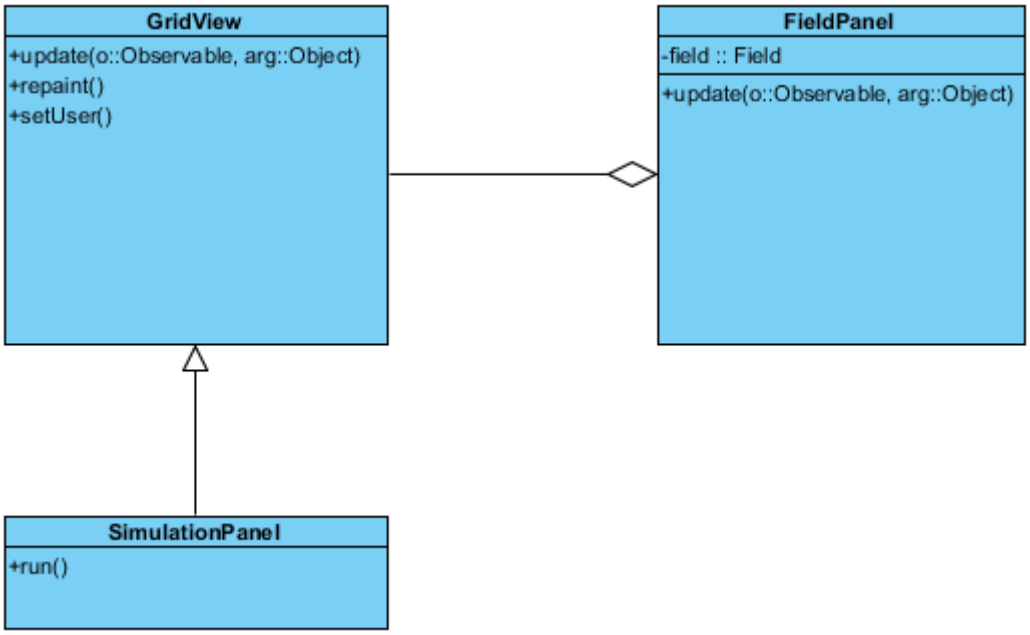
Au tout début, nous avons fait les calculs du côté client, car notre serveur ne devait gérer que la base de données, la technologie utilisée n'étant pas faite pour les calculs complexes. Nous nous sommes rendus compte, à l'aide de nos clients, que c'était un mauvais choix, si nous voulions pouvoir monter en charge. Ce type de structure est également une erreur d'architecture. Nous avons donc transféré tous les calculs côté serveur, et nous avons également changé la technologie du serveur.

Nous avons fait plusieurs choix concernant l'architecture de l'application : tout d'abord, nous avons décidé de mettre en place une architecture Client-Serveur.

Notre serveur fonctionne selon le schéma MVC. En effet, tous les contrôleurs sont reliés aux modèles. Les vues quant à elles sont spéciales du fait que notre application aie une interface Java. Elles ne sont en effet que des messagers qui renvoient les données créées par le serveur.

Enfin notre client a integré le schéma de conception "Observeur-Observable", pour réaliser les vues. Ici le schéma est simplifié car s'il était complet, on inclurait également le schéma MVC, ce qui aurait rendu ce diagramme incompréhensible :

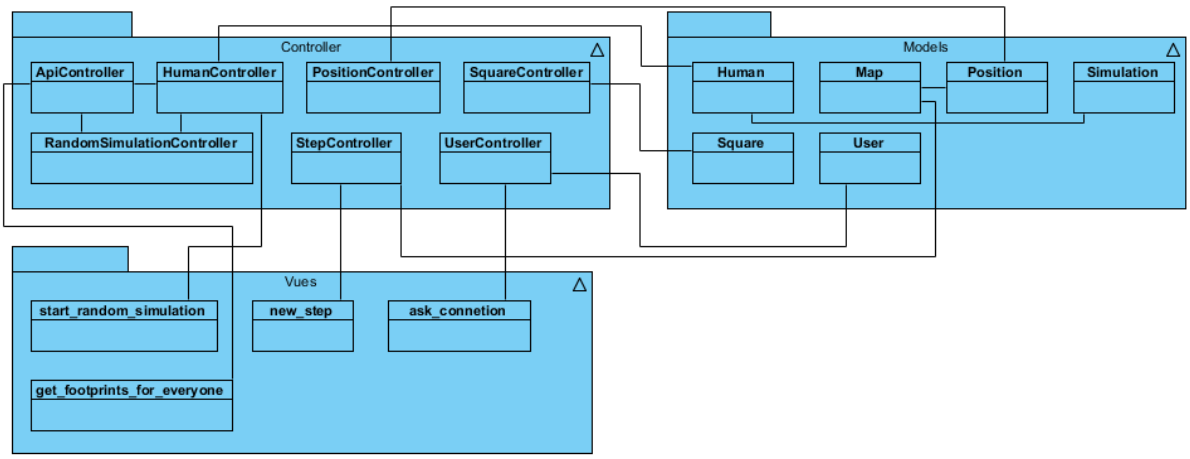
FIGURE 2 – Diagramme de classe du schéma Observeur - Observable



