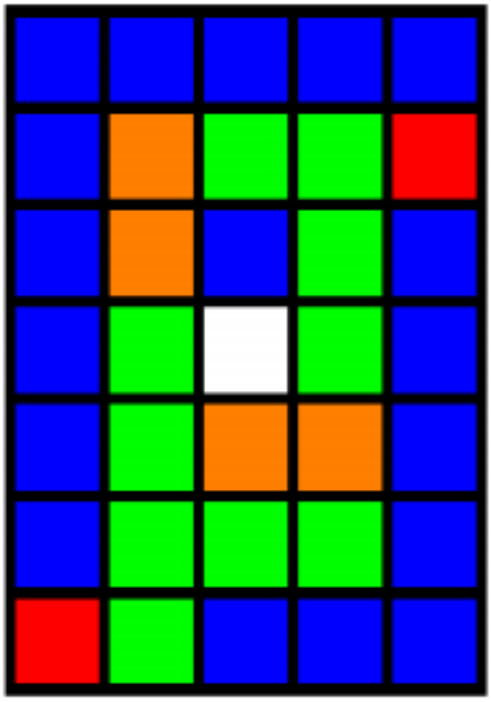
RAPPORT

Projet de Robotique

Tuteur : Sylvain Castagnos



Licence MIASHS- Sciences Cognitives

2020/2021 – 18/12/2021

Auteurs :

CARBONNIER Nicolas
HAJEK Valentin
SCHEFFMANN Tom

Institut des sciences du Digital
Management & Cognition
COMPOSANTE DE L'UNIVERSITÉ DE LORRAINE

Introduction

Afin d'avoir une première approche de la robotique nous avons pour projet de programmer des robots qui jouent au Twister dont le but principal est de se placer sur des points de couleurs. Pour réaliser ce projet nous avons utilisé deux robots Lego Mindstorms utilisant le logiciel EV3 qui est le kit robotique de troisième génération de la gamme Mindstorm. Le robot est équipé d'une machine virtuelle java permettant de programmer ces derniers.

Notre plateau de jeu Twister est composé de 6 couleurs (blanc, rouge, vert, bleu, orange, noir), la couleur rouge concerne le point de départ de la partie, la couleur noire concerne les bords de chaque carré, quant aux autres couleurs elles font partie intégrante du jeu, le blanc est la seule couleur qui posséde un carré unique au sein du plateau.

Le robot quant à lui posséde des équipements qui nous seront utile à relever le défi, un capteur de couleur, d'un capteur ultra-son, de deux moteurs respectivement roue gauche et roue droite.

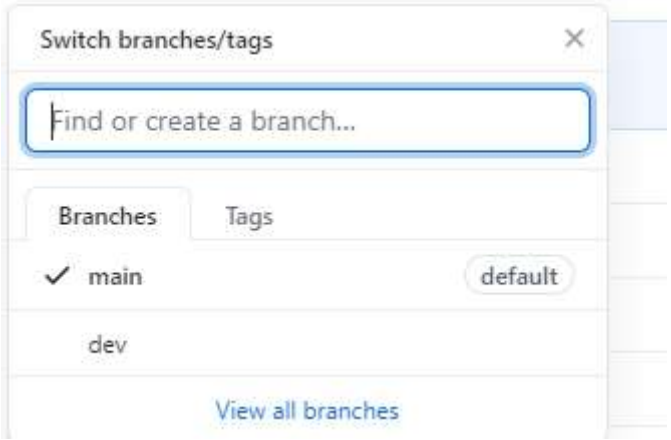
Le but final étant que deux robots Lego puisse jouer à Twister dans l'environnement donné en passant par plusieurs étapes, l'exploration de l'environnement de jeu afin de connaître l'emplacement des différentes couleurs, la représentation de la cartographie, l'envoie de celle-ci à son collègue de jeu puis le jeu en lui-même contenant l'envoie à tour de rôle de la case où devra se rendre le robot, la navigation de celui-ci jusqu'à la case en respectant diverses contraintes.

I. Conception

a. Gestion du travail

Organisation technique

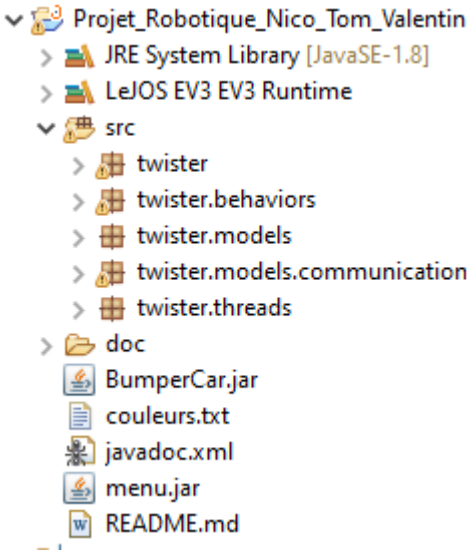
Pour gérer correctement et faciliter le code sur notre projet nous avons utilisé git. Chaque membre du groupe avait accès au git, et chacun effectuait les comit pull, push en renseignant les autres membres s'il y avait des nouveautés. Pour mieux nous organiser et être sûr d'avoir un rendu fonctionnel nous avons décidé de travailler sur deux branches, une branche master et une branche dev.



