Projet IA & Robotique

Rapport de projet

Tuteur:

DAMIEN Pellier

Étudiants:

AURAY Cédric

CALVIGNAC Sébastien

SIMON Dorian

KAMISSOKO Djoko

Année scolaire : 2020-2021

Plan

I) INTRODUCTION

Ce Projet est réalisé dans le cadre de l'UE "Initiation à l'Intelligence Artificiel" proposé par le département MIASHS(Mathématique et Informatique Appliquées aux sciences Humaines et Sociales) en troisième année de cursus. Cette UE a vocation à nous faire expérimenter les rudiments de l'intelligence artificielle et de la gestion de projet informatiques. Ce document a donc vocation de présenter notre travail tout au long du semestre. Pour cela nous verrons dans un premier temps les besoins et les objectifs du projet. Dans un second temps nous verrons les stratégies développées pour la gestion du projet. Dans un troisième temps nous verrons la gestion technique du projet. Et enfin dans un dernier temps nous verrons le bilan du projet. Pour conclure cette introduction j'aimerais insister sur les difficultés mécaniques rencontrées qui nous ont empêché d'exprimer le plein potentiel du travail que nous avons fourni en vue de la compétition.

II) BESOINS ET OBJECTIFS DU PROJET

Tout ce qui sera développé dans cette partie est un résumé succinct du cahier des charges. C'est pour ça qu'après chaque sous-partie il y a un renvoie au cahier des charges.

1. CONTEXTE

Ce projet d'intelligence artificielle prend la forme d'une compétition. Compétition durant laquelle deux robots s'affronteront pour savoir lequel est capable de ramener un maximum de palet dans la partie adverse d'en-but. (Concept de base, page 7)

2. LES ENJEUX

Cette compétition met en avant la capacité des robots à être autonome pour la récupération d'un maximum de palet dans un temps imparti. Pour qu'un robot puisse participer à la compétition, il doit répondre aux critères d'acceptabilités fixés par les organisateurs de la compétition. (Critères d'acceptabilité et de réception, page 10)

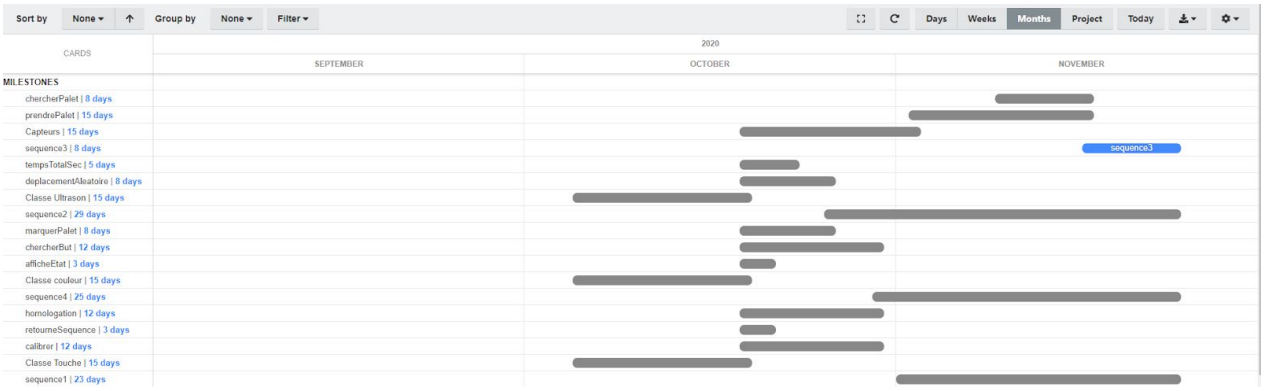
3. OBJECTIFS ET CONTRAINTES

Pour réussir à être autonome, le robot devra savoir percevoir et savoir se déplacer. Il est donc de notre ressort de faire que les capteurs transmettent des informations permettant au robot d'adapter ses déplacements.(Contraintes matérielles, page 11)
Comme il s'agit d'une compétition, nous voulons obtenir un maximum de palet dans un minimum de temps afin de gagner cette compétition. De plus, un bon classement durant cette course nous permettrait de décrocher des points bonus.
La contrainte la plus importante durant cette compétition est la contrainte de temps. Cette contrainte c'est trouver exacerbée par le confinement qui nous a empêché de travailler normalement. En effet, sur les douze séances cinq se sont déroulées durant le confinement. (Contraintes de délais, page 11)

