ALLIROL Lucas 4ETI

BLANCHARD Loïc

VERCAEMER Julien

WILLER Thibault

Compte rendu Projet majeur informatique 2017

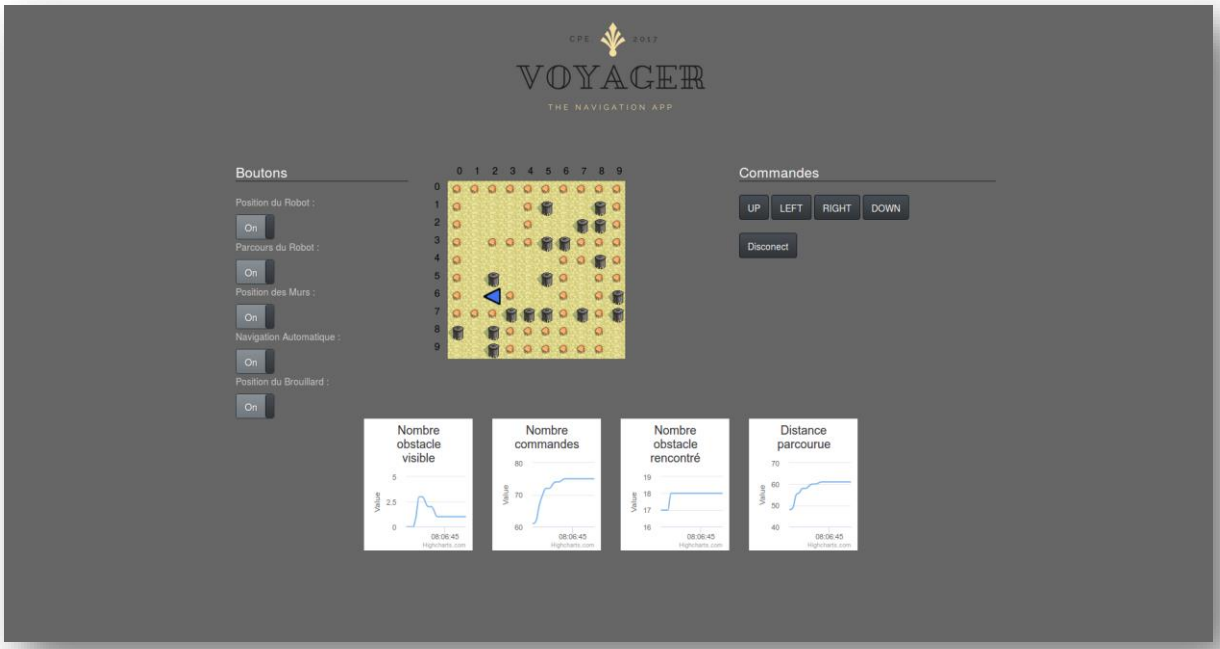


Table des matières

Introduction

Nous devions pour ce projet réaliser une page web permettant de contrôler un personnage sur une carte.

Pour cela il nous fallait réaliser plusieurs fonctionnalités :

- La création d'un simulateur de robot, qui reproduit les mouvements du robot sur une carte
- Un affichage de plusieurs informations envoyées par le robot, par exemple la distance parcourue ou le nombre d'obstacles rencontrés
- L'affichage des déplacements du robot, avec la connexion à une base de données ainsi que la création d'une page utilisateur et une autre admin
- Une navigation automatique du robot qui parcourt la carte en explorant toutes les zones d'ombres de façon optimale.

I) Le Front End

Le front end représente l'interface visuelle ainsi que les différentes actions des utilisateurs de l'application. L'application est constituée de trois pages distinctes.