La branche master concernait ce qu'on avait validé, ce qui était fonctionnel et notre branche dev contenait ce qui était en cours de développement. Cette manière de faire a été un bénéfice sur le sprint final où ce que nous avions changé n'était plus fonctionnel pour la vidéo, nous avons donc pu switcher de branche pour avoir un rendu répondant à la majeure partie des objectifs. De plus nous n'avons rien perdu de ce que nous étions en train de corriger car c'est resté sur notre branche dev.

Néanmoins nous avons rencontré des complications avec git qui nous a fait perdre un temps majeur lors du développement du projet. Nicolas et Valentin qui ont l'habitude de l'utilisation de git se sont vu plusieurs fois confronté à des conflits lorsque l'on voulait pull le projet. A l'opposé des outils de jetbrains, eclipse se veut capricieux avec git, la gestion des merge n'est pas très claire. Nous avions donc souvent comme solution finale, une manière brouillonne de faire, on enregistrait une copie de notre projet à la main et nous clonons de nouveau le projet pour repasser à la main ce que nous avions fait, ce qui n'était pas pratique.

Nous avons choisi d'organiser notre projet en différents package en mettant les behaviors ensemble, models, et threads avec des sous packages si nécessaire ce facilitait l'organisation du code.



Organisation du travail

Pour nous organiser dans notre travail dès le départ nous avons déterminé les grandes étapes du projet, afin de le respecter au mieux. Pour ça nous avons créé un drive avec les différents objectifs attendus, ainsi qu'un tableau qui permettait à chaque ajout majeur de détailler globalement ce qui a été fait. Cette manière de s'organiser à été respectée par l'ensemble du groupe ce qui a facilité l'écriture du rapport, nous avions ainsi une trace écrite et un justificatif à chaque implémentation effectuée.

Nous avons également utilisé l'outil trello dès le départ du projet, pour clarifier des deadlines pour les objectifs primaires. Néanmoins nous nous sommes vis vite dépassé, l'apparition du second confinement qui a impacté les premières séances de travail qui suivait, l'incapacité à certains membres du groupe à pouvoir être en présentiel pendant le tp ont fait que le trello a été mis de côté à la moitié du projet lorsque certaines deadlines se sont vu dépassé.

b. Contraintes :

Ce projet comporte des contraintes organisationnelles :

- Le projet s'effectue en groupe de trois, alors nous allons devoir répartir les tâches au sein de notre groupe, en prenant en compte les compétences de chacun
- La communication est également une contrainte dans un groupe avec des niveaux différents. Il faut pouvoir communiquer ses résultats et ses idées au reste du groupe.

Ce projet possède aussi des contraintes techniques :

- Nous avons dû utiliser le contenu du cours de Mr Castagnos. Nous avons découvert l'environnement LeJos, qui permet d'envoyer des informations à un robot EV3. Nous nous sommes familiarisés avec la librairie EV3JLib et avons pris en compte la documentation mise à notre disposition.
- Nous devions utiliser des outils de gestion de projet tel que GitHub. Heureusement, nous connaissions déjà cette ressource, cela a donc été simple et rapide à mettre en place.

Ce projet a également des contraintes de temps :

- Encore étudiants, nous ne travaillions pas à temps plein sur notre projet. Nous devions donc gérer nos cours et nos autres projets simultanément et pour certains le travail à côté. Néanmoins les séances supplémentaires ajoutées par Mr. Castagnos étaient d'une grande aide.

Ce projet a également des contraintes de contexte :

- Le contexte sanitaire nous a, quelquefois, empêché de nous rendre à l'IDMC pour pouvoir y essayer nos programmes sur les robots. De plus, nous ne pouvions pas nous voir pour travailler ensemble. Nous aurions eu davantage de temps pour finir si cette contrainte n'avait pas été présente.