III) GESTION DE PROJET

1. LA PLANIFICATION DE PROJET ET LES OUTILS DE GESTION

Comme dit précédemment nous avions 12 semaines de cours. Sur ces douze semaines de cours, il y a 4 rendus. Le cahier des charges en semaines trois ; le plan de développement en semaine cinq ; le plan de tests en semaine dix et enfin le code source et la documentation en semaine onze. Initialement nous avions prévu de suivre le diagramme de Gantt ci-dessous (diagramme issu du plan de développement).



Toutefois ce planning a été remis en question à cause du confinement. Celui-ci a rendu plus complexe les interactions au sein de notre groupe. Les derniers rendus ont été déplacés pour qu'on puisse se concentrer sur la partie codage.

2. RÉPARTITION DES TÂCHES ET DES TECHNOLOGIES

Initialement la répartition était décrite sur le diagramme de Gantt. Cependant cette organisation a été remise en cause par la crise sanitaire. En effet, le robot était présent chez Cédric donc seul lui pouvait tester nos méthodes. Durant le confinement nous envoyons donc les morceaux de codes développés à Cédric pour qu'il puisse les testés. Une fois le code testé, il mettait en ligne le code fonctionnel.

Ce projet informatique est réalisé en groupe et les fonctionnalités du programme peuvent être développées en parallèle par plusieurs membres de groupes. C'est pourquoi nous nous sommes servi de github pour collaborer sur le projet. Cet outil est essentiel et nous nous sommes servis de lui tout au long du projet. Bien que parfois certaines contributions ont été envoyées par message au membre du groupe qui était en possession du robot, notamment lorsque ces contributions avaient besoin d'être testé. Au paragraphe sur les perpectives nous discutons des éventuels fonctionalités de github qui peuvent servir d'antidotes à cette méthodologie non conforme.