La première correspond à la page d'accueil de l'application : Elle est faite de plusieurs formulaires, un pour se connecter, un autre pour s'inscrire et enfin un dernier pour choisir les paramètres de simulation. Ces informations seront envoyées aux serveurs (et donc au côté back-end) via une méthode post request. Seul un utilisateur peut utiliser l'application, si un autre utilisateur souhaite se connecter alors que l'application est déjà utilisée, il sera bloqué et devra attendre que l'autre se déconnecte de l'application. Cette page web utilise simplement les technologies xhtml et css.

La deuxième page correspond à l'interface utilisateur de l'application, on y accède après avoir entré le bon identifiant et le bon mot de passe associé dans le formulaire connexion dans la page d'accueil. Sur cette page on retrouve divers éléments :

- L'élément central : le plateau de simulation. Ce plateau est constitué de 5 canvas superposés. Chaque canvas est chargé d'afficher une information : la grille de la carte, le robot, la trajectoire du robot, les obstacles et les cases inconnues. Un fichier JavaScript contient diverses fonctions d'affichage. Ces fonctions lisent un fichier json récupéré dans une variable depuis le serveur et affiche les informations en fonction de ce que contient ce fichier. A chaque action utilisateur, l'ensemble des canvas est régénéré en fonction du nouveau fichier json et actualise donc la simulation.
- L'élément de gauche : les boutons. Ces boutons on/off affichent ou non différentes informations sur le plateau de simulation. Techniquement, cela revient simplement à afficher ou non les canvas. Un dernier bouton « Navigation automatique » est censé activer ou non une fonction de découverte de la carte de manière automatique.
- L'élément de droite : les boutons de commandes. Ces boutons correspondent aux déplacements du robot. Dès qu'un des boutons est activé, une fonction JavaScript envoie une information au serveur via un élément txt. Un autre bouton de déconnexion permet à l'utilisateur de retourner à la page d'accueil, et d'ainsi libérer l'utilisation de l'application.
- L'élément du bas : les graphiques. Ces graphiques affichent des informations relatives à la simulation. Nous avons récupéré des scripts tout fait, puis adapté ces derniers à l'application. Les données affichées sont récupérées dans le fichier json.

La troisième page correspond à l'interface administrative de l'application. Elle est en tout point identique à l'interface utilisateur à la différence prêt de l'ajout d'un bouton pour bloquer la possibilité aux utilisateurs d'utiliser l'application.

II) Le Back End

La partie Back End est la partie non visible du projet, ce sont les traitements, les outils et les algorithmes mis en place pour aboutir à un projet fonctionnel.

On a tout d'abord la partie simulation du robot, avec la définition de la carte et sa génération aléatoire, les déplacements possibles du robot et le tracé du chemin parcouru.

On a ensuite la partie connexion avec le serveur, base de données et droits de l'utilisateur et de l'administrateur.

Partie Simulateur :

Dans la partie simulatrion nous devions gérer le modèle du programme, c'est là qu'est codé, en Java, le comportement du robot et son environnement. Le programme est structuré en un grand nombre de packages et de classes, pour avoir une vue globale se référer au diagramme de classe.

Lorsqu'un déplacement veut être effectué, le web browser envoie une requête qui sera traitée par une servlet grâce à une communication de type jax-RS. Une map aura déjà été créée dans le modèle, dès l'instant où le client s'est connecté, en ayant rentré la taille de la map (forcément carrée) et la probabilité d'apparition des obstacles.

La map et l'ensemble des informations relatives au modèle sont toujours stockés dans la mémoire application du serveur et récupérés à chaque nouvelle requête du client. Lorsque le client veut se déplacer à gauche par exemple, on récupère la map, on effectue le déplacement (s'il est possible) et on renvoie au client la map mise à jour ainsi qu'un lot d'informations et de mesures mis à jour dans un fichier json.