Fonctionnalités et choix de la mise en œuvre :

Dans un premier temps il est nécessaire de comprendre concrètement les étapes de développement du projet, pour ça nous avons décidé de lister les étapes primaires pour répondre à l'objectif du projet puis les étapes secondaires qui nous permettraient de rendre le projet plus complet. Ensuite nous avons classifié un ordre sur ces étapes primaires. De plus nous avons renseigné au préalable du développement des idées de conception

I Etape : Fonctionnalités primaires répondant aux règles du Twister
II Etape : Fonctionnalités secondaires plus poussé, à réaliser une fois les étapes primaires terminé

Fonctionnalité	**Description générale**	**Etape (ordre)**
Couleur du jeu	Définition des couleurs que le robot doit connaître afin de pouvoir les détecter par la suite. Définir des seuils de perception de couleurs avec des tests sur les cinq couleurs.	I Etape (1)
Rester dans l'espace de jeu	Nous avons défini les dimensions du terrain de jeu. La cartographie se fait donc machinalement, puis pour se déplacer, on utilise les coordonnées des cases.	I Etape (2)
Arrêter le programme	Nous avons un bouton pour arrêter le programme, et qui éteint aussi les vapeurs. Aussi, lorsque l'on passe en dessous d'un certain de seuil de pourcentage de batterie, le programme s'arrête.	I Etape (3)
Cartographie de l'environnement	Déposer sur la case rouge puis se déplacer sur la carte en détectant le nombre de couleurs. Enregistré la position des couleurs depuis le point de départ (hauteur, largeur)	I Etape (4)
Navigation	Le robot doit pouvoir se déplacer dans l'environnement en restant dans l'espace de jeu. Il doit positionner son capteur au centre d'une case.	I Etape (5)

	Gérer l'accélération (vitesse linéaire et rotation)	
Cartographie collaborative	Envoie de la cartographie et informations utile au jeu au second robot (protocol serveur) en bluetooth ou en wifi	I Etape (6)
Définir les règles de jeu	-Robot 1 envoie une couleur au Robot 2 en choisissant aléatoirement une couleur - Robot 2 reçoit l'information et se déplace vers la couleur et met son capteur au centre Chaque robot écrit sur l'écran la couleur choisie	I Etape (7)
Evite-moi si tu peux	Faire en sorte que les deux robots ne se percutent pas, ils doivent donc se contourner Dijkstra, A * (envoie de position (abscisse et ordonnée) à chaque tour pour calculer le chemin) ou capteur à ultrason	II Étape (1 & 3)
Minimiser le parcours	Minimiser le nombre de cases à parcourir. Exemple : parmi toutes les cases d'une couleur x il doit choisir la plus proche et le parcours le moins long.	II (Etape 2)

Mise en oeuvre

Fonctionnement du programme

Notre programme se décompose en différents types de classes. Outre la classe principale BumperCar, il y a les classes de comportement Move, Turn, ColorDetector et Quit étendues de la classe abstraite ThreadBehavior. Cette dernière implémente l'interface Behavior et permet de définir le Thread ayant déclenché le comportement pour que celui-ci puisse être notifié à la fin de son exécution. Il y a également les classes Menu, ColorCalibration, Cartography, Game et Navigation étendues de Thread. Il était nécessaire de décomposer les fonctionnalités du robot en thread car le lancement de l'Arbitrator, gérant les comportements, est bloquant. Il était donc impossible de demander des actions à l'utilisateur et de lancer les différentes fonctionnalités dans le thread principal une fois l'Arbitrator lancé.

Dans le programme le robot est représenté par la classe Robot qui est définie par ses coordonnées x et y, sa direction, un plateau de jeu board et différents booléens pour connaître le

comportement qui doit se déclencher (moveForward, moveBackward, turnLeft, turnRight, takeColor, calibrateColor). Ce sont ces booléens que la méthode takeControl des comportements va vérifier pour se lancer. Il possède également l'attribut nbCases, définissant le nombre de cases à travers lors du prochain déplacement, ainsi que le booléen colorCalibrated et le tableau à deux dimensions rgbs, permettant d'enregistrer les codes RGB des couleurs lors de sa calibration. Le plateau, quant à lui, est représenté par la classe Board qui est définie par un tableau board à deux dimensions contenant les codes couleurs des cases du plateau de jeu, ainsi que par un booléen cartographed permettant de savoir si le plateau a été cartographié. C'est également dans cette classe que se trouve les méthodes casesDeLaCouleur et caseLaPlusProche, permettant de trouver la case la plus proche d'un robot pour une couleur donnée.

Nous avons également créé les interfaces TwisterColor, contenant les codes couleurs, leur code RGB et leur représentation textuelle, Parameters, contenant différents paramètres comme les codes de direction, la taille de plateau, la taille des roues du robot et la vitesse de déplacement ou de rotation, ainsi que RegleJeu, contenant une fonction retournant une couleur aléatoire.