IV) DÉVELOPPEMENT TECHNIQUE

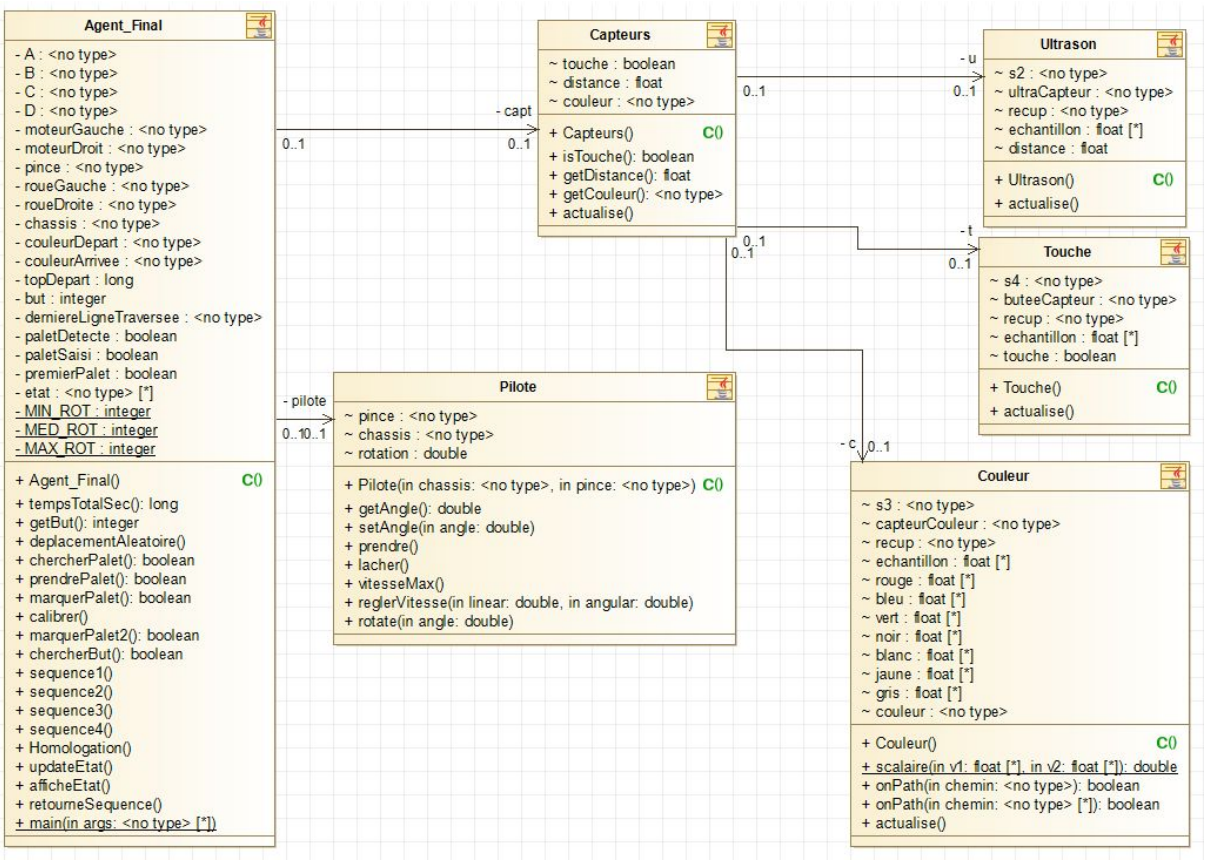
1. LA STRATÉGIE

L'idée générale de l'implémentation algorithmique était essentiellement de miser sur la résilience du système ; la capacité du robot à tenir le temps maximum sur le terrain. Le premier pré-requis était alors de proposer une classe permettant de centraliser les données en provenance des instances régissant les représentations des informations en provenance des capteurs. Ainsi qu'une méthode permettant de réaliser de nouveaux échantillons et d'autres pour pouvoir accéder à celles-ci en continu lors des déplacements. Ainsi lors des premières phases de tests nous avions une méthode de déplacement nous permettant de ne jamais sortir des limites posées par les lignes blanches du terrain.

La deuxième étape était de déterminer un ensemble d'états, une méthode pour actualiser l'état courant parmi ce champ, et une méthode permettant de retourner une séquence d'actions spécifique selon l'état courant de l'agent.

Également la proposition de surcharger la méthode rotate() de la librairie Lejos, permettant de de conserver un historique des rotations dans un attribut de l'instance Pilote. Cette solution nous permet de connaître l'orientation du robot sur le terrain. Toutefois la problématique majeure est le heurt dans les murs ; ceux-ci ne sont pas bien détectés par le capteur à ultrasons lorsque le robot s'en approche en diagonale. Ainsi l'on stocke l'information de la dernière ligne de couleur verte ou blue traversée afin de pouvoir basculer sur cette recherche si le recours à cet historique ne permet pas en un temps suffisant de trouver l'enbut adverse. Lorsque celui-ci est trouvé, une nouvelle calibration est réalisée, ce qui a pour effet de remettre cette "boussole" à zéro lorsque le robot est aligné perpendiculairement au mur de fond.

c) architecture logiciel (classes : agent, UML)