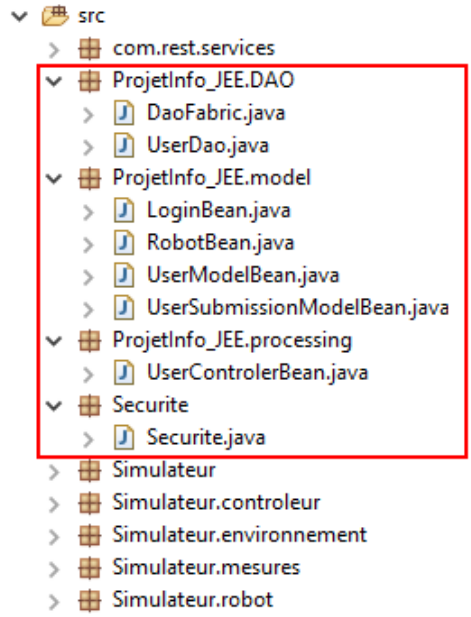
Nous allons maintenant décrire le fonctionnement global du simulateur. Nous pourrions en parler sur 20 pages mais nous allons tâcher d'être le plus concis possible.

Notre classe RobotCtrl sert de chef d'orchestre, il contient les classes robot, environnement et mesures. L'environnement correspond à la map réelle avec la position de tous les obstacles. La classe mesures contient tous les attributs relatifs aux mesures telles que les obstacles rencontrés et le nombre de commandes effectuées. Enfin la classe robot contient la map découverte par le robot (et donc celle affichée dans le web Browser) et le pattern de vision.

Voici en gros ce qu'il se passe lorsqu'un déplacement est effectué : on regarde d'abord s'il est possible (dans les limites de la map et pas d'obstacle sur la case d'arrivée). Ensuite on déplace le robot à ces nouvelles coordonnées. Enfin on applique le pattern : on récupère le pattern de vision du robot (en fonction de son orientation) puis on copie la map réelle sur la map du robot à l'endroit indiqué par le pattern : la map vue par le robot est ainsi mise à jour ! A chacune de ces étapes, nous prélevons des données pour mettre à jours nos mesures. Nous avons également développé une fonction de découverte automatique de la map mais nous n'avons pas su l'implémenter coter web browser de façon optimale nous avons donc choisi de laisser tomber cette fonctionnalité.

Partie JEE :

Les classes Java qui ont servi pour cette partie :



- Dans le package ProjetInfo_JEE.DAO, on trouve la fabrique d'utilisateurs. On utilisera les méthodes de DaoFabric lors de la connexion à la base de données.

- Dans le package ProjetInfo_JEE.model, les différents beans avec différents scoped selon les cas d'utilisation.

- Dans le package ProjetInfo_JEE.processing, la liaison avec la base de données. On appelle les méthodes de cette classe lors de la connexion et de l'inscription de la page d'accueil userLogin.xhtml.

- Dans le package Sécurité, les informations sur qui est connecté au site et l'état de la télécommande.

Nous avons réalisé la connexion base de données et la page d'accueil userLogin.xhtml en JSF. On a tout d'abord créé deux bases de données différentes : une base de données admin et une autre base de données user, elles contiennent deux informations : un login et un password. N'importe qui pourra s'inscrire en tant qu'utilisateur via un formulaire et une requête SQL, il sera alors stocké dans la base de données user. La base de données admin n'est pas modifiable, et les administrateurs ont été écrit en dur lors de sa création. Il y a un seul formulaire de connexion, on se connecte à la page utilisateur ou administrateur en rentrant les bonnes informations.

Un seul utilisateur peut être connecté à la fois sur la page de commande du robot utilisateur. En effet lors du login, si la personne qui se connecte est un utilisateur, ses informations sont stockées dans un objet statique. On check à chaque connexion le contenu de l'objet, et s'il est déjà rempli, le nouvel utilisateur ne peut pas se connecter. Lors de la déconnexion d'un utilisateur, l'objet est réinitialisé ce qui permet à un nouvel utilisateur de se connecter.

Pour la télécommande de l'administrateur on a défini un booléen global en statique. En utilisant l'API REST, on a implémenté un bouton télécommande. Si l'administrateur appuie dessus, le booléen change à true s il valait false et à false s'il valait true. Quand un utilisateur veut bouger le robot le programme check tout d'abord si le booléen a la valeur false (éteinte).