Le programme se lance donc avec la classe principale BumperCar qui se charge alors d'initialiser le Board, le MovePilot, le Robot et l'EV3ColorSensor. Elle initialise ensuite les ThreadBehavior Move, Turn, ColorDetector et Quit, puis créé l'Arbitrator et le Menu avant de les lancer. Menu prend ensuite le relais dans l'exécution du programme, propose tout d'abord à l'utilisateur de calibrer le robot et lance donc un thread ColorCalibration. Après avoir été notifié à la fin de l'exécution de ce dernier, le menu propose ensuite à l'utilisateur de choisir une cartographie parmi deux types (suivant la position de départ du robot) et la réception de la cartographie d'un autre robot. Si l'un des deux premiers choix est fait, le menu lance un thread Cartography et attends la fin de son exécution de la même manière qu'avec ColorCalibration. L'utilisateur se retrouve ensuite devant le menu principal permettant de lancer la partie, afficher le plateau ou quitter le programme. S'il lance la partie, le menu va réinitialiser les coordonnées et la direction du robot avant de lancer Game qui va se charger de récupérer une couleur aléatoire et de lancer Navigation. Ce thread va donc demander la case la plus proche pour cette couleur à Board et donner les instructions au robot pour atteindre cette case. Une fois la case atteinte, Game va demander à l'utilisateur s'il veut relancer un tour ou non.

Le lancement des comportements par l'Arbitrator fonctionne de manière similaire pour tous les Behavior. Le comportement va regarder les valeurs des booléens de Robot le concernant et se lance si l'un d'eux est vrai. À la fin de son exécution, il passe le booléen l'ayant déclenché à faux puis notifie le Thread défini par la méthode setThread de la classe abstraite ThreadBehavior.

Couleur du jeu

Objectif	Définition des couleurs que le robot doit connaître afin de pouvoir les détecter par la suite
Description	Le robot doit être capable de distinguer les différentes couleurs du plateau sans erreur. Mise en place d'une calibration pour calculer le code RGB de chaque couleur présente sur le plateau.
Contraintes et règles de gestion	Le code RGB obtenu par le EV3ColorSensor n'est pas toujours le même suivant la distance entre le capteur et le plateau, les conditions d'éclairage et le plateau en lui-même, car deux plateaux étaient à notre disposition.
Niveau de priorité	Très Haute
Objectif atteint ?	Oui

La calibration et la détection des couleurs se font toutes les deux avec le Behavior ColorDetector, pouvant être déclenché par les booléens calibrateColor et takeColor.

Le thread ColorCalibration, chargé de calibrer le robot, va demander à l'utilisateur plusieurs prises successives d'une même couleur pour en calculer un code RGB moyen. Lorsque l'utilisateur appuie sur le bouton RIGHT, cela déclenche le ColorDetector qui va simplement modifier l'attribut detectedColor du thread avec le code RGB qu'il a détecté. Le thread va ensuite faire le total des valeurs rouges, vertes et bleues avant de les diviser par le nombre de prises pour en obtenir la moyenne et enregistrer le code RGB obtenu dans le robot pour la couleur associée.

Lors de la détection d'une couleur, utilisée pour la cartographie, le ColorDetector va appeler sa méthode getColor avec le code RGB qu'il a détecté en paramètre. Cette méthode va ensuite calculer la distance euclidienne entre ce code RGB et les codes RGB enregistrés dans le robot pour obtenir la couleur la plus proche et en retourner le code couleur. Le ColorDetector enregistre ensuite le code obtenu dans le plateau aux coordonnées actuelles du robot.

$$\text{distance} = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

Rester dans l'espace de jeu

Objectif	Faire en sorte que le robot reste dans l'espace de jeu
Description	Il est impératif qu'au cours de la partie, notre robot ne sorte pas du plateau.
Contraintes et règles de gestion	Réussir à caractériser les cases du plateau par leurs coordonnées. Utiliser des valeurs fixes pour rester dans le plateau lors de la cartographie.
Niveau de priorité	Haute
Objectif atteint ?	Oui

L'espace de jeu possède une largeur et une hauteur définie.

Arrêter le programme

Objectif	Arrêter le programme
Description	Il faut que le programme puisse s'arrêter en cas de batterie faible ou lorsque l'utilisateur appuie sur un bouton donné.
Contraintes et règles de gestion	Il faut que l'on puisse avoir accès aux paramètres de la batterie.
Niveau de priorité	Haute
Objectif atteint ?	Oui

Nous avons un Thread Quit, qui va quitter le programme si la batterie passe en dessous d'un certain seuil ou si l'utilisateur appuie sur un bouton donné, dans notre cas ESCAPE.