Agent_Final
- A : <no type>
- B : <no type>
- C : <no type>
- D : <no type>
- moteurGauche : <no type>
- moteurDroit : <no type>
- pince : <no type>
- roueGauche : <no type>
- roueDroite : <no type>
- chassis : <no type>
- couleurDepart : <no type>
- couleurArrivee : <no type>
- topDepart : long
- but : integer
- derniereLigneTraversee : <no type>
- paletDetecte : boolean
- paletSaisi : boolean
- premierPalet : boolean
- etat : <no type> [*]
- MIN_ROT : integer
- MED_ROT : integer
- MAX_ROT : integer
+ Agent_Final() C()
+ tempsTotalSec(): long
+ getBut(): integer
+ deplacementAleatoire()
+ chercherPalet(): boolean
+ prendrePalet(): boolean
+ marquerPalet(): boolean
+ calibrer()
+ marquerPalet2(): boolean
+ chercherBut(): boolean
+ sequence1()
+ sequence2()
+ sequence3()
+ sequence4()
+ Homologation()
+ updateEtat()
+ afficheEtat()
+ retourneSequence()
+ main(in args: <no type> [*])
- capt
0..1
0..1
Capteurs
~ touche : boolean
~ distance : float
~ couleur : <no type>
+ Capteurs() C()
+ isTouche(): boolean
+ getDistance(): float
+ getCouleur(): <no type>
+ actualise()
0..1
- u
0..1
Ultrason
~ s2 : <no type>
~ ultraCapteur : <no type>
~ recup : <no type>
~ echantillon : float [*]
~ distance : float
+ Ultrason() C()
+ actualise()
0..1
0..1
- t
0..1
Touche
~ s4 : <no type>
~ buteeCapteur : <no type>
~ recup : <no type>
~ echantillon : float [*]
~ touche : boolean
+ Touche() C()
+ actualise()
- pilote
0..1
0..1
Pilote
~ pince : <no type>
~ chassis : <no type>
~ rotation : double
+ Pilote(in chassis: <no type>, in pince: <no type>) C()
+ getAngle(): double
+ setAngle(in angle: double)
+ prendre()
+ lacher()
+ vitesseMax()
+ reglerVitesse(in linear: double, in angular: double)
+ rotate(in angle: double)
- c
0..1
Couleur
~ s3 : <no type>
~ capteurCouleur : <no type>
~ recup : <no type>
~ echantillon : float [*]
~ rouge : float [*]
~ bleu : float [*]
~ vert : float [*]
~ noir : float [*]
~ blanc : float [*]
~ jaune : float [*]
~ gris : float [*]
~ couleur : <no type>
+ Couleur() C()
+ scalaire(in v1: float [*], in v2: float [*]): double
+ onPath(in chemin: <no type>): boolean
+ onPath(in chemin: <no type> [*]): boolean
+ actualise()

2. LES SOLUTIONS LOGICIELS

Capter l'information de l'environnement

Les classes Ultrason, Touche et Couleur
La méthode actualise() implémentée dans les classe Ultrason, Touche et Couleur permet de réaliser un nouvel échantillon via le capteur physique correspondant et stockera l'information après une transformation analogique dans l'attribut de classe dédié. Particularité pour l'instance de Couleur qui réalisera un échantillon et le comparera avec les 7 échantillons précedemment réalisés à la calibration (dans le constructeur de Couleur) pour déterminer par un calcul de la différence et stocker la String correpondant à la couleur déterminée dans l'attribut de classe.

La classe Capteurs
Permet de centraliser les informations en provenance des 3 instances précédentes. L'appel à actualise() appelle actualise() sur Ultrason, Touche et Couleur, et récupère les valeurs d'attributs de ces instances via des getters pour les mettre à jour dans les attributs de Capteurs.

Via actualise() et les getters de Capteurs, l'information est accessible en continu et sur une modalité de fréquence amplement suffisante pour percevoir l'information avec une précision suffisante, hormis le cas particulier des murs en plexiglas perçus sur la diagonale.