3.5.3 Analyse critique

Le MVC (Modèle Vue Contrôleur) est une architecture qui permet d'être maintenue à jour simplement. Chaque vue est reliée à un ou plusieurs contrôleurs, la vue les appelle en fonction des actions faites. Les vues écoutent également les modèles, en cas de changement. Le contrôleur lui, va mettre en lien ce que la vue demande et ce que le modèle répond. Voici un exemple du modèle MVC de notre Serveur, avec la particularité qu'ici, les vues sont de simples routages de la surface du serveur (MVC web) :

FIGURE 3 – Diagramme de classe du schéma MVC



Malheureusement, notre client ne suit pour le moment pas l'architecture MVC. Cependant, la restructuration qui serait nécessaire pour ce changement ne demanderait l'ajout que de peu de code.

En revanche, notre serveur a été fait en suivant le modèle MVC. Ce qui a l'avantage d'être entretenable facilement et permet un compréhension claire d'un point de vue extérieur.

Notre serveur et notre client sont totalement détachables. On peut tout à fait mettre un autre serveur en communication avec notre client. La seule condition est de suivre le protocole d'application qui a été défini.

3.6 Sprint 3 :

Dans l'objectif d'un troisième sprint, voici les stories que nous avons écrites et qui ont été validées par nos clients.

3.6.1 Stories CASPAR

En tant qu'administrateur, je veux que le temps de session soit contrôlé afin de pouvoir gérer les connexions des utilisateurs.
Critères d'acceptabilité :

1. Lors de la connexion le client ouvre une session avec le serveur.
2. Au bout de 15 minutes d'inactivité, le serveur ferme la session avec le client.

En tant que mandataire, je veux que la simulation prenne en compte les autorisations d'accès à certaines zones, afin de respecter le fait que certaines soient interdites à certaines personnes.
Critères d'acceptabilité :

1. Certaines zones sont accessibles uniquement à certains utilisateurs.
2. La simulation prend en compte l'accessibilité de ces zones pour calculer le chemin.

3.6.2 Stories AL

En tant que mandataire, je veux pouvoir rejouer les déplacements d'une simulation sur une période de temps donnée, afin d'avoir une visualisation précise lors du rejeu des traces.
Critères d'acceptabilité :

1. Le système a enregistré tous les déplacements et est capable de less trier par intervalles de temps.
2. Le système est capable d'envoyer des traces de déplacement sur des périodes de temps précises.

En tant que mandataire, je veux que lors de l'absence de données réelles une simulation se mette en place afin de modéliser le comportement des utilisateurs selon un modèle de circulation cohérent pour une destination précise.
Critères d'acceptabilité :

1. Les déplcements ne s'arrêtent pas sur la carte.
2. La simulation génère des déplacements selon un modèle de circulation crédible.
3. Les utilisateurs ne se téléportent pas et vont de salle en salle.
4. La simulation prend en compte les contraintes liées au bâti.

En tant que client, je veux voir là ou les utilisateurs se déplacent le plus souvent afin de pouvoir cartographier les différents points chaud du campus.
Critères d'acceptabilité :

1. Mettre en place un système comptant le nombre de personnes ayant visité chaque salle pour un temps donné.
2. Pouvoir exporter le résultat dans un fichier récapitulatif.

En tant que mandataire, je veux pouvoir rejouer les traces d'un groupe d'utilisateur afin de mieux voir les déplacements pour un groupe de personnes.
Critères d'acceptabilité :

1. Lors que l'on rejoue des traces, nous pouvons choisir un groupes de personnes à rejouer en particulier.

En tant que mandataire, je veux pouvoir rejouer les traces d'un seul utilisateur afin de pouvoir suivre ses déplacements.
Critères d'acceptabilité :

1. Lors que l'on rejoue des traces, nous pouvons choisir une personne à rejouer en particulier.