Cartographie de l'environnement

Objectif	Capable de retranscrire la cartographie établie
Description	Il faudra être capable d'enregistrer la position des couleurs sur le plateau de jeu pour pouvoir par la suite naviguer correctement dans l'espace de jeu
Contraintes et règles de gestion	Détection des couleurs qui peut varier en fonction de la lumière, ne pas sortir du plateau de jeu, algorithme de parcours pour cartographier l'ensemble du plateau paramétrable pour d'autres plateau du même type
Niveau de priorité	Très Haute
Objectif atteint ?	Oui

La mise en place de la cartographie se fait par l'utilisation d'un algorithme de parcours en L en fonction de la hauteur et la largeur du plateau. Le robot va donc avancer sur n cases et effectué un prélèvement de la couleur sur ce L, à chaque L effectué on va réduire de 1 la hauteur / largeur afin d'arriver correctement à la fin de la cartographie. Les angles pour tourner et la distance parcourue entre chaque case a été définie en brut.

Bluetooth

La mise en place de la communication se fait par deux classes distinctes qui sont organisées dans le package twister.model.communication, on y trouve une classe Emetteur et une classe Recepteur qui chacune possède des méthodes en fonction de ce qui va être demandé par l'acteur sur les robots.

De plus, un fichier config.properties permet de configurer au préalable du jeu les données du robots destinataires. Les données configurées dans ce fichier de config sont ensuite récupérées dans la classe Emetteur lorsque celui-ci doit envoyer une information.

Règle du jeu

Objectif	Définir les règles du jeu du twister
Description	Chaque robot doit pouvoir émettre et recevoir l'envoie d'une couleur à tour de rôle pour pouvoir savoir où naviguer. Le robot "décide" et envoie la couleur choisie au second robot
Contraintes et règles de gestion	Utiliser le protocole bluetooth, avoir un générateur de couleur aléatoire, alterner les rôles d'emetteur et recepteur à tour de rôle
Niveau de priorité	Haute
Objectif atteint ?	Oui

La classe Emetteur contient une méthode emettreJeu qui enverra la couleur générée par la méthode getRandom de la classe RegleJeu contenu dans le package twister.model
La classe ReglesJeu à une méthode permettant de générer une couleur aléatoire de la classe TwisterColor et d'obtenir le code de cette couleur, le code de la couleur est ensuite retourné.

La classe Recepteur avec la méthode recepteurJeu pourra récupérer la couleur générer pour ensuite utiliser navigation vers cette couleur.

Cartographie collaborative

Objectif	Pouvoir envoyer la cartographie établie par l'un des robots
Description	Envoie de la cartographie et des informations utile pour l'autre robot
Contraintes et règles de gestion	Prendre en compte les 2 positions de départ, envoyer un objet par le protocole bluetooth
Niveau de priorité	Haute
Objectif atteint ?	Oui

La classe Emettre contient une méthode emettreCartographie qui enverra un objet avec la méthode writeObject prenant en paramètre l'objet envoyé (ici notre cartographie) et le dataOutPutStream

La classe recepteur avec la méthode recepteurJeu pourra récupérer la cartographie.

Navigation

Objectif	Le robot doit pouvoir se déplacer dans l'environnement
Description	Le robot lors de la réception de l'instruction doit pouvoir se déplacer vers cette case de manière optimisée.
Contraintes et règles de gestion	Rester dans l'espace de jeu, prendre en compte la cartographie et les données prélevés par le robot
Niveau de priorité	Haute
Objectif atteint ?	Partiel

La navigation est un thread, ça nous permet donc de déclencher la fonctionnalité de navigation pour le robot en parallèle de l'exécution de l'Arbitrator. Notre constructeur de navigation prend en compte le robot qui va effectuer la navigation, le plateau de jeu, la couleur voulue et le game va être notifié de la navigation effectuée.

On aura ensuite une méthode run qui va récupérer la case la plus proche par rapport à la couleur générée par le deuxième robot. La case la plus proche est déterminée à l'aide de la méthode caseLaPlusProche de la position actuelle du robot. On vérifiera si le robot est déjà sur cette case. S'il n'y est pas, il va dans un premier temps se rendre à la bonne coordonnée x suivant sa position, puis à la bonne coordonnée y. S'il n'est pas bien orienté avant de se rendre sur une des bonnes coordonnées, il va tourner vers la droite jusqu'à être dans la bonne direction.