Interaction robot et page utilisateur :

Afin de bouger le robot on utilise quatre boutons directionnels qui envoient des informations au serveur pour mettre à jour la carte. Pour mettre en place cela on a utilisé l'api REST.

Quand on appuie sur un des boutons on déclenche une fonction JavaScript :

```
function click_up() {
    $.post("rest/cmd/UP",{},
           function(data,status){
             myJson=data;
             MajCanvas();
             stop_deplacement==0;
    });

    };
```

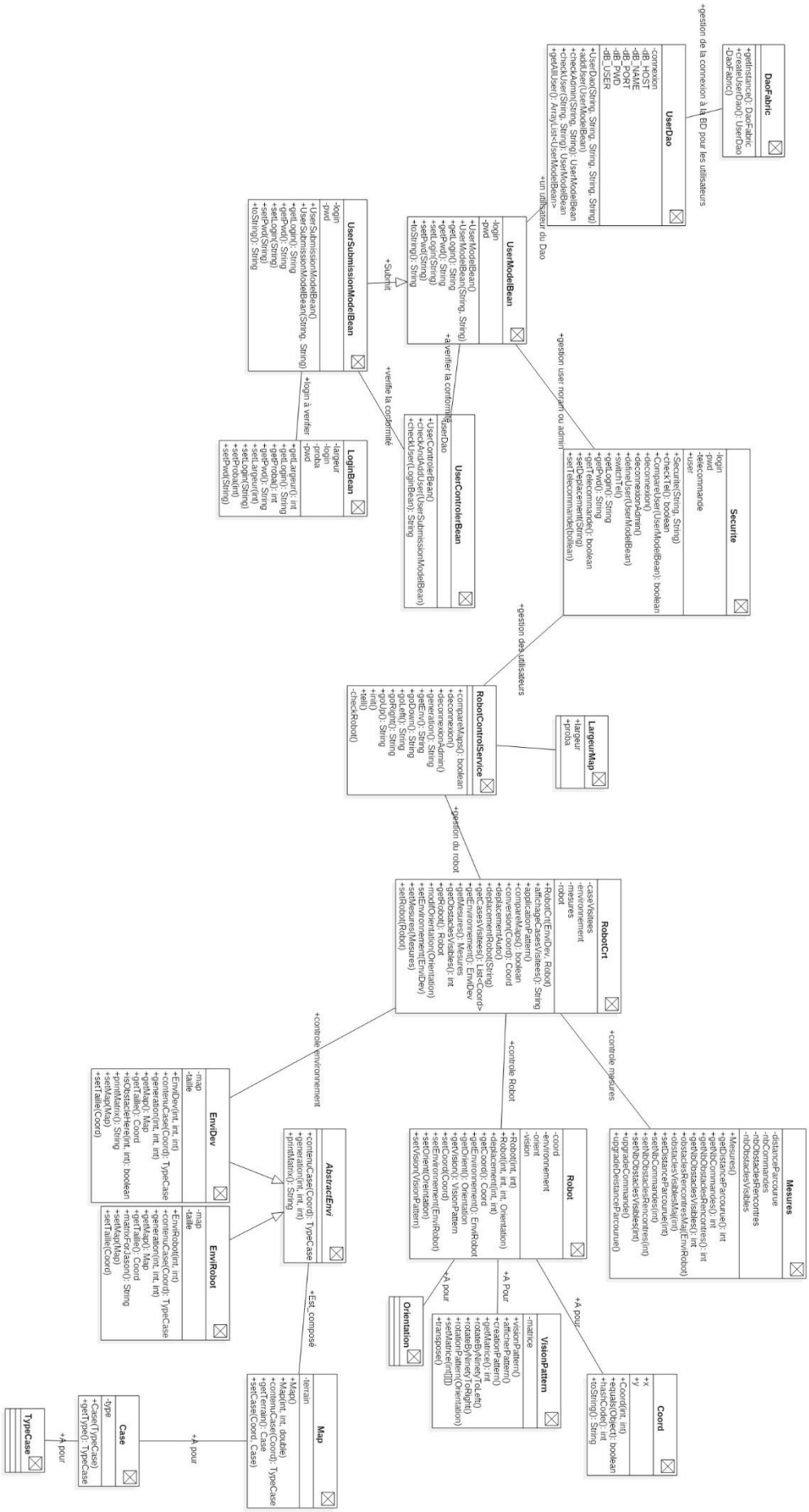
Qui va appeler une méthode de la classe RobotControlerService :

```
@POST
@Produces(MediaType.APPLICATION_JSON)
@Path("UP")
public String goUp()
{
    if(Securite.CheckTel() == true)
    {
        ( ((RobotBean) context.getAttribute("ROBOT")).getRobotCrt()).deplacementRobot("UP");
        return this.getEnv();
    }
    System.out.println("Un administrateur a interdit l'accÃ¨s au robot");
    return "Un administrateur a interdit l'accÃ¨s au robot";
}
```