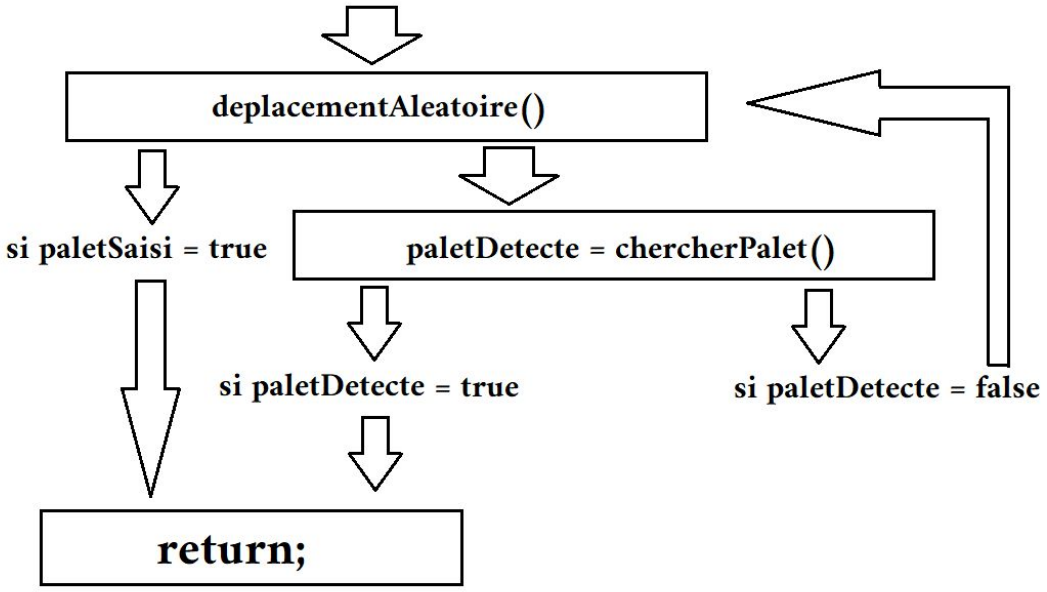
Etats, mise à jour de l'état courant et retour de la séquence d'actions
Le tableau de String 'etat' permet de représenter l'ensemble des 4 états possibles + l'état courant à l'indice 0.
La méthode updateEtat() permet de mettre à jour l'état courant, en plaçant l'état déterminé à l'indice 0 du tableau. Cela est réalisé par 3 conditionnelles successives s'appuyant sur des boolean attributs de la classe Agent_Final, représentant des configurations spécifiques (paletPremier vaut true tant que l'appel à la première séquence "en dur" n'est pas réalisé ou échoué ; paletDetecte vaut true si un objet est perçu en face du robot, paletSaisi si un palet est en possession des pinces du robot)
retourneSequence() permet de retourner la séquence d'actions appropriée selon l'état courant déterminé.

Ainsi à l'exécution, il s'agit de réaliser dans une boucle (avec condition d'arrêt sur le temps et le nombre de palets marqués) un appel successif à updateEtat(), puis à retourneSequence(). Le robot déterminera à chaque temps les actions à réaliser pour aboutir à la résolution prévue de son état courant. Les échecs des séquences induisent la mise à jour des attributs et donc de l'état courant, qui fera repartir le robot sur une phase de déplacement aléatoire et de recherche sur le terrain.

Les séquences d'actions

sequence1()
Permet l'appel à Homologation(). Ce codage "en dur" permet de récupérer et de marquer le premier palet en avançant en ligne droite jusqu'à activer le capteur de pression, serrer les pinces, effectuer une rotation légère et avancer jusqu'à l'en-but pour déposer le palet. Une sécurité est posée dans le cas où le robot perçoit une ligne blanche ou une valeur de distance trop réduite (considérée comme un autre robot ou un mur) avant l'activation du capteur de pression durant le déplacement en ligne droite. Dans les deux cas, paletPremier vaut maintenant false et l'appel à updateEtat() dans le main de Agent_Final ne retournera plus cet état comme état courant. L'état courant devient "rechercheAleatoire", retournant sequence2() par appel à retourneSequence dans le main.