II. Résultats Obtenus

Diagramme de classe

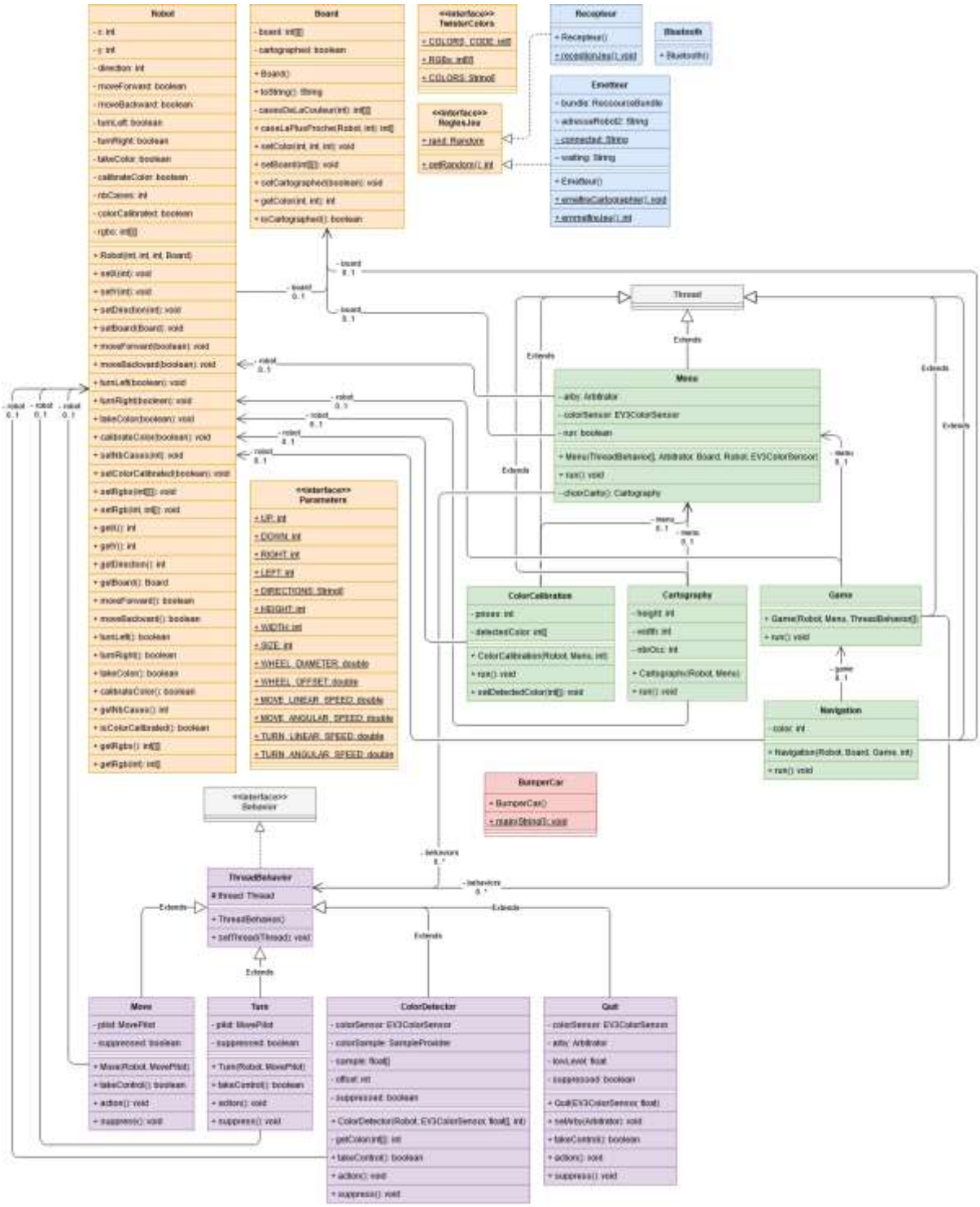
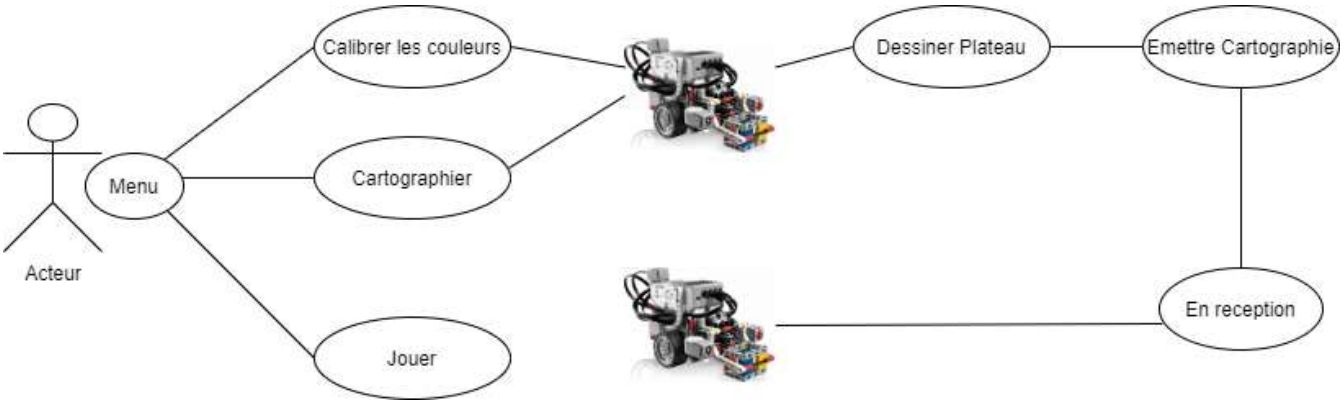