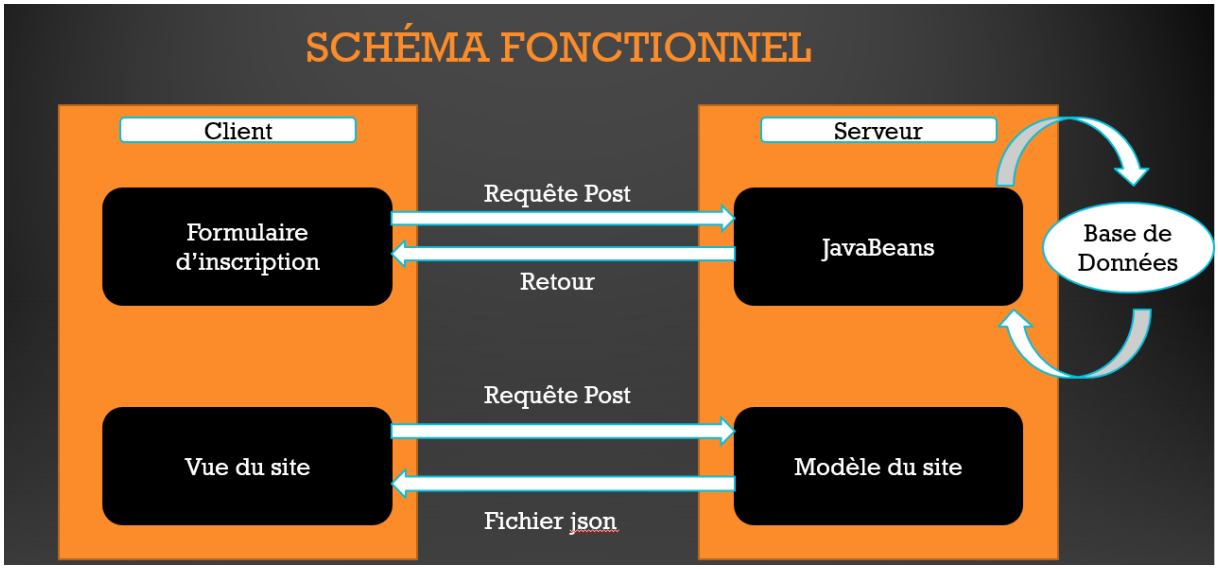
Elle vérifie si le déplacement est correct et renvoie la carte mise à jour (si la télécommande n'est pas enclenchée)

III) Diagramme de Classe

Le diagramme des classes du Back-end est page suivante.