sequence2()



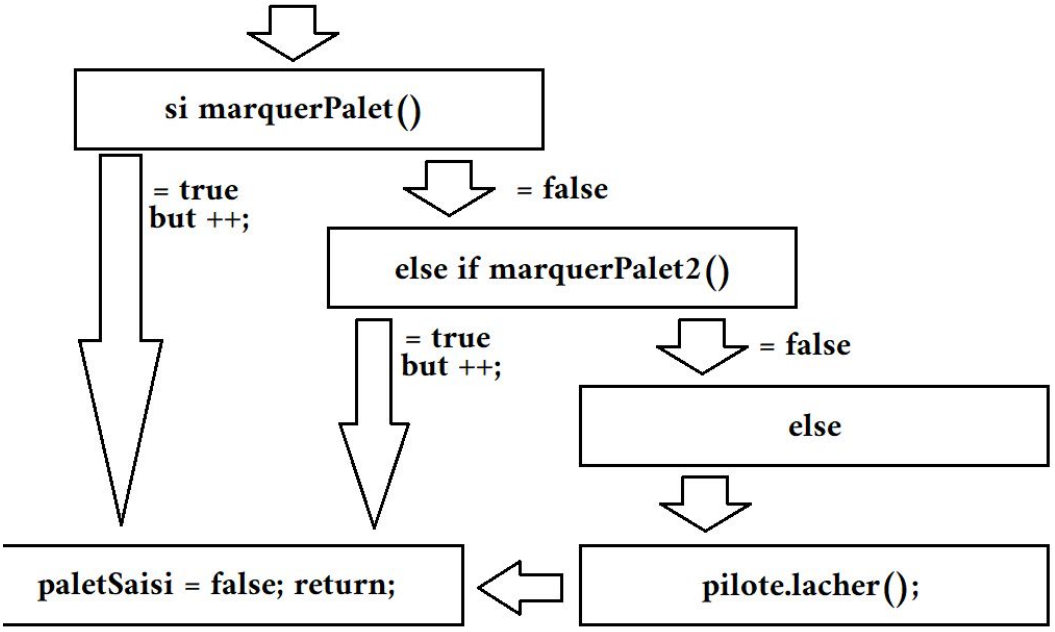
D'après cette schématisation du fonctionnement de la méthode ;

- L'on réalise deplacementAleatoire() qui permet de se déplacer d'une distance variable dans les limites du terrain, puis d'effectuer une rotation aléatoire en 0 et 360°.
- Puis l'on cherche un palet avec l'appel à chercherPalet(), qui retourne true ssi un objet est détecté autour du robot dans un rayon de 80 cm environ. Dans ce cas paletDetecte() vaut maintenant true et l'on quitte l'exécution de sequence2(), updateEtat() retournera l'état courant correspondant à la séquence permettant d'attraper le palet détecté devant le robot.
- Tant qu'un palet n'est pas détecté et que paletDetecte n'est pas mis à jour avec un retour valant true de chercherPalet(), on effectue un appel à deplacementAleatoire() puis chercherPalet().
- Si un palet touche le capteur de pression par hasard durant l'exécution de deplacementAleatoire(), on stoppe le déplacement, l'on serre les pinces et l'on quitte l'exécution. Mise à jour l'attribut paletSaisi qui vaut true.

- Si paletSaisi vaut true, on quitte l'exécution de la séquence, updateEtat() dans le main retournera sequence4() correspondant à la séquence permettant de marquer le palet.

sequence3()
Permet d'aller chercher le palet perçu, à priori situé devant le robot. Si prendrePalet() réussit et renvoie true, paletSaisi vaut maintenant true et paletDetecte false. Si échec, paletDetecte vaut false. Arrêt de l'exécution après cet appel unique.

sequence4()