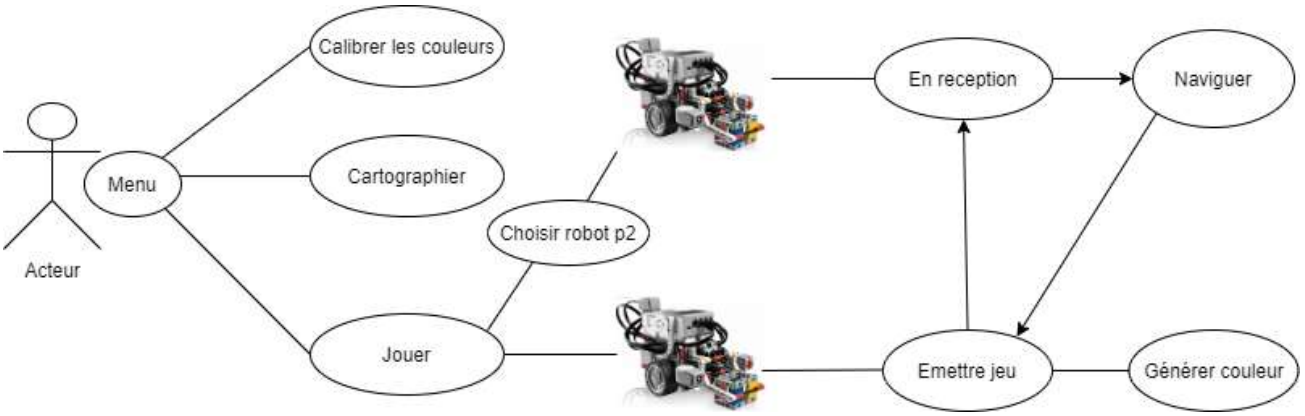
Diagramme de cas d'utilisation :

Nous avons choisi de retranscrire idéalement ce que doit faire notre application pour répondre à l'objectif. Néanmoins comme on a pu le voir précédemment certains objectifs n'ont pu être remplis que partiellement, ce diagramme de cas d'utilisation et donc fictif à comment notre programme aurait dû fonctionner.

Première partie : Calibration et cartographie



Deuxième partie : Le jeu



III. Analyse Critique :

Analyse Générale

L'ensemble du projet n'a pas été géré de la bonne manière. Pourtant dès le départ nous avions mis en place les bons outils, la bonne méthodologie en fonction de nos expériences passées. Néanmoins nous avons sous estimé l'impact qu'aurait la crise sanitaire sur notre méthodologie de travail. On peut donc constater un léger relâchement lorsqu'on a vu qu'on était dans les temps par rapport au trello et à un moment nous nous sommes épris pour un problème avec la mise en place de nos threads ce qui nous a impacté sur plusieurs séances. Derrière ce problème nous avons essayé de remplir les briques sans suivi réellement méthodologique et avons au fur et à mesure abandonner certaines de nos bonnes habitudes tel que la continuité du Trello. Nous n'avons donc pu concevoir le projet tel que nous l'imaginons à la base, la dernière séance était une sorte de mise de poker, on a rassemblé ce que chacun faisait et essayé de mettre en commun néanmoins nous avons rencontré quelques problèmes, principalement sur la navigation. Nous avions beau les avoir testées de manière isolée en amont lorsque nous les avons mises en commun nous avons eu des conflits à corriger dans l'urgence.

Analyse technique

Navigation

Dans l'état actuel du programme, Navigation ne fonctionne pas. En effet, le robot semble avoir des déplacements aléatoires et n'arrive pas à se rendre sur la case voulue. De plus, la manière dont se déplace le robot fait qu'il peut se décaler progressivement et ne plus s'arrêter au centre des cases. Cela est lié au fait que le robot donne parfois des secousses lorsqu'il démarre et que ses rotations ne sont pas toujours complètes car les roues peuvent glisser.