DaoFabric
+getInstance(): DaoFabric
+creatUserDao(): UserDao
DaoFabric()
+gestion de la connexion à la BD pour les utilisateurs
UserDao
-connexion
-dB_HOST
-dB_NAME
-dB_PORT
-dB_PWD
-dB_USER
+UserDao(String, String, String, String, String)
+addUser(UserModelBean)
+checkAdmin(String, String): UserModelBean
+checkUser(String, String): UserModelBean
+getAllUser(): Array(List<UserModelBean>
+un utilisateur du Dao
UserModelBean
-login
-pwd
+UserModelBean()
+UserModelBean(String, String)
+getLogin(): String
+getPwd(): String
+setLogin(String)
+setPwd(String)
+toString(): String
+gestion user normal ou admin
Securite
-login
-pwd
-telecommande
+user
+Securite(String, String)
+CheckTel(): boolean
+CompareUser(UserModelBean): boolean
+deconnexion()
+deconnexionAdmin()
+definirUser(UserModelBean)
+switchTel()
+getLogin(): String
+getPwd(): String
+getTelecommande(): boolean
+setDeplacement(String)
+setTelecommande(boolean)
+Submit
UserSubmissionModelBean
-login
-pwd
+UserSubmissionModelBean()
+UserSubmissionModelBean(String, String)
+getLogin(): String
+getPwd(): String
+setLogin(String)
+setPwd(String)
+toString(): String
+verifie la conformité
+login à verifier
LoginBean
-largeur
-login
-proba
-pwd
+getLargeur(): int
+getLogin(): String
+getProba(): int
+getPwd(): String
+setLargeur(int)
+setLogin(String)
+setProba(int)
+setPwd(String)
+verifie la conformité
UserControlerBean
+UserControlerBean()
+checkAndAddUser(UserSubmissionModelBean)
+checkUser(LoginBean): String
UserDao
+gestion des utilisateurs
RobotControlService
+compareMaps(): boolean
+deconnexion()
+deconnexionAdmin()
+generation(): String
+goDown(): String
+goLeft(): String
+goRight(): String
+goUp(): String
+init()
+initI()
-checkRobot()
LargeurMap
+largeur
+proba
+gestion du robot
RobotCtrl
-caseVisitees
-environnement
-mesures
-robot
+RobotCtr(EnvDev, Robot)
+afficharpCaseVisitees(): String
+applicationPattern()
+CompareMaps(): boolean
+conversion(Coord): Coord
+deplacementAuto()
+deplacementRobot(String)
+getCasesVisitees(): List<Coord>
+getEnvironnement(): EnvDev
+getMesures(): Mesures
+getObstaclesVisibles(): int
+modifOrientation(Orientation)
+setEnvironnement(EnvDev)
+setMesures(Mesures)
+setRobot(Robot)
+controle environnement
+controle Robot
+controle mesures
Mesures
-distanceParcourue
-nbCommandes
-nbObstaclesRencontres
-nbObstaclesVisibles
+Mesures()
+getDistanceParcourue(): int
+getNbCommandes(): int
+getNbObstaclesRencontres(): int
+getNbObstaclesVisibles(): int
+obstacleVisibleMoi(int, int, EnvRobot)
+setDistanceParcourue(int)
+setNbCommandes(int)
+setNbObstaclesRencontres(int)
+setNbObstaclesVisibles(int)
+upgradeCommande()
+upgradeDistanceParcourue()
AbstractEnvi
+contenuCase(Coord): TypeCase
+generation(int, int, int)
+printMatrix(): String
EnvDev
-map
-taille
+EnvDev(int, int, int)
+contenuCase(Coord): TypeCase
+generation(int, int, int)
+getMap(): Map
+getTaille(): Coord
+isObstacle(Coord): boolean
+printMatrix(): String
+setMap(Map)
+setTaille(Coord)
EnviRobot
-map
-taille
+EnviRobot(int, int)
+contenuCase(Coord): TypeCase
+generation(int, int, int)
+getMap(): Map
+getTaille(): Coord
+isObstacle(Coord): boolean
+printMatrix(): String
+setMap(Map)
+setTaille(Coord)
Robot
-coord
-environnement
-orient
-vision
+Robot(int)
+Robot(int, int, int, Orientation)
+deplacement(int, int)
+getCoord(): Coord
+getEnvironnement(): EnvRobot
+getOrient(): Orientation
+getVision(): VisionPattern
+setEnvironnement(EnvRobot)
+setOrient(Orientation)
+setVision(VisionPattern)
+A pour
+A pour
+A pour
Orientation
VisionPattern
-matrice
+visionPattern()
+afficherPattern()
+creationPattern()
+getMatrice(): int
+rotationAntiHoraire()
+rotationHoraire()
+rotationPattern(Orientation)
+setMatrice(int[][])
+transposee()
Coord
+x
+y
+Coord(int, int)
+equals(Object): boolean
+toString(): String
+Est_composé
Map
-terrain
+Map()
+Map(int, int, double)
+getCase(Coord): TypeCase
+getTerrain(): Case
+setCase(Coord, Case)
+A pour
Case
-type
+Case(TypeCase)
+getType(): TypeCase
+A pour
TypeCase