Premier appel à marquerPalet() :
- si renvoie true, on actualise l'attribut but en ajoutant la valeur 1, paletSaisi vaut maintenant false et l'on quitte l'exécution de la séquence.
- si renvoie false, l'on considère que le cumul des différences entre l'historique théorique des rotations effectuées et la valeur réelle est trop importante et ne permet plus de trouver l'en-but adverse dans un temps imparti pour marquer ; alors l'on réalise un appel à marquerpalet2() qui s'appuie sur l'historique de la dernière ligne verte ou bleue rencontrée pour déterminer l'en-but cible.
    - si renvoie true, on actualise l'attribut but en ajoutant la valeur 1, paletSaisi vaut maintenant false et l'on quitte l'exécution de la séquence.
    - si renvoie false, on desserre les pinces, paletSaisi vaut maintenant false et l'on quitte l'exécution de la séquence.

V) BILAN DU PROJET

DÉFI ET DIFFICULTÉS RENCONTRÉES

Contraintes sanitaires

En Mars 2020, la France fut confrontée à la crise sanitaire mondiale causée par le covid-19, une nouvelle maladie infectieuse respiratoire apparue en décembre 2019 en chine. Cette crise a entraîné un premier confinement du pays pendant 55 jours et un deuxième confinement à partir du 29 octobre 2020. Et ce deuxième confinement a eu un impact sur le déroulement de ce projet.

En effet, l'accès à la salle machine était restreint donc pas d'accès à la table de compétition afin de réaliser des tests. D'autres parts, la mise en place du travail à distance était difficile car ce projet nécessitait la collaboration et la contribution de l'ensemble du groupe. De plus, les échéances avec le professeur étaient à distance avec une contrainte de temps de 15 min par semaine. Malgré ces contraintes, nous avons pu nous adapter à cette situation et accomplir les missions qui nous ont été assignées.

Batterie

Alors que nous étions confiant la veille de la compétition, le jour j nous aura réservé une catastrophe. Aucun capteur ne marchait. Quelques minutes seulement avant la compétition nous sommes à la recherche de la source du problème. Lors du diagnostic des anomalies nous sommes intervenus dans le programme du robot en réduisant ses capacités au strict minimum. L'objectif étant d'une part de localiser la panne et d'autre part de réduire la dépendance du robot aux capteurs qui n'était plus fonctionnel. Nous soupçonnons que ces problèmes étaient liés au niveau de la batterie. En effet, avant le deuxième affrontement aux qualifications, nous avons eu une panne totale de batterie puisque le robot ne s'allumait plus. Pourtant le robot était chargé toute la nuit de la veille de la compétition. La leçon aura donc été de garder une batterie de secours en cas de panne.

Perspectives

Il existe des fonctionnalités de github dont nous nous sommes pas servis mais que nous pensons aurait pu nous être utile si on avait pris le temps d'en prendre connaissance. Notamment, les fonctionnalités qui relèvent du contrôle de version, les issues, les branches et les pull request.

VI) CONCLUSION

Malgré notre cuisant échec lors de la compétition, ce projet nous a permis de développer de nouvelles aptitudes. Nous avons notamment pu apprendre à rédiger des documents tels qu'un cahier des charges fonctionnel, un plan de développement ou un plan de test. De plus, nous avons pu acquérir des nouvelles connaissances techniques tels que des algorithmes de recherche classique en intelligence artificielle, la gestion de Github ou bien l'écriture de la javadoc. Enfin, l'autonomie dans laquelle nous évoluons nous a permis de nous heurter à des problèmes encore inconnus pour nous tels que la variabilité du matériel.

Si ce projet était à refaire nous ne changerions rien à ce que nous avons fait car nous avons simplement joué de malchance avec notre matériel.

VI) BIBLIOGRAPHIE

Lego. (s. d.). LEGO® MINDSTORMS® Education EV3 Intelligent Brick. https://education.lego.com. Consulté 16 septembre 2020, à l'adresse https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-intelligent-brick/45500

Lego mindstorm. (2013). *ev3_user_guide* [E-book]. https://le-www-live-s.legocdn.com/ev3/userguide/1.4.0/ev3_userguide_fr.pdf

Annexe

Schéma 1 : Séquence 1 (représentation physique)

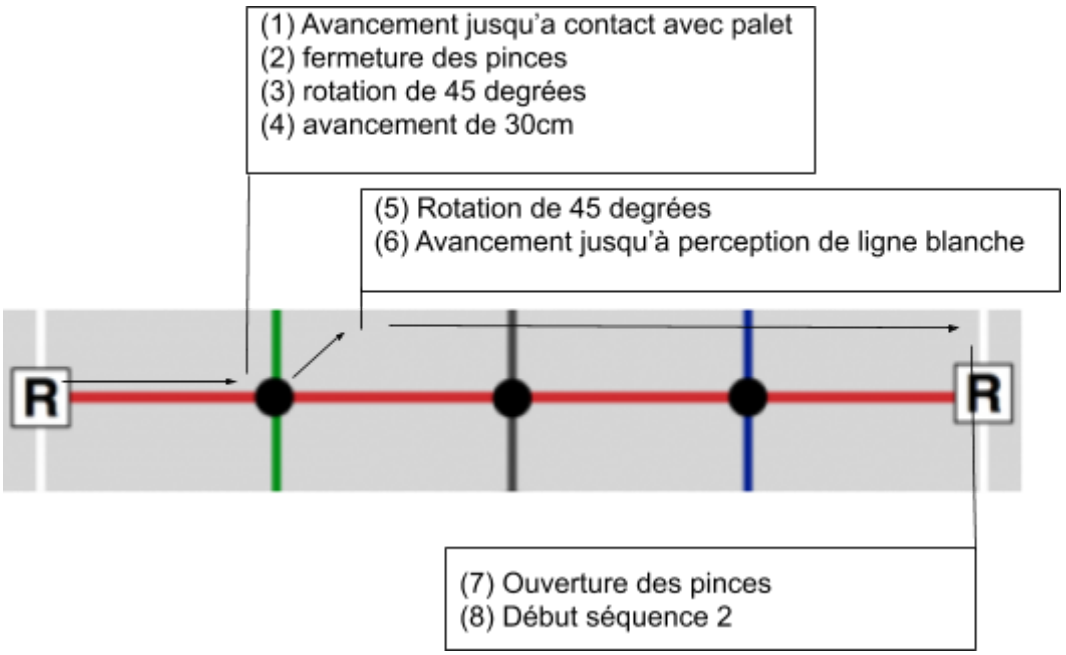


Schéma 2 : Séquence 2 (représentation conceptuel)

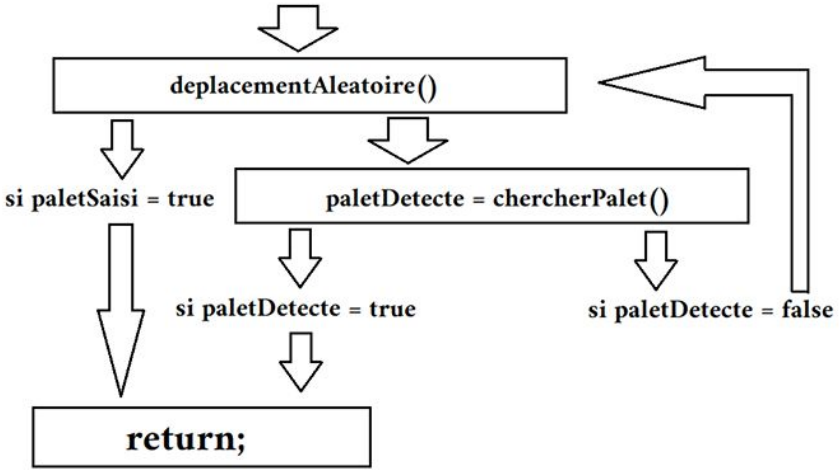


Schéma 3 : Séquence 4 (représentation conceptuel)