Nous ne comprenons pas vraiment pourquoi le robot a des déplacements aléatoires car l'algorithme utilisé fonctionne théoriquement. Cela ne peut pas être lié à une méthode annexe car le fonctionnement de caseLaPlusProche est correct car nous l'avons vérifié en amont.

Outre le problème d'algorithme, il reste une optimisation à faire car pour le moment, le robot tourne vers la droite tant qu'il n'est pas dans la bonne direction. Il faudrait parfois qu'il tourne vers la gauche si cela lui permet d'être bien orienté avec moins d'actions.

Ajout de fonctionnalités théoriques

Dans la définition du projet nous avons défini comme objectif secondaire de mettre en place les fonctionnalités évite-moi si tu peux et la minimisation du parcours.

Minimiser le parcours correction

La minimisation du parcours a été globalement effectuée, en effet on utilise une méthode case la plus proche qui va renvoyer la couleur la plus proche du robot en utilisant une autre fonction qui va elle servir à avoir une liste des cases de la couleur sélectionnée. Puis nous avons une méthode run, qui va définir la manière la plus efficace de s'y rendre. Dans l'idéal nous aurions aimé implémenter l'algorithme a* ou Dijkstra qui sont des algorithmes du plus court chemin les plus courants.

Evite-moi si tu peux

Cette fonctionnalité n'a pas du tout été mise en place néanmoins nous y avons réfléchi au préalable du projet et mis en commun nos expériences passées avec les algorithmes de plus court chemin et d'élément "obstacle". Dans l'idéal nous aurions implémenté un algorithme A* pour minimiser le parcours qui est célèbre dans les algorithmes de plus court chemin dans les jeux vidéo, nous aurions ajouté en contraintes la position x, y (que l'on définit déjà actuellement) du robot qui n'est pas en train de naviguer afin que le second robot soit pris en compte comme un obstacle à éviter lors du calcul du plus court chemin.

Bilan

Bilan de groupe :

En général nous sommes réellement satisfait de ce projet. Nous avons tous pu appréhender la programmation java dans un contexte différent et donné ainsi vit au robot. De plus chacun à participé, nous avions au départ une bonne organisation de projet et des tâches complémentaires. De plus pour chaque question que l'on se posait nous avions accés à une documentation riche et claire et également nous pouvions poser des questions à Mr. Castagnos. Les 4 dernières heures à tous travailler ensemble furent enrichissantes, nous produisions de manière efficace et logique afin de combler un retard que l'on avait accumulé. Nous en avons tous tiré quelque chose de différent pour nos prochains projets.

Bilan personnel :

Nicolas :

J'ai vraiment apprécié ce projet, qui nous a permis de (re)découvrir la programmation Java dans un autre contexte, que je trouve plus concret étant donné que l'on donne vie en quelque sorte à un robot, amené à se déplacer dans le monde physique. Le sujet était également très intéressant et permettait d'utiliser plusieurs fonctionnalités offertes par le robot EV3 Mindstorms.

Les contraintes d'accès aux robots ne nous ont pas permis de finaliser le projet mais personnellement je reste satisfait des objectifs atteints.

Valentin :

Ce projet était intéressant pour moi et ça m'a permis d'avoir une première approche de la robotique à travers un langage que je connaissais déjà "Java". On a pu utiliser la documentation Lejos et avoir accès aux robots Lejos Mindstorm grâce à Mr. Castagnos qui ne sont pas des robots qu'on pourrait avoir forcément accès à la maison. Néanmoins la contrainte sanitaire nous a amené de nombreuses complications et nous avons mal géré les contraintes de temps qui en découlait et au fur et à mesure nous avons pris du retard sur les objectifs fixés. Je reste tout de même satisfait de ce projet qui était pour l'ensemble je pense une approche intéressante de la robotique.

Tom :

J'ai trouvé ce projet très intéressant, le fait d'appliquer nos connaissances pour agir directement sur des robots a été très enrichissant. De plus, le sujet nous a permis de découvrir au fur et à mesure les différentes fonctionnalités du robot.
Grâce à Nicolas et Valentin qui m'ont beaucoup épaulé pendant ce projet, j'ai pu ajouter ma pierre à l'édifice et je les en remercie, car ils m'ont aidé à progresser.
Il est cependant dommage que nous n'ayons pas eu un accès total aux robots, mais le contexte sanitaire n'y était clairement pas favorable.

Source

Cours de Mr. Castagnos
https://stackoverflow.com/
http://www.lejos.org/ev3/docs/