IV) Schéma fonctionnel



V) Répartition du travail dans le groupe

Lots	Objectif	Difficultés			Responsable		
		1	2	3			
[FCT-INFO0-Simulator]							
	[FCT-INFO0-TelCom-Env]	x			Lucas	Loic	
	[FCT-INFO0-TelCom-Robot]		x		Lucas	Loic	
	[FCT-INFO0-TelCom-Measures]	x			Lucas	Loic	
	[FCT-INFO0-TelCom-RobotCrt]	x			Lucas	Loic	Julien
[FCT-INFO1-TelCom]							
	[FCT-INFO1-TelCom-Conn]	x			Thibault		
	[FCT-INFO1-TelCom-WebCmd]	x			Julien		
	[FCT-INFO1-TelCom-WebAdm]	x			Thibault		
[FCT-INFO2-Measure]							
	[FCT-INFO2-Measure-Aff]		x		Julien	Loic	
	[FCT-INFO2-Measure-AffAuto]		x				
	[FCT-INFO2-Measure-AffGraph]			x	Julien		
[FCT-INFO3-Carte]							
	[FCT-INFO3-Carte-Traj]	x			Lucas		
	[FCT-INFO3-Carte-AffTraj]		x		Julien		
	[FCT-INFO3-Carte-Obs]			x	Julien		
[FCT-INFO4-AutoNav]							
	[FCT-INFO4-AutoNav-customdisplay]		x		Julien		
	[FCT-INFO4-AutoNav-autoLauncher]		x				
	[FCT-INFO4-AutoNav-slam]			x			

VI) Lien vers vidéo YouTube Teaser

La chaîne YouTube du projet est la suivante :

https://www.youtube.com/channel/UCdZ-vn6Agr-Lac_9_eOfL4g

Conclusion

Finalement nous avons réalisés trois des quatre fonctionnalités : Simulation du robot, Connexion au serveur et affichage des données, et tracé du robot sur la carte. Si nous avions eu plus de temps on aurait pu terminer notre déplacement automatique, améliorer l'affichage de la carte et améliorer la connexion utilisateur (une seule connexion à la fois est trop restrictif). On pourrait par exemple afficher une carte pour un utilisateur qui essaye de se connecter, mais si une personne utilise déjà le robot il ne pourrait pas appuyer sur les boutons de direction. Enfin il faudrait améliorer l'aspect graphique du site.

Nous avons quand même passé de bons moments à coder ce site et on a beaucoup appris, notamment comment lier plusieurs notions apprises en un seul projet (JEE, POO, TLI …). On a démarré très lentement le projet, mais après avoir défini notre ligne directrice et le rôle de chacun nous avons travaillé efficacement pour rendre le projet le plus abouti